Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra Informačních technologií

Single Page Aplikace

S využitím MVVM frameworků AngularJS a React

Bakalářská práce

Autor: Marek Horyna
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Lukáš Vacek

Hradec Králové Červenec 2015

Prohlášení:

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

V Hradci Králové dne 17.8.2015 Marek Horyna

Poděkování:

Děkuji vedoucímu bakalářské práce Mgr. Lukáši Vackovi za odborné vedení, cenné rady, věcné připomínky, ochotu a čas strávený při konzultacích v průběhu zpracování této bakalářské práce.

Anotace

Cílem této práce je představení Single Page Aplikací, s nimi souvisejících historických milníků, které vedly ke vzniku jednostránkových aplikací, a části architektur, díky kterým je možné tento princip webových stránek využívat. Představeny budou dva nejpoužívanější nástroje – Javascript Model-View-Framework Angular, který umožňuje takové stránky vytvářet, jeho části architektury a princip fungování celého frameworku a knihovna React, která v MVC architektuře představuje vrstvu View, a slouží k vykreslování uživatelského rozhraní pomocí jednotlivých komponent. S knihovnou React bude také představen algoritmus Virtual DOM, na němž tato knihovna pracuje.

Praktickou částí této práce je porovnání výsledků měření a popsání nutných rozdílů ve zdrojových kódech při nahrazení vrstvy uživatelského rozhraní Angularu knihovnou React, a změření rychlosti vykreslení uživatelského rozhraní frameworku Angular, a Angularu, který bude mít vrstvu uživatelského rozhraní nahrazenou knihovnou React.

Annotation

Title: Single Page Apps and comparing frameworks Angular and React

The aim of this Bachelor's theses is to introduce the Single Page applications and the related milestones that led to the creation of single page applications; and the architecture parts that allows us to use this web page concept. Two most used tools will be introduced - JavaScript Mode-View-Framework Angular, which allows to create such pages, its architecture parts and the workings concept of the whole Framework; and the React library, which represents the View layer in the MVC architecture and renders user interface through individual components. Along with the React library, the Virtual DOM algorithm, on which this library works, will be introduced.

The practical part of this Bachelor's thesis will be about replacing the user interface layer of Angular framework with the React library. Then, the measurement of performance speed of rendering the user interface using the Angular framework, and compared to the Angular framework with a user interface layer replaced with React library. Comparison will be done for rendering performance speed and for differences in source codes.

Obsah

1 Úvod ... 1
2 Cíl práce ... 2
3 Metodika zpracování ... 3
4 Single Page Aplikace ... 4
4.1 Základní stavební kameny ... 4
4.1.1 XML HTTP Request ... 4
4.1.2 Asynchronous JavaScript + XML ... 4
4.2 Princip technologie ... 5
4.2.1 Výhody a usnadnění vývoje oproti klasickým webům ... 5
4.2.2 Nevýhody a časté problémy při vývoji Single Page Aplikací ... 6
4.3 Frameworky a knihovny pro tvorbu Single Page Aplikací ... 7
4.3.1 Angular ... 7
4.3.2 React ... 7
4.3.3 Meteor ... 7
4.3.4 Ember ... 8
5 AngularJS ... 9
5.1 Základní koncepty ... 9
5.1.1 Šablony na klientské části ... 9
5.1.2 Model View Controller ... 9
5.1.3 Data Binding ... 11
5.1.4 Angular Dependency Injection ... 12
5.2 Struktura Angularu ... 13
5.2.1 Scope ... 13
5.2.2 Service ... 17
5.2.3 Directive ... 18

5.2.4 Controller ... 18
5.2.5 Provider ... 19
5.2.6 Module ... 20
5.2.7 Filter ... 21
5.3 Změna view pomocí routování ... 21
6 React ... 22
6.1 Jako řešení problému ... 22
6.2 Reaktivní programování ... 22
6.3 Virtual DOM ... 23
6.4 Struktura a použití knihovny React ... 24
6.4.1 Komponenty ... 24
6.4.2 JSX rozšíření syntaxe pro JavaScript ... 25
6.4.3 Spojování a zanořování komponent ... 25
7 Využití Angularu a Reactu v jedné aplikaci ... 27
7.1 Způsob nahrazení vrstvy uživatelského rozhraní ... 27
7.1.1 Vytvoření direktivy v Angularu pro zobrazení dat v tabulce ... 28
7.1.2 Vytvoření komponenty Reactu pro zobrazení dat v tabulce ... 29
7.1.3 Napojení komponenty na Angular ... 29
7.2 Cíl výzkumu ... 30
7.3 Způsob měření výsledků ... 31
7.4 Testovací data pro měření ... 31
8 Shrnutí výsledků porovnání ... 33
8.1 Výsledky měření rychlosti ... 33
8.1.1 Výsledky měření vykreslení Angularem ... 34
8.1.2 Výsledky měření vykreslení Reactem ... 36
8.1.3 Porovnání výsledků ... 38

8.2 Syntaktické rozdíly mezi jednotlivými architekturami 41

9 Závěry a doporučení 43

10 Seznam použité literatury 44

11 Seznam příloh 45

12 Přílohy 46

Seznam obrázků

Obr. 1 Spolupráce mezi jednotlivými vrstvami v architektuře Model-View-Controller. 10
Obr. 2 Two-Way Data Binding v Angularu. 12
Obr. 3 Komunikace mezi View, Scopem a Controllerem v Angularu. 13
Obr. 4 Hierarchie modelů kopírující strukturu DOMu v Angularu. 15
Obr. 5 Nahrazení View vrstvy Reactem v architektuře MVC. 27
Obr. 6 Testovací aplikace pro měření rychlosti vykreslení webové aplikace. 32
Obr. 7 Výsledky měření rychlosti vykreslování v Angularu. 36
Obr. 8 Výsledky měření rychlosti vykreslování v Reactu. 38
Obr. 9 Porovnání výsledků měření vykreslení pro prohlížeč Google Chrome. 39
Obr. 10 Porovnání výsledků měření vykreslení pro prohlížeč Mozilla Firefox. 39
Obr. 11 Porovnání výsledků měření vykreslení pro prohlížeč Internet Explorer. 40
Obr. 12 Porovnání výsledků měření vykreslení pro prohlížeč Safari. 40

Seznam tabulek

Tabulka 1 Měření rychlosti načtení testovacích dat. 33
Tabulka 2 Měření rychlosti průchodu DOMu. 34
Tabulka 3 Měření rychlosti vykreslení aplikace Angular / Google Chrome. 34
Tabulka 4 Měření rychlosti vykreslení aplikace Angular / Mozilla Firefox. 35
Tabulka 5 Měření rychlosti vykreslení aplikace Angular / Internet Explorer. 35
Tabulka 6 Měření rychlosti vykreslení aplikace Angular / Safari. 35
Tabulka 7 Měření rychlosti vykreslení aplikace React / Google Chrome. 37
Tabulka 8 Měření rychlosti vykreslení aplikace React / Mozilla Firefox. 37
Tabulka 9 Měření rychlosti vykreslení aplikace React / Internet Explorer. 37
Tabulka 10 Měření rychlosti vykreslení aplikace React / Safari. 37

1 Úvod

Technologie webových aplikací se neustále vyvíjí a velmi aktuálním tématem jsou právě Single Page Aplikace a knihovny psané v JavaScriptu, díky kterým je možné dané webové aplikace velice jednoduše vyvinout. Mezi knihovny, které nám vývoj jednostránkových aplikací umožní, patří například Angular, který tvoří celý framework podle architektury MVC[1], nebo také React, který se soustředí na tvorbu uživatelského rozhraní webových aplikací, pomocí algoritmu Virtual DOM (podrobně popsaný v kapitole 6.3), který virtuálně v paměti porovnává aktuální DOM[2] s tím, který se má vykreslit a poté vykresluje pouze upravené části webové aplikace. Právě tyto dvě velice známé a používané knihovny jsou tématem této bakalářské práce.

Při startování nového projektu se v první řadě řeší, na jakých technologiích bude postavený. U větších projektů se často zvažují knihovny pro tvorbu Single Page Aplikací, jako například již výše zmíněné. Proto se v této bakalářské práci zaměříme na rychlost vykreslení uživatelského rozhraní Angularu. Jelikož React není kompletní framework, ale slouží pouze na vykreslení uživatelského rozhraní, nahradíme jím vrstvu uživatelského rozhraní Angularu.

[1] MVC – Zkráceny zápis pro Model-View-Framework, popsaný podrobněji v kapitole 5.1.2
[2] Document Object Model – objektově orientovaná reprezentace HTML dokumentu, tvoří strukturu webové stránky a umožňuje přístup k modifikaci reprezentace webové stránky

2 Cíl práce

Cílem této bakalářské práce je představení Single Page Aplikací, moderního stylu psaní webových stránek. Největší pozornost bude věnována především kompletnímu Model-View-Controller frameworku Angular od společnosti Google a knihovně pro tvorbu uživatelského rozhraní React od společnosti Facebook.

Praktická část bude věnována změření a srovnání rychlostí vykreslovacích algoritmů frameworku Angular a knihovny React. Jelikož React není kompletní MVC framework jako Angular, nahradíme vrstvu pro uživatelské rozhraní (View) knihovnou React a použijeme ji jako nadstavbu Angularu, a veškerá data z aplikace tak budou předávána přímo do komponent Reactu. Výsledkem práce bude jasné porovnání, který vykreslovací algoritmus je v konkrétních prohlížečích výhodnější.

3 Metodika zpracování

V této bakalářské práci jsou zkoumány dvě knihovny, nebo jejich části, které mají zodpovědnost za vykreslení uživatelského rozhraní webové aplikace. Zkoumaná knihovna Angular je navržena jako celý MVC framework pro tvorbu Single Page Aplikací. Popsán bude celý princip, na jakém knihovna pracuje a veškeré součásti, jako jsou direktivy, služby, Controllery a datové modely Scope. Další zkoumanou knihovnou bude React, která je založena na reaktivním programování[3]. S touto knihovnou bude představen i algoritmus Virtual DOM, kterým je React poháněn.

Metodicky budou porovnávány rozdíly v implementaci jednotlivých knihoven pro vykreslení uživatelského rozhraní. Aby bylo možné tyto dvě knihovny porovnat korektně, je nutné vrstvu uživatelského rozhraní Angularu nahradit Reactem. Tím docílíme toho, že pouze tato vrstva bude to, co se bude mezi testovanými metodami lišit.

Všechny prohlížeče budou porovnány a graficky tak bude možné zjistit, který prohlížeč je pro jakou metodu vývoje výhodnější z hlediska rychlosti. Porovnávána bude rychlost vykreslování po dokončení implementace nahrazení vrstvy uživatelského rozhraní. Rychlost bude měřena testovací aplikací, která bude vykreslovat několik desítek tisíc elementů do DOMu webového prohlížeče. Srovnání bude prováděno pomocí skriptu, implementovaného v testovací webové aplikaci.

[3] Reaktivní programování je princip pro psaní webových aplikací, jehož hlavní vlastností je vytváření celého zobrazovaného obsahu znovu s každou změnou

4 Single Page Aplikace

Trendy ve vývoji webových aplikací jdou neustále kupředu a jedním z posledních trendů moderní doby jsou právě Single Page Aplikace, neboli jednostránkové aplikace. Jednostránkové aplikace se jim říká z toho důvodu, že veškerý obsah webové stránky je zobrazen právě na jedné stránce, načtené webovým prohlížečem. Další zdrojové soubory, které jsou potřebné pro zobrazení obsahu uživateli, jsou stahovány dynamicky až po interakci s uživatelem.

4.1 Základní stavební kameny

První základy dynamických webových aplikací byly položeny už v roce 1994 společností Netscape, která tehdy představila technologii JavaScript. Evoluce jednostránkových webových aplikací byla podmíněna několika milníky v historii informatiky, které byly v následujících letech od vydání JavaScriptu představeny, a jejich konečné spojení umožnilo dnešní podobu jednostránkových aplikací. [1]

4.1.1 XML HTTP Request

Technologie, která se zkráceně nazývá XHR, byla představena již v roce 2000, jako tehdy ještě skoro neznámá technologie pro stahování datových souborů na pozadí načteného webového souboru. Programátoři webových aplikací měli tedy možnost programového přístupu k vlastním HTTP požadavkům přes JavaScript. [1]

Ačkoliv tato technologie má ve svém názvu XML, stahování souborů nebylo omezeno pouze na tento datový typ. Pomocí XHR bylo možné stáhnout velkou množinu různých formátů datových souborů, jako například HTML, JSON, ZIP. [1]

4.1.2 Asynchronous JavaScript + XML

Asynchronní zpracovávání[4] webových stránek pomocí knihovny, která je zkráceně označována jako AJAX, bylo poprvé představeno až 5 let po představení XHR veřejnosti. Článek „Ajax: A New Approach to Web Applications“, kde se tento termín objevil vůbec poprvé, publikoval Jess James Garrett v dubnu roce 2005. [2]

---

[4] Asynchronní zpracovávání požadavků je princip, kdy při volání serveru s požadavkem na data není server blokován pro vykonávání ostatních požadavků

AJAX ve skutečnosti není samostatná technologie, ale pouze pojem označující použití několika technologií dohromady za účelem nějakého cíle. Tento přístup k tvorbě webových stránek byl prosazován v komerční sféře nejvíce společností Google a produkty Gmail, Google Groups, Google Maps, a dalšími. [2]

4.2 *Princip technologie*

Základním principem technologie jednostránkových webových aplikací je načtení jednoho konkrétního webového souboru. S prvním přístupem k tomuto souboru jsou načteny všechny další zdroje definované v hlavičce HTML dokumentu. Mezi těmito zdroji mohou být CSS soubory pro „nastylování“ jednotlivých částí HTML dokumentu, nebo také JavaScript soubory, které dokáží na příkaz, nebo po načtení dokumentu, vykonat konkrétní úkoly.

Všechny další zdroje jsou načítány během interakce uživatele s webovou stránkou. Takovou interakcí se může rozumět např. pohyb kurzorem po obsahu webové stránky, posun scrollbaru, nebo například stisk tlačítka. Tyto zdroje jsou načítány pomocí již zmíněné technologie XHR, konkrétně tedy principu AJAX.

Interakcí uživatele s webovou aplikací může být pomocí JavaScriptu ze serveru stažen další obsah (multimédia, HTML kód, text apod.), který je načten do paměti prohlížeče a posléze při překreslení zobrazen místo jiného obsahu, který byl na této stránce doposud. Tím pádem můžeme uživateli „pod rukama“ vyměnit obsah části webové stránky, aniž bychom museli znovu načíst a vykreslit celý dokument. Oproti klasickým webovým stránkám tak uživatele připravujeme o probliknutí bílé obrazovky při přechodu z jedné stránky na druhou. Stejným způsobem mohou být načítány i CSS[5] soubory, nebo dodatečné JavaScript soubory.

4.2.1 Výhody a usnadnění vývoje oproti klasickým webům

Obecně se dá říci, že tato technologie je určena pro sofistikovanější webové aplikace, jež mají určitou logiku, kterou je nutné vyhodnocovat na webovém prohlížeči.

---

[5] CSS (Cascading Style Sheets) popisují jak zobrazit různé (X)HTML/XML elementy v rámci dokumentu

Mezi výhody jednostránkových aplikací tak můžeme zařadit následující: [1]

- Není nutné stále načítat ta samá data. Z toho vyplývá zmenšení datového toku od uživatele k serveru a také naopak ze serveru k uživateli.
- Aplikace načítá pouze data, která jsou v danou chvíli nutná k vykonání požadavku uživatele.
- Šablonování je usnadněno díky vyměňování určité části dokumentu a jeho následného vyrenderování v dalším cyklu vykreslování.
- Velmi výhodné z hlediska rychlosti interakce uživatele je, že při vývoji jednostránkových aplikací je možné ukládat načtené soubory do paměti, nebo na pozadí načítat soubory, které uživatel může navštívit.

4.2.2 Nevýhody a časté problémy při vývoji Single Page Aplikací

Mezi největší nevýhodu této technologie patří pravděpodobně fakt, že celá webová aplikace je poháněna právě JavaScriptem, který některá ze zařízení nemusí vůbec podporovat, ačkoliv poměr takovýchto zařízení je zanedbatelný a stále klesá. Mezi hlavní nevýhody můžeme zařadit: [1]

- Pomalé počáteční načtení webové aplikace, které je způsobeno tím, že jsou nejprve načteny soubory a až potom je JavaScript zpracovává a následně je vykresluje uživateli do prohlížeče.
- Optimalizace pro vyhledávací enginy, která může být u některých projektů složitější z toho hlediska, že většina textových souborů využívaných v jednostránkových aplikací bývá dotahována dynamicky, a tím se „zneviditelní“ vyhledávacím botům.
- Možnost problémů s historií a tlačítkem ZPĚT u webových prohlížečů u některých aplikací, které nejsou napsané podle daných standardů.

4.3 Frameworky a knihovny pro tvorbu Single Page Aplikací

Za posledních několik let byla představena spousta frameworků[6], které usnadňují vývojářům vývoj jednostránkových webových aplikací. Každý z nich má svoje výhody i nevýhody a správný výběr frameworku závisí především na požadavcích vyvíjeného projektu.

4.3.1 Angular

V dnešní době je Angular pravděpodobně nejoblíbenější framework pro vývoj jednostránkových aplikací. Původně byl vytvořen slovenským programátorem Miško Heverym, který projekt následně udržoval a rozvíjel pod záštitou společnosti Google. První verze Angularu byla vyvinuta v roce 2009, mezi komunitu se Angular začal dostávat až v průběhu roku 2010. Tento framework je zařazován pod architekturu MVVM (Model-View-ViewModel) nebo MVC (Model-View-Controller). Architektura MVC je detailněji popsána v kapitole 5.1.2. [3]

4.3.2 React

React byl vytvořen Jordanem Walkem, software engineerem společnosti Facebook. Při vývoji tohoto frameworku byl ovlivněn technologií XHP[7], která byla rovněž vytvořena společností Facebook, a to především tvorbou HTML komponent pro PHP. [4]

4.3.3 Meteor

Meteor byl poprvé představen na konci roku 2011 společností Meteor Development Group. Jeho hlavní charakteristikou je produkování multiplatformního kódu pro web, Android a iOS. Ve frameworku je integrovaná spolupráce s databází MongoDb, díky které není potřeba řešit jakoukoliv synchronizaci dat, ta zásluhou tohoto frameworku probíhá automaticky. [5]

---

[6] Framework je sada/balík knihoven, funkcí, programů, které usnadňují vývoj softwaru
[7] Technologie, která umožňuje XML zápis přímo v programovacím jazyce PHP

4.3.4 Ember

Tento framework byl vytvořen členem jQuery týmu Yehuda Katzem a představen také na konci roku 2011. Nyní je udržovaný Ember Core týmem. Stejně jako Angular, je zařazován pod architekturu MVC (Model-View-Controller). [6]

5 AngularJS

Framework, který v dnešní době nabývá stále více na popularitě, je AngularJS (v textu také jednoduše jako Angular). Jedná se o javascriptovovou knihovnu, která umožňuje vývoj rozsáhlých Single Page Aplikací.

5.1 *Základní koncepty*

V Angularu je použito několik základních konceptů a technologií, které vývojáře při vývoji v tomto frameworku provází od začátku až do konce.

5.1.1 Šablony na klientské části

Standardní vícestránkové webové stránky pracují tak, že nejdříve spojí HTML dokument s daty na serveru a až potom pošlou hotovou stránku prohlížeči na vykreslení. Angular a ostatní jednostránkové aplikace naopak vezmou data (včetně např. textů, fragmentů HTML kódu, obrázků apod.), a pošlou je na klientské zařízení, aby se obsah v šabloně nahradil až v prohlížeči. V tomto případě je role serveru tedy taková, že se stará pouze o zprostředkování všech požadovaných HTML šablon a také správných dat k těmto šablonám. [8]

Proměnné v šabloně jsou reprezentovány jako dvojité složené závorky, mezi nimiž je název proměnné. V ní také může být odkaz na objekt a jeho vlastnost přes tečkovou konvenci. V příkladu je také použitý nestandardní atribut ng-repeat, který je součástí Angularu, a říká frameworku, že tento element má být zopakován pod sebou tolikrát, kolik je v poli items položek. [8]

```
<table>
   <tr ng-repeat ="item in items" class="overview-info">
          <td class="name">{{item.name}}</td>
   </tr>
</table>
```

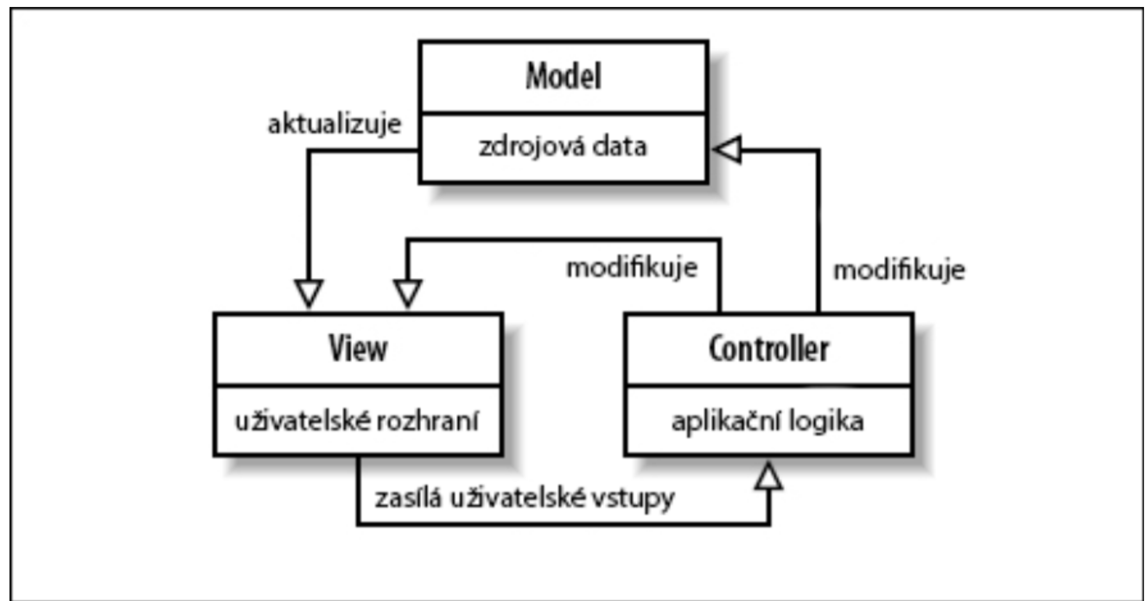
Příklad č. 1 – HTML šablona obohacená o proměnné z Angularu

5.1.2 Model View Controller

Model-View-Controller architektura byla představena už v roce 1970 jako část nástroje Smalltalk a stala se velmi populární při téměř každém návrhu a vývoji aplikace, kde bylo využíváno některé z uživatelských rozhraní. [8]

Základní myšlenkou architektury MVC je, že zdrojový kód vyvíjené aplikace je rozdělen do tří různých částí: [7]

- **Model** – Část kódu, která má k dispozici všechna potřebná zdrojová nebo databázová data.
- **Controller** – Část kódu, která vykonává veškerou aplikační logiku a obsluhuje vstupy zaslané přes uživatelské rozhraní.
- **View** – Část kódu, která zobrazuje data uživateli jako uživatelské rozhraní aplikace, přes které může uživatel zasílat vstupy zpět do aplikace.



Obr. 1 Spolupráce mezi jednotlivými vrstvami v architektuře Model-View-Controller.
Zdroj: vlastní zpracování

View dostává z Modelu data, která má zobrazit společně s uživatelským rozhraním. Uživatel, který využívá aplikaci postavenou na architektuře MVC, může přes vrstvu View vkládat svá vstupní data do aplikace, která jsou po odeslání události dále poslána ke zpracování do vrstvy Controller. Vstupními daty můžeme rozumět nejen vyplněná políčka ve formuláři, ale také události jako jsou stisknutí tlačítka myši, nebo klávesnice. Controller potom v závislosti na typu akce modifikuje Model, a tím pádem i aplikační data. Tím, že Controller změní data v Modelu, také modifikuje View, které po změně dat v Modelu bývá okamžitě překresleno, aby měl uživatel k dispozici vždy ta nejnovější data. [8]

Za View je u webových aplikací standardně považován Document Object Model (dále již jen DOM), který bude v této práci často zmiňován. Vyjímkou nejsou ani aplikace vytvořené pomocí Angularu. DOM jinými slovy označuje celou stromovou strukturu HTML dokumentu, a zároveň datovou strukturu používanou ve většině XML parserů. Controller je třída napsaná v JavaScriptu a data Modelu jsou uložena ve vlastnostech objektu.

5.1.3 Data Binding

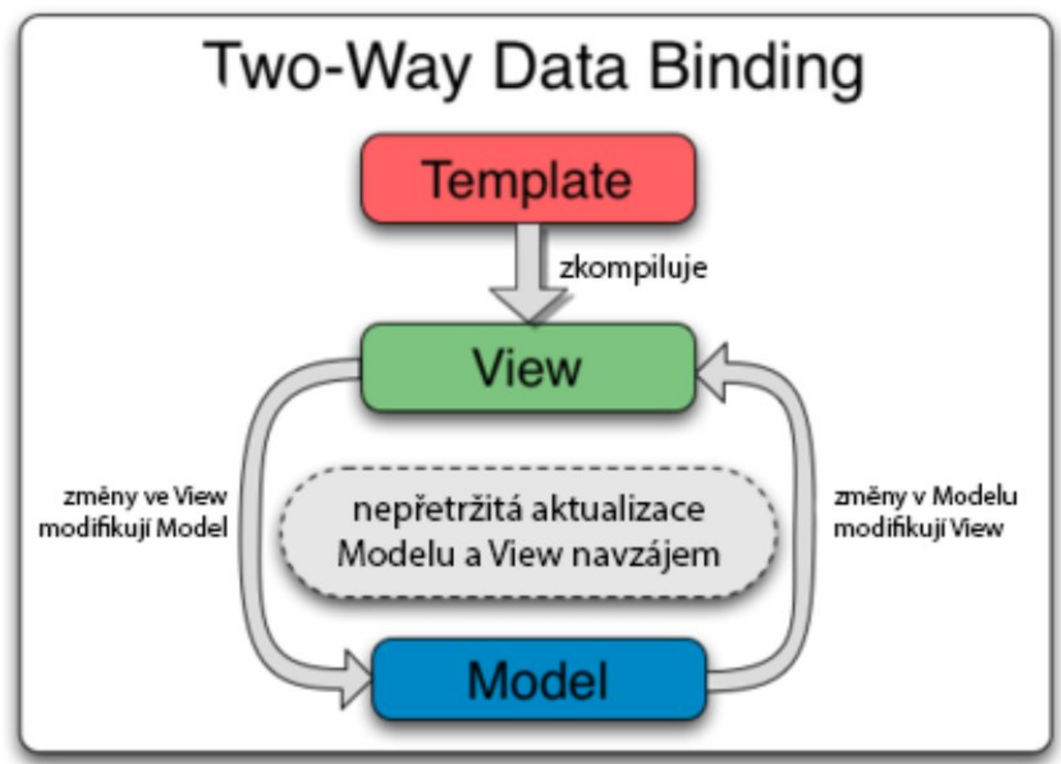
Předtím, než byly technologie AJAX a jednostránkové aplikace běžné, platformy jako PHP, Rails nebo JSP pomáhaly vytvářet uživatelské rozhraní tím, že spojovaly řetězce dat s HTML dokumentem předtím, než byl prohlížeči odeslán k zobrazení uživateli. Knihovny jako jQuery potom rozšířily tento klientský model a daly tak vývojářům možnost aktualizovat části DOMu jednotlivě, bez aktualizace a překreslení kompletně celé webové stránky. To lze udělat spojením HTML šablony s daty a následovně vložením výsledku kamkoliv do již zobrazeného DOMu nastavením innerHtml na elementu, do kterého chceme výsledek zobrazit. [9]

Angular výše popsané dělá bez jakékoliv nutnosti psaní kódu. Všechna tato funkcionalita je ve frameworku již zabudovaná a není nutné psát další nadstavby v aplikaci. Stačí pouze nadefinovat místa, kde se budou data zobrazovat a při aktualizaci dat v Modelu aplikace se tato data automaticky aktualizují.

Většina šablonovacích systémů zobrazuje data pouze v lineárním (tzv. One-Way Data Binding) směru. To znamená, že HTML šablona přidá požadovaná data z Modelu do dokumentu a zobrazí dokument v prohlížeči. Po provedení této akce nedochází k automatickému reflektování změn z Modelu do View a naopak, žádné změny, které uživatel provede ve View, nejsou propsány ani do Modelu. Tento styl propojení Modelu a View nutí vývojáře, aby stále, po každé události a každé změně, synchronizovali View a Model, a tím psali zbytečné množství aplikačního kódu navíc. [9]

Angular na rozdíl od výše zmíněného postupu využívá obousměrné propojení (tzv. Two-Way Data Binding) Modelu s View aplikací. To znamená, že v případě, že se data změní na Modelu, je tato změna reflektována na View k uživateli a naopak, při změně ve View aplikace (například při interakci uživatele

s aplikací – změna hodnoty v text boxu), se tato změna následně propíše zpátky do Modelu. [7]



Obr. 2 Two-Way Data Binding v Angularu.
Zdroj: GOOGLE A.S. AngularJS API Docs [online]. 2014 [cit. 2015-07-20]. Dostupné z: https://docs.angularjs.org, vlastní zpracování

5.1.4 Angular Dependency Injection

Dependency Injection je software design pattern[8], který napovídá jak odebrat třídám zodpovědnost za získávání objektů, které potřebují ke své činnosti a místo toho jim předávat služby při vytváření. [7]

Takový podsystém, který je implementován jako tento pattern, má v sobě i knihovna Angularu. Jejím úkolem je vytváření komponent, najití jejich závislostí a poskytnutí je ostatním komponentám, které je požadují. Jak používat dependency injection si ukážeme v následujících kapitolách.

---

[8] Software Design Pattern – návrhový vzor pro programátory, určující nejosvědčenější metody, jak řešit časté problémy při vývoji softwaru

5.2 Struktura Angularu

Za hlavními stavební kameny Angularu by se daly považovat View, Controller a Scope.

Obr. 3 Komunikace mezi View, Scopem a Controllerem v Angularu.
Zdroj: GOOGLE A.S. AngularJS API Docs [online]. 2014 [cit. 2015-07-20]. Dostupné z: https://docs.angularjs.org

View představuje HTML dokument, který bude uživateli zobrazen po navštívení aplikace přes webový prohlížeč. Dokument je obvykle nejprve načten jako statický HTML soubor a až po inicializaci Angularu doplněn o vyhodnocená data.

Controller je nejčastěji definován přímo v HTML formou direktivy Angularu, nebo v hlavním JavaScript souboru, který definuje start aplikace. Stará se o aplikační logiku aplikace, tedy vyhodnocení dat, která mají být zobrazena uživateli ve View, a také o akce během interakce uživatele s aplikací.

Scope je označován jako spojnice mezi View a Controllerem. V architektuře MVC tento objekt představuje Model. Obsahuje tedy veškerá data dané komponenty, která má definovaný vlastní Model. [7]

5.2.1 Scope

Tento termín v Angularu popisuje objekt, který v architektuře MVC představuje Model. Obsahuje veškerá data svého objektu, stejně tak i svých nadřazených objektů, které daný Scope vytvořily. Tyto Modely jsou ve webové aplikaci hierarchicky uspořádány a kopírují tak DOM strukturu aktuálního HTML dokumentu. [7]

5.2.1.1 Tvorba nového modelu a dědičnost

Zaregistrováním a spuštěním aplikace Angular se vytváří první Scope. Tento první Scope, předchůdce všech neizolovaných Modelů se nazývá `$rootScope`. Popis, jak se registruje nová aplikace, bude popsán v dalších kapitolách. Všechny další Scope jsou vytvářeny přes `$injector` v každé z komponent aplikace, jako jsou například Controllery nebo direktivy.

Každý Scope, který je vytvořen v rámci dokumentu, standardně nevidí, co jeho potomci (resp. další zanořené Scope) mají nadefinované v jejich Modelu. Potomci, kteří jsou v rámci aplikace vytvořeny, mohou být dvojího typu: [7]

- **Child Scope** dědí všechny proměnné i funkce z jeho předchůdců.
- **Isolated Scope** nedědí od předchůdců nic, tím je tedy vytvořen úplně nový Model.

Většina Modelů je vytvořena automaticky tak, jak je renderován HTML dokument, který se má zobrazit ve View. V případě, že je ale nutné někde vytvořit vlastní Scope ručně, je možné z nějakého existujícího Scope zavolat metodu `$new`, která vytvoří nový, poděděný Scope. [7]

5.2.1.2 Watchers a sledování změn v modelu

Představme si, že máme již nějaký existující Model, který je propagován na View a uživatel tento model nějakým způsobem upravuje, a propaguje nám tak jeho vstupní data do Modelu. Angular má tu možnost sledovat veškeré změny v Modelu, a reagovat tak na různé změny v uživatelově interakci s aplikací.

Je tedy možné definovat metodou `$watch`, takzvaného „listenera“, který každým průchodem aplikace testuje, jestli se daná hodnota, kterou listener sleduje, změnila nebo zůstala stejná. V případě, že se hodnota změní, se vyvolává funkce definovaná listenerem. [7]

```
$scope.$watch('inputName', (newVal, oldVal) => {
    // oldVal je původní hodnota
    // newVal je nová hodnota zadaná uživatelem

    $scope.inputName = removeWhiteSpaces(newVal);
});
```

Příklad č. 2 – Registrace listenerů $watch

5.2.1.3 Hierarchie Scope

Při vytváření nového Scope v Angularu je obecně nastavena dědičnost z rodičovského Modelu. To znamená, že dochází k převzetí Modelu ze všech nadřazených Modelů, které jsou v hierarchii nad vytvářejícím se Modelem. Výjimka existuje v případě takzvaného „Isolated Scope“, resp. izolovaného Modelu, kdy vytvářený Scope nemá žádný rodičovský Model. **Izolovaný model** vzniká v případě vytváření nové direktivy, která má definovaný svůj vlastní Model (bude více popsáno podrobněji v následujících kapitolách). [7]

Při vytváření podřazeného Scope nastává proces, který vezme celý Model rodiče a vytvoří nový Model se stejnými vlastnostmi, které poté „zakrývají“ vlastnosti rodiče. Při používání těchto vlastností potom není přistupováno k rodiči, ale ke konkrétnímu Modelu, který musí svá data následně zpropagovat do rodičovských Modelů. Tato dědičnost je vykonávána přímo JavaScriptem, nikoli Angularem. [7]

```
<body ng-app="angularWithReactApp">
    {{applicationTitle}}
    <div ng-controller="applicationCtrl">
        {{applicationVersion}}
        <div ng-controller="navigationCtrl">
            <ul>
                <li ng-repeat="item in navItems">{{item}}</li>
            </ul>
        </div>
        <div ng-controller="contentCtrl">
            {{content}}
        </div>
        <div ng-controller="formCtrl">
            some static content..
        </div>
    </div>
</body>
```

```
$rootScope(applicationTitle)
└── applicationCtrl - $scope
    (applicationTitle, applicationVersion)
      ├── navigationCtrl - $scope
      │   (applicationTitle, applicationVersion, navItems[])
      ├── contentCtrl - $scope
      │   (applicationTitle, applicationVersion, content)
      └── formCtrl - $scope
          (applicationTitle, applicationVersion)
```

Obr. 4 Hierarchie modelů kopírující strukturu DOMu v Angularu.
Zdroj: vlastní zpracování

V hierarchii Scope u každé Angular aplikace existuje právě jeden hlavní Scope, který je nadřazený všem ostatním modelům. Tento Scope je označován jako již zmíněný `$rootScope` a všechny vlastnosti definované na `$rootScope` jsou pak zpropagovány do všech podřazených Modelů. [8]

5.2.1.4 Životní cyklus Scope

Životní cyklus Modelu tedy začíná až se vznikem jisté komponenty, která má svou vlastní logiku. Touto komponentou může být například direktiva (podrobněji

popsaná v kapitole 5.2.3) nebo celý formulář dokonce i stránka. Každá tato komponenta si vytváří vlastní Scope, v němž si další komponenty, které tato nadřazená komponenta obsahuje, vytváří také vlastní Scope, který je poděděný od toho rodičovského. Celý životní cyklus modelu má těchto pět fází: [7]

1. **Vytvoření Scope** - Má na starosti $injector. Po vložení $scope do některé z komponent aplikace se pro daný objekt vytvoří nový Model.
2. **Registrace watcherů** - Probíhá po registraci modelu k objektu, a ten sleduje celý model a propaguje potom hodnoty modelu do DOMu.
3. **Propagace Scope** - Bývá vykonávána během bloku $apply při vykonávání nějaké konkrétní logiky. Angular veškerou logiku volá právě v tomto bloku, takže většinou žádné další extra volání $apply není potřeba řešit.
4. **Pozorování Scope** - Na konci bloku $apply, volá Angular svoji metodu $digest. Ta propaguje změny v Modelu skrz celou hierarchii Modelů až nahoru do `$rootScope`. Během tohoto cyklu jsou veškeré pozorované vlastnosti registrované ve watchers kontrolovány, zda u nich nenastala nějaká změna.
5. **Destrukce Scope** - Když potomek Modelu není dále potřebný, tak tvůrce (rodič) Modelu má zodpovědnost odstranit znovu tento Model a uvolnit tak prostředky, které byly alokovány jeho vytvořením. Po destrukci Modelu se odebere průchod z cyklu $digest, a dovolí tak garbage collectorem zpracovat paměť zbylou po odstraněném Modelu.

5.2.1.5 Předávání dat mezi Scope

V Angularu existuje tzv. propagace eventů mezi jednotlivými modely. Model jednoduše vytvoří událost, naplní ji daty a událost potom odešle určité části jeho rodičů nebo potomků. Model, který má registrovaný listener pro událost s tímto identifikátorem, si ji odchytí a zpracuje. Existují dva typy odesílání událostí: [7]

- **$broadcast** odešle událost do všech Modelů, které jsou ve stromové struktuře pod ním až k listům.
- **$emit** odešle naopak událost všem Modelům, které jsou jeho předchůdci a to až po $rootScope.

Životní cyklus události začíná po zavolání jedné z těchto dvou metod pro propagaci události a končí po odregistrování propagace během propagování, nebo po dokončení propagace do všech Modelů, které mohou čekat na tuto událost. [7]

Další Model může čekat na událost z jiného Modelu. K registrování obsluhy události slouží metoda $on, která má definovaný název události, na kterou čeká a metodu pro obsluhu události.

```
app.controller("applicationCtrl", function ($scope, $http, $timeout) {
    $scope.$on('dataSourceUpdated', (event, data) => {
        // čekáme na event dataSourceUpdated
        $scope.content = data.newContent;
    });
});
```

Příklad č. 3 – Definice Controlleru a metody $on, která čeká na vyvolanou událost

5.2.2 Service

Service (služba) je jedno z obecných slov, které mají v každém kontextu úplně jiný význam. V Angularu „service" označuje službu zprostředkovávající nějaký konkrétní objekt. Může se jednat o celý objekt, který má na starosti vykonání konkrétního úkolu, může se jednat o službu definovanou jako singleton[9], která přistupuje k datům ze serveru, a další. Vytvořenou službu potom lze v jednotlivých komponentách zaregistrovat pomocí Angular Injectoru. [7]

5.2.2.1 Vytváření a registrace služeb

Angular má ve svém frameworku naimplementovanou škálu služeb, které jsou využívány k řešení běžných problémů při vývoji webových aplikací, jako jsou například správa cookies, přístupy do local storage, volání http requestů na server.

Vývojáři však mají možnost vytvořit si vlastní služby, které budou dělat to, co oni potřebují, a tuto službu poté mohou použít i v jiných částech aplikace. Každá vytvořená služba je poté registrována přímo u modulu, ke kterému patří. Při registraci služby musí být poskytnuto jméno služby a factory funkce služby.

[9] Singleton je návrhový vzor, popisující globální instanci třídy, ke kterou přistupuje celá aplikace

Factory funkce vytváří instanci služby, ve které musí být pomocí Angular Injectoru vloženy všechny objekty, na nichž je vytvářená služba závislá a pomocí těchto závislostí je vytvořena instance služby.

5.2.3 Directive

Jednou z nejlepších vlastností Angularu je možnost psát znovupoužitelné HTML šablony nebo rozšiřovat logiku existujících HTML elementů pomocí takzvaných direktiv. V předešlých kapitolách byly direktivy několikrát zmiňovány.

Direktivy se poté používají jako samostatné HTML elementy, nebo pouze jako atributy navázané na existující HTML elementy. Pokud HTML kompilátor Angularu, který vykresluje zobrazení na View, narazí na element, který je definovaný v Angularu, nebo element s vlastností, která je definována v Angularu, nahradí pak část DOMu šablonou z té konkrétní direktivy.

```
<div ng-controller="formCtrl">
    <complex-form-component age="loadedAgeFromServerVariable" />
</div>
```

Příklad č. 4 – Použití direktivy v HTML šabloně

Angular má spoustu direktiv implementovaných již ve frameworku, jako například `ng-controller, ng-model, ng-class, ng-app`.

5.2.3.1 Vytváření a registrace direktiv

Direktiva je vytvářena podobně jako služba. Její registrace probíhá na úrovni definování modulu a vyžaduje normalizovaný název direktivy a factory funkci, která je popsána v kapitole 5.2.2.1 Vytváření a registrace služeb.

5.2.4 Controller

Controllerem se rozumí JavaScript konstruktorová funkce, která využívá části Angular Scope. V architektuře MVC představuje stejnojmennou vrstvu a stará se výhradně o aplikační logiku dané aplikace, nebo její části, kterou má Controller na starost. Controller může být inicializován dvěma způsoby: [7]

- **Direktivou** v DOMu – Pomocí direktivy ng-controller lze nastavit Controller na konkrétní HTML element.
- Při **definici stavu aplikace** – Při definici routování a „podstránek" aplikace, které mění svůj obsah v závislosti na interakci s uživatelem. Routování bude podrobněji popsáno v kapitole 5.3.

Po inicializaci controlleru Angular vytvoří novou instanci objektu za použití Controller konstrukturové funkce. Nový Scope, který se v rámci toho Controlleru vytvoří, je pak k dispozici jako služba `$scope` na vložení do Controlleru přes Dependency Injection.

Podle dokumentace Angularu by Controller měl být používán: [7]

- Inicializace prvotních hodnot a nastavení Modelu $scope.
- Definování chování k Modelu $scope a definice chování při změnách hodnot v Modelu.

Dále také uvedla, k čemu Controller rozhodně nepoužívat: [7]

- **Manipulace s DOMem** – Controller by měl obsahovat výhradně business logiku, ne ovlivnění toho, co bude uživatelovi zobrazeno za DOM konstrukci.
- **Formát vstupů** – Formát vstupů by měl být definovaný přímo input elementem, ne v Controlleru formuláře.
- **Formát výstupů** – Změna formátu u výstupu. Pro tento případ má Angular připravené filtery, které se definují přímo v DOMu.
- **Sdílení kódu nebo stavů mezi Controllery** – Část kódu, která by měla být sdílená, nebo společná více Controllerům, by měla obstarávat služba.
- **Starání se o životní cyklus ostatních komponent** – Například vytváření instancí služeb, apod.

5.2.5 Provider

Všechny výše popsané registrační metody (Controller, directive, service) jsou pouze nadstavbou registrační metody, která se nazývá provider. Provider je tedy

základ celé logiky registrace objektů k modulu, ostatní typy registrací jsou pouze lehké nadstavby. [7]

Při definici provideru je nutné dodržet požadavek, že každá provider registrační funkce musí mít definovanou metodu $get. Tato metoda je factory funkce, která je vyvolána při požadavku na vytvoření instanci providera.

5.2.6 Module

Modul může být představen jako kontejner na různé části aplikace. Díky modulům je tedy možné aplikaci rozdělit do dílčích částí, například podle typu definovaných objektů, nebo podle logické dělitelnosti aplikace.

Moduly také usnadňují znovu-použitelnost jednotlivých komponent. Jako modul můžeme definovat nějakou samostatnou komponentu, jako například balíček direktiv formulářových prvků. Tento balíček potom můžeme používat v každé Angular webové aplikaci, kterou jako vývojáři vytváříme. [7]

Každá jednostránková webová aplikace psaná v Angularu má jeden hlavní modul, který je vnímán jako modul aplikace. Tento modul potom musí být definován v HTML šabloně pomocí Angular direktivy `ng-app`. Angular pak při prvním průchodu DOMu vyhledá zmíněnou direktivu a nastaví pomocí této direktivy veškerý výkonný JavaScript, který pak vykonává celou logiku pro danou část stromové struktury HTML dokumentu. Životní cyklus modulu se rozděluje do dvou fází: [7]

- **Konfigurační fáze** – Je fáze, kde všechny registrace instancí jsou shromážděny a nakonfigurovány.
- **Startovací fáze** – Je fáze, kde je možné spustit veškerou logiku, která následuje po vytvoření instancí.

5.2.6.1 Konfigurační fáze modulu

Provideři mohou být konfigurováni výhradně během konfigurační fáze modulu. Nedává smysl vytvářet nová nebo měnit stávající nastavení potom, co objekty jsou již vytvořené a připravené k použití.

5.2.6.2 Startovací fáze modulu

Fáze, kde je povoleno vykonání logiky, a která by měla být uskutečněna při inicializaci aplikace, se nazývá startovací fáze. Dalo by se o ní uvažovat jako o „Main“ metodě, kterou má téměř každý programovací jazyk jako způsob první metody, která je vykonána. Jediný rozdíl oproti ostatním jazykům je ten, že Angular může mít definovaných více těchto metod, tudíž se nejedná o jediný vstup do aplikace. [7]

5.2.7 Filter

Filtry slouží k formátování hodnot, které se mají zobrazit uživateli v prohlížeči. Mohou být používány přímo z DOMu, při vypisování hodnoty z Modelu $scope, nebo přímo ve zdrojovém kódu, v některé ze služeb.

5.3 Změna view pomocí routování

Obecně jednostránkové aplikace fungují tak, že načtou pouze první stránku a veškeré změny obsahu probíhají až na té konkrétní stránce. V Angularu máme načtenou také pouze první stránku, ale někde na té stránce máme ohraničený obsah, který budeme chtít měnit v závislosti na přechodech z jednoho odkazu na druhý. Tento problém řeší routování, jehož implementace je již součástí Angularu. Při inicializaci routování je na konec webové adresy aplikace umístěn znak #, a cokoliv za tímto znakem jsou proměnlivé parametry, které obsahují také jednu z předdefinovaných cest, podle které je pak určeno, do jaké části aplikace bude uživatel přesměrován.

Služby $route a $routeProvider registrované v Angularu stačí nakonfigurovat a celou práci poté dělají automaticky. Routování se nastavuje konfigurační fází modulu a definujeme zde všechny routy, které podporujeme, spolu s adresami na HTML šablony, které máme po inicializaci routy načíst a nakonec například i Controller, který chceme pro aplikaci používat.

6 React

Knihovna pro vytváření rychlých uživatelských rozhraní od společností Facebook a Instagram. Tato knihovna je velkým množstvím lidí zařazována v architektuře MVC pod písmeno V jako View, jelikož React nemá žádný Model, ani Controllery, ale stará se pouze o uživatelské rozhraní jako takové, tedy o renderování View pro uživatele webové aplikace.

6.1 Jako řešení problému

Knihovna React byla podle společnosti Facebook vyvinuta pro velké webové aplikace, kde se data hodně mění během času, který uživatel na té dané webové aplikaci stráví. Takové aplikace vyžadují spoustu režie, uživateli na stránce stále něco přibývá, mizí, mění se a každá tato akce vyžaduje změnu struktury v DOMu a jeho následné překreslení. [10]

Největším problémem u klasického DOMu je, že není optimalizovaný pro dynamické uživatelské rozhraní, a proto je jeho překreslování náročnou operací na rychlost. Vezměme si pro představu nějakou moderní webovou aplikaci, jako je Facebook nebo Instagram. Po delší době procházení seznamem příspěvků zjistíme, že se nám na stránce vygenerovaly již stovky příspěvků, které mohou představovat tisíce kořenů stromu v DOMu. Kdybychom například chtěli posunout všechny tyto příspěvky o několik pixelů do strany, mohla by tato akce trvat i více jak jednu sekundu, a to je na dnešní moderní webové aplikace příliš dlouhá doba. [10]

Facebook si tento problém uvědomoval, a proto přišel s novým řešením, jak tento problém vyřešit. Tím je Virtual DOM, který je základem celé knihovny React.

6.2 Reaktivní programování

Základním principem reaktivního programování je vytváření celého zobrazovaného obsahu znovu s každou změnou. Tento princip dovolí psát přehledné a dobře udržovatelné aplikace, díky své jednoduchosti. Každá změna dat vyvolá vytvoření nového Modelu jen na základě svého současného stavu. Vytváření standardního DOMu je časově náročná operace, což znemožňuje použití reaktivního principu.

6.3 Virtual DOM

Operace s prvky standardního DOMu jsou časově náročné, kvůli velkému množství vlastností prvků (více než 160). Proto se pro frameworky využívající reaktivního programování vytváří v paměti prohlížeče zjednodušený virtuální DOM z prvků s menším množstvím vlastností. Operace s tímto domem jsou mnohoslovně rychlejší.

Virtual DOM značí technologii úpravy standardního DOMu v paměti prohlížeče do jakési odlehčené abstraktní verze toho, co vidíme v okně prohlížeče. Framework potom při každém renderování vytvoří novou kopii virtuálního DOMu. Při ukládání se nejprve porovnávají a hledají odlišnosti oproti reálnému stromu a poté se změní pouze to, co je potřeba změnit, ne tedy nutně celý strom. Tento postup je znatelně rychlejší, než práce přímo s DOMem prohlížeče, protože není potřeba provádět všechny dlouhotrvající operace, které jsou při práci s reálným DOMem běžné. [11]

Tento způsob překreslování DOMu prohlížeče je rychlejší a vyplatí se jen v případě, pokud s nimi aplikace pracuje správně a efektivně. Při práci s virtuálním DOMem existují dva problémy, které je potřeba řešit: [11]

- Kdy reálný DOM překreslovat z toho virtuálního.
- Jak překreslovat efektivně a rychle.

DOM překreslujeme vždy, pokud se nám změní data. A to ta pouze data, která mají nějaký výstup pro uživatele, a ten jejich změnu dokáže nějakým způsobem zaregistrovat. Pokud jejich změnu nezaregistruje, nemá smysl mu překreslovat to, co už má v prohlížeči zobrazené. Data můžeme kontrolovat oproti změně několika způsoby. Lze je procházet v pravidelném intervalu a při prvním výskytu změny v datech udělat změny ve virtuálním DOMu a přenést ho do toho reálného, nebo je možné vyvolávat provedení změn ve virtuálním DOMu pouze v případě, kdy se nám nějaká změna dat uskuteční na základě notifikací nebo odpovědí na data ze serveru. [11]

Rychlé a efektivní překreslování je dáno několika základními podmínkami, které je nutné dodržet: [11]

- Efektivní algoritmus pro vyhledávání změn mezi virtuálním a reálným DOMem.
- Dávkování čtecích/zapisovacích operací v DOMu.
- Efektivní překreslování určitého, a pouze konkrétního kořenu DOMu.

Všechny tyto problémy za nás řeší právě React a další podobné knihovny, které umí efektivně pracovat s virtuálním DOMem.

6.4 Struktura a použití knihovny React

Struktura Reactu je pouze o vytváření znovupoužitelných komponent, které jsou používány při generování Virtual DOMu a následného renderování do DOMu klasického. Tyto komponenty se dají dále spojovat nebo zanořovat a vytvářet tak složitější komponenty složené z těch menších. To velmi usnadní znovu použitelnost kódu, testování komponent a rozdělení zodpovědnosti menším prvkům.

Použití knihovny je jen o vložení knihovny React do HTML dokumentu, případně JSX Transformeru, který z kódu v syntaxi JSX vygeneruje kód použitelný pro JavaScript.

6.4.1 Komponenty

Knihovna React je o vytváření modulárních a sestavovatelných komponent pro uživatelské rozhraní webové aplikace. Za komponentu se dá označit část HTML dokumentu, která označuje konkrétní část uživatelského rozhraní. Například se může jednat o komponentu PoložkaMenu. Jako položku menu definujeme jeden odkaz v navigaci webové aplikace. Tuto komponentu definujeme jednou a poté ji použijeme několikrát, s jiným parametrem (například urlAdresa). Sada takových komponent může být zanořena v další komponentě, například Navigace, a při vykreslování uživatelského rozhraní budeme volat pouze vyrenderování navigace pro aplikaci a vnořené komponenty se nám vytvoří (vykreslí) automaticky. Spojování a zanořování komponent probereme v následujících kapitolách. [10]

6.4.2 JSX rozšíření syntaxe pro JavaScript

React spolu se svým příchodem zavedl speciální rozšíření syntaxe JavaScriptu, která je velmi podobná syntaxi XML pro psaní zdrojového kódu Reactu. Díky tomuto rozšíření je možné psát JavaScript objekty, tedy komponenty knihovny React, pomocí zjednodušené HTML syntaxe.

```
var Greeting = React.createClass({
    render: function() {
        return (
          <div className="title">Ahoj, zdraví Tě JSX.</div>
      );
    }
});
```

Příklad č. 5 – Definice komponenty Reactu rozšířením JSX

Použití tohoto rozšíření je volitelné, ale doporučené. Doporučeno je především proto, že je stručné a je obdobou syntaxe pro definování stromových struktur s atributy. Díky XML syntaxi pro definice stromových struktur je kód čitelnější a lépe formátovaný, než klasický kód JavaScriptu. JSX nemění chování základního JavaScriptu, takže je stále s tímto rozšířením možné bez omezení použít JavaScript. Použití čistého JavaScript zápisu bez JSX nemá žádná omezení. [10]

Rozšíření JSX je volitelné, není nutné tedy toto rozšíření používat a je možné stále využívat základní JavaScript. Stačí využít metodu `React.createElement(tag, attributes, children)`, které se předá jako parametr označení HTML tagu, vlastnosti tagu a případné vnořené elementy, které jsou vyrenderovány jako vnitřní HTML vytvářeného tagu. [10]

6.4.3 Spojování a zanořování komponent

Jedna z největších výhod Reactu spočívá ve spojování a zanořování vytvořených komponent. Tvorba složitějších komponent skládáním z těch jednodušších, které využívají ty již vytvořené, nám dává možnost snadno rozdělit zodpovědnost za jednotlivě prováděné akce do samostatných menších komponent. Dostáváme tedy stejnou možnost jako při používání funkcí a tříd, kdy funkci vytvoříme jednou a na jednom místě a využíváme ji na několika místech v různých částech webové

aplikace. Takto se dá snad vytvořit i sada komponent, pro jednoduché znovu použití v dalších aplikacích. [10]

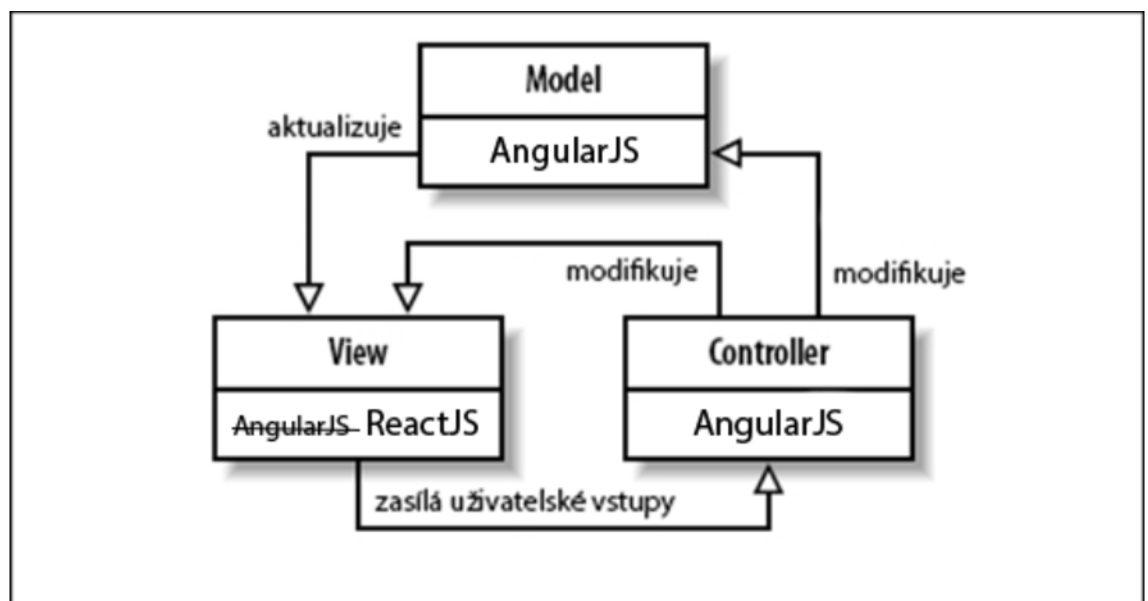
```
var myTable = React.createClass({
    render: function() {
        return (
          <table className="myTable">
            <TableHeader />
            <TableRow />
            <TableRow />
            <TableRow />
            <TableFooter />
          </table>
      );
    }
});
```

Příklad č. 6 – Použití jednotlivých komponent Reactu, které dohromady tvoří novou komponentu

7 Využití Angularu a Reactu v jedné aplikaci

Cílem praktické části této práce je zjistit, jaký zobrazovací Model je pro jednostránkové aplikace výhodnější a hlavně rychlejší. Angular je považován za kompletní MVC framework, React dokáže v této architektuře zastoupit pouze V, jako View.

Hlavní podstatou tohoto výzkumu bude nahrazení View vrstvy Angularu Reactem. Změříme zpracování a vykreslení dat pomocí samotného Angularu a poté pomocí Angularu s nahrazenou View vrstvou Reactem.



Obr. 5 Nahrazení View vrstvy Reactem v architektuře MVC.
Zdroj: vlastní zpracování

Nahrazení této vrstvy bude znamenat eliminaci šablonovacího systému, který je integrován v Angularu a předávání výsledků po zpracování dat přímo do knihovny Reactu, která se postará o přepsání Virtual DOMu a jeho následné nahrazení v klasickém DOMu webového prohlížeče.

7.1 Způsob nahrazení vrstvy uživatelského rozhraní

Způsob generování View ze šablony v Angularu probíhá na základě prvků v šabloně, které jsou definované jako šablonové prvky Angularu. Zpravidla se jedná o výpisy proměnných z Modelu pomocí dvojitých složených závorek, nebo direktivy ng-repeat, která opakuje daný element tolikrát, kolikrát se vyskytuje

v kolekci, které chceme vypsat. React naopak používá komponenty, kterým předáme data a vykreslíme je. Tato knihovna komponenty vykresluje samostatně, nebo jako větší složenou komponentu.

Jako první krok při našem výzkumu bude, že do webové aplikace přidáme reference na obě dvě knihovny, Angular a React. Do testovací aplikace, která bude využita při měření, připravíme základní části aplikace. Jelikož plánujeme z architektury MVC zachovat Model i Controller Angularu, začneme tím, že připravíme základ testovací aplikace a nadefinujeme si tyto dvě vrstvy.

Souborem app.js budeme definovat základní konfiguraci a chování aplikace, která se bude navenek tvářit jako aplikace Angularu. Prozatím si nadefinujeme strukturu, kterou později doplníme o logiku potřebnou pro náš výzkum.

```
var app = angular.module("angularWithReactApp", []);

app.controller("applicationCtrl", function ($scope) {
    // application logic here
});
```

Příklad č. 7 – Definice hlavního modulu aplikace, Controlleru

7.1.1 Vytvoření direktivy v Angularu pro zobrazení dat v tabulce

Abychom měli nějaká výchozí data, na kterých budeme moci stavět, začneme tím, že změříme, jak rychle nám zobrazí testovací data Angular. Pro testovací účely vytvoříme samostatnou direktivu, které předáme načtená data a necháme je zobrazit. Tato jednoduchá direktiva, která bude mít na starosti pouze zobrazení načtených dat, bude mít samostatný JavaScript definiční soubor a HTML šablonu.

```
app.directive('ngDataTable', function() {
	return {
		scope: {
			items: '='
		},
		templateUrl: 'app/ng/dataTable.html'
	};
});
```

Příklad č. 8 – Definice direktivy, která slouží k vykreslení tabulky

Zdrojové soubory této direktivy vytvářím do složky ng. Ta bude obsahovat všechny soubory, které budou používány pro čistý Angular. Tuto vytvořenou direktivu si později zobrazíme a necháme ji vykreslit naše testovací data.

7.1.2 Vytvoření komponenty Reactu pro zobrazení dat v tabulce

Direktivu, která nám bude zobrazovat testovací data a základní strukturu aplikace máme připravenou, ale nemáme v ní zatím použité prvky Angularu. Abychom docílili toho, že danou aplikací napojíme na vykreslovací systém Reactu, musíme vytvořením komponent, které budou naše data zobrazovat.

Při vytváření nových komponent pro větší přehlednost můžeme využít rozšíření JSX standardního jazyku JavaScript.

```
<script src="node_modules/react/dist/JSXTransformer.js"></script>
```

Příklad č. 9 – Připojení transformeru JSX ve webové aplikaci

Chceme vytvořit komponentu, která bude vytvářet stejnou DOM strukturu jako direktiva, kterou máme připravenou pro vykreslení dat pomocí čistého Angularu. Abychom využili možnosti Reactu, rozložíme tabulku do několika komponent.

První vytvořenou komponentou bude hlavička tabulky. Ta bude použita v hlavní komponentě, která bude přebírat data a vytvářet řádky tabulky. Všechny tato připravené komponenty potom zabalí do HTML elementu tabulky s požadovanou třídou a vykreslí ji.

7.1.3 Napojení komponenty na Angular

Komponenty Reactu máme vytvořené a nyní musíme vytvořenou komponentu napojit na data, která máme definovaná v Modelu Angularu. Jedním ze způsobů je vytvoření standardní direktivy Angularu, na kterou můžeme případně napojit i Controller. Jediný rozdíl oproti standardní direktivě spočívá v tom, že nebude výsledný HTML kód vytvářet Angular, ale zavoláme knihovnu Reactu a řekneme ji, kterou komponentu má vykreslit. Při volání renderování komponenty Reactu předáme také celý Model dat přímo do komponenty, která si je sama zpracuje. [12]

Komponenta bude připravena ve Virtual DOMu a následně vykreslena do klasického. Je tu ještě skutečnost, kterou nesmíme opomenout - komponenta bude vykreslena pouze jednou. A to i v případě, když se data změní, nenastane žádné automatické překreslení komponenty. Proto musíme najít způsob, jak vykreslit komponentu při každé změně dat. [12]

Existují dvě varianty. Komponentu budeme překreslovat pouze v případě změny dat. Programově je tedy nutné si ošetřit případy, kdy nastává změna dat (tedy načtení nových dat ze serveru, případně uživatelské vstupy, které mění data přímo u klienta). Tím můžeme docílit například vyvoláním události v částech kódu, která data mění a v komponentě potom pouze na tuto událost naslouchat.

```
$scope.$on('DataHasChanged', () => {
    renderComponents();
});
```

Příklad č. 10 – Registrace listeneru na událost, který po vyvolání překreslí komponenty

Druhou variantou je registrace listeneru $watch, který bude kontrolovat data předávaná do direktivy a bude čekat na změnu v datech. Problém nastane v případě, kdy aplikace obsahuje mnoho komponent, a každá má zaregistrovaný jeden nebo více listenerů. To může znamenat razantní zpomalení celé webové aplikace.

```
$scope.$watch('myData', () => {
    renderComponents();
});
```

Příklad č. 11 – Registrace listeneru na předávaná data, který po vyvolání překreslí komponenty

7.2 Cíl výzkumu

Cílem mého výzkumu je zanalyzovat přínos použití Reactu místo prezentační vrstvy architektury frameworku Angular. Porovnávat budu čistý Angular a jeho rychlost renderování tabulky o velikosti několika tisíců položek. Jedná se o netriviální tabulku, která obsahuje velké množství dat.

První krok tohoto výzkumu je zjištění, jak dlouho trvá čistému Angularu tato testovací data vykreslit do webového prohlížeče. Dalším krokem bude nahradit prezentační vrstvu Angularu knihovnou React a znovu změřit rychlost vykreslení stejných testovacích dat. Oba dva způsoby budou vykreslovat stejný počet položek a budou mít naprosto identickou stromovou strukturu DOM. Další částí bude porovnání čitelnosti kódu a syntaktického zápisu zdrojového kódu.

7.3 Způsob měření výsledků

Výsledky budeme měřit pomocí vlastního skriptu, který bude kontrolovat s každým průběhem aplikace, zda jsou data plně vykreslena. První částí měření bude vlastní čas vykonávání tohoto skriptu, ten potom od naměřených hodnot odečteme.

Načtení testovacích dat pro měření rychlosti vykreslení všech prvků ze seznamu na stránce také zabere určitý čas, ten ale nebude vlastní měření ovlivňovat, protože čas začneme měřit až s načtením dat z JSON souboru do paměti a skončíme měření až s potvrzením skriptu, že data byla úspěšně zobrazena.

Pro měření budeme mít k dispozici pět různých JSON souborů, kde každý bude obsahovat vlastní data (5000 položek). Abychom řádně otestovali funkčnost algoritmů překreslení, nebudeme měřit čas po znovunačtení stránky, nýbrž aplikačně. Controller bude připraven tak, aby po stisknutí tlačítka načetl nová data a donutil tak příslušný framework vykonat překreslení aktuální tabulky za nová data. V testovacích datech nebude žádná položka v řádku stejná, abychom vždy framework přiměli k překreslení všech řádků tabulky.

7.4 Testovací data pro měření

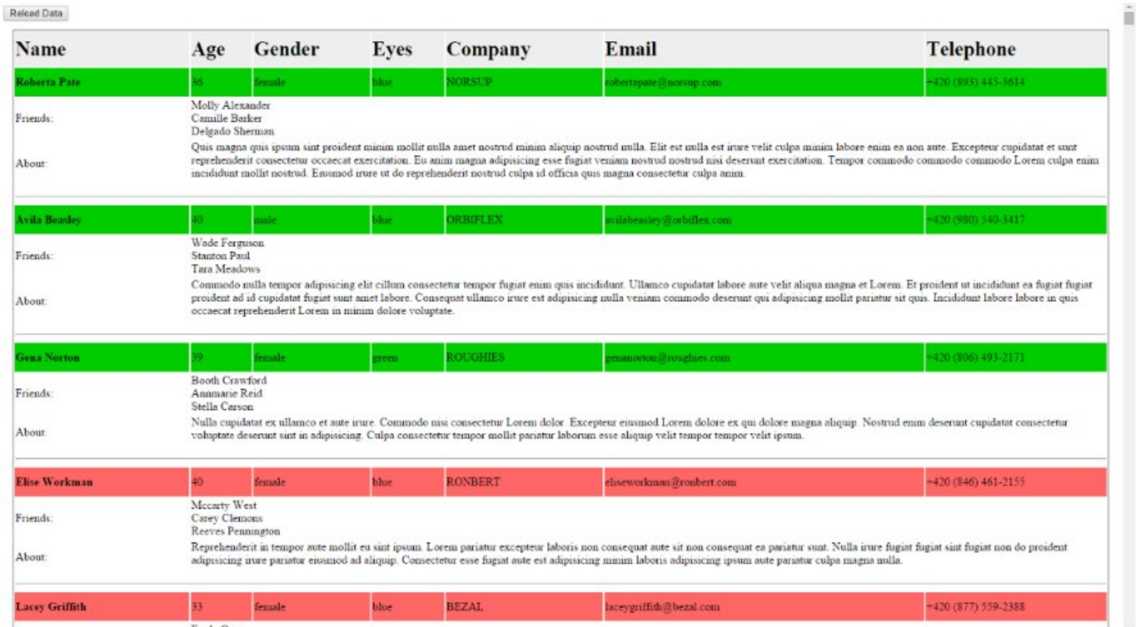
Testovací data jsou vygenerovány, náhodně zvoleným online generátorem, který data vygeneroval do formátu JSON. Tato testovací data obsahují 5000 položek a reprezentují databázi uživatelů. Testovací data jsou navržena tak, aby obsahovala relativně mnoho položek a každý objekt měl v sobě i další kolekci, která reprezentuje seznam přátel. Při vykreslování dat je tak nutné provést dvojitý cyklus procházení dat.

Data do aplikace načteme přes modul Angularu $http, který slouží k vykonávání HTTP requestů. Tato data budeme načítat s první inicializací hlavního Controlleru a uložíme je do Scope. Čas, který aplikace stráví při načítání dat ze souboru JSON, se do vlastního měření vykreslení dat nebude započítávat.

```
$http.get('sampleData/data.json').success(function(response) {
    $scope.content = response;
});
```

Příklad č. 12 –Načtení vzorových dat pomocí Angularu

Po načtení dat do testovací aplikace připravené k měření se zobrazí tabulka se všemi testovacími daty.



Obr. 6 Testovací aplikace pro měření rychlosti vykreslení webové aplikace.
Zdroj: vlastní zpracování

8 Shrnutí výsledků porovnání

Po přípravě veškerých částí webové aplikace, na které bude prováděno měření vykreslení testovacích dat, byla provedena měření vykreslení na různých webových prohlížečích. Zkoumání rychlost vykreslení dat bylo provedeno na pěti různých souborech, kde každý z nich obsahoval 5000 položek simulujících databázi osob a jeho přátel.

8.1 Výsledky měření rychlosti

Při každém vykreslení webové aplikace bylo měřeno několik konkrétních operací, které byly prováděny.

1. Načítání dat ze souboru JSON.
2. Doba vzniklá mezi načtením souboru a zobrazením klíčového slova v prohlížeči.
3. Doba prohledávání stromu, která bude posléze odečtena od doby vzniklé mezi načtením souboru a zobrazením klíčového slova.

Při testování měření rychlostí byly používány tyto počítačové konfigurace a verze webových prohlížečů:

- Google Chrome, verze 43.0.2357.134
- Mozilla Firefox, verze 37.0.2
- Internet Explorer, verze 11.0.9600.17914
- Safari, verze 8.0.7

Jako první byla měřena operace, která načítala samotná testovací data pro vykreslení renderování DOMu. Tato data jsou načítána ze souboru JSON a načtení těchto dat měl na starosti Angular.

Tabulka 1 Měření rychlosti načtení testovacích dat.

Google Chrome	Mozilla Firefox	Internet Explorer	Safari
191ms	222ms	125ms	112ms
160ms	140ms	156ms	115ms

174ms	233ms	176ms	112ms
196ms	256ms	162ms	168ms
~ 180ms	~ 213ms	~ 155ms	~ 127ms

Zdroj: vlastní zpracování

Aplikační kód, který prohledává DOM a hledá výskyt klíčového slova, které nám zaručí, že v prohlížeči byla vykreslena data z testovacího souboru, byl nejdříve rychlostně otestován na datech separovaně od samotného vykreslování, aby byla zjištěna průměrná rychlost prohledání DOMu.

Tabulka 2 Měření rychlosti průchodu DOMu.

Google Chrome	**Mozilla Firefox**	**Internet Explorer**	**Safari**
117ms	133ms	384ms	100ms
114ms	139ms	376ms	95ms
115ms	146ms	374ms	102ms
105ms	137ms	402ms	97ms
~ 113ms	~ 139ms	~ 384ms	~ 99ms

Zdroj: vlastní zpracování

8.1.1 Výsledky měření vykreslení Angularem

Nakonec bylo provedeno měření samotného vykreslování DOMu pomocí frameworku Angular. Čas byl měřen od dokončení načtení dat až po vlastní nalezení klíčového slova v DOMu. Testovací data byla po prvním měření pouze prohazována bez nutnosti znovunačtení webové stránky. Měření bylo provedeno několikrát pro každý prohlížeč zvlášť a v tabulkách jsou zobrazeny všechny prováděné operace, včetně celkového času pro vykreslení DOMu v prohlížeči.

Tabulka 3 Měření rychlosti vykreslení aplikace Angular / Google Chrome.

	Renderování	**Vyhledávání**	**Výsledek**
Prvotní vykreslení	4539ms	107ms	4432ms
2. Test data	13110ms	109ms	13001ms
3. Test data	17073ms	114ms	16959ms

4. Test data	10216ms	108ms	10108ms
5. Test data	11251ms	107ms	11144ms

Zdroj: vlastní zpracování

Tabulka 4 Měření rychlosti vykreslení aplikace Angular / Mozilla Firefox.

	Renderování	Vyhledávání	Výsledek
Prvotní vykreslení	5627ms	147ms	5480ms
2. Test data	10411ms	154ms	10257ms
3. Test data	10455ms	152ms	10303ms
4. Test data	10904ms	145ms	10759ms
5. Test data	10623ms	160ms	10463ms

Zdroj: vlastní zpracování

Tabulka 5 Měření rychlosti vykreslení aplikace Angular / Internet Explorer.

	Renderování	Vyhledávání	Výsledek
Prvotní vykreslení	17564ms	404ms	17160ms
2. Test data	34449ms	387ms	34062ms
3. Test data	38847ms	460ms	38387ms
4. Test data	39312ms	409ms	38903ms
5. Test data	39352ms	395ms	38957ms

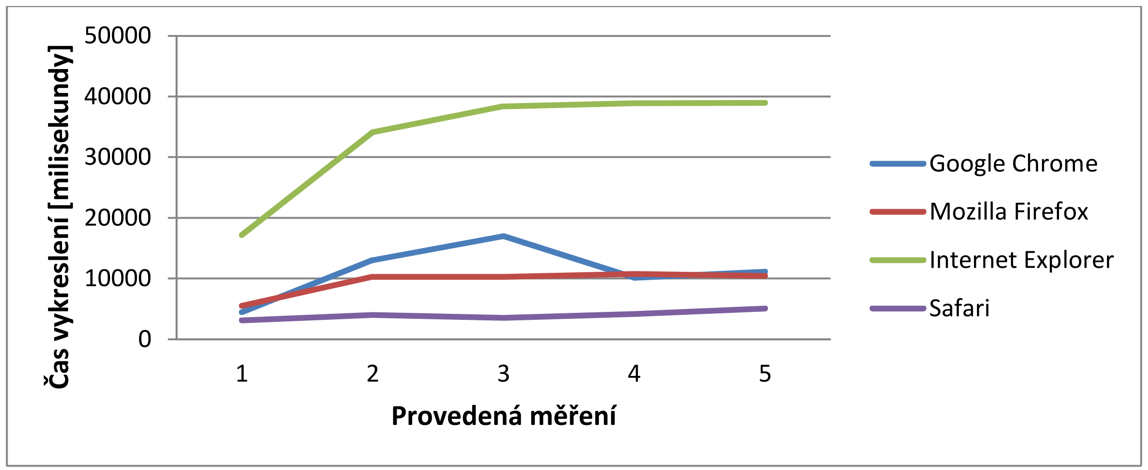
Zdroj: vlastní zpracování

Tabulka 6 Měření rychlosti vykreslení aplikace Angular / Safari.

	Renderování	Vyhledávání	Výsledek
Prvotní vykreslení	3166ms	61ms	3105ms
2. Test data	4097ms	86ms	4011ms
3. Test data	3598ms	79ms	3519ms
4. Test data	4241ms	83ms	4158ms
5. Test data	5137ms	100ms	5037ms

Zdroj: vlastní zpracování

Na naměřených hodnotách je vidět, že Internet Explorer má největší problémy s rychlostí při renderování DOM stromu, naopak nejrychleji byla data vykreslena v prohlížeči Google Chrome, ačkoliv ostatní testované prohlížeče se držely na podobných hodnotách.



Obr. 7 Výsledky měření rychlosti vykreslování v Angularu.
Zdroj: vlastní zpracování

Zajímavá je také křivka času vykreslení v jednotlivých prohlížečích. Po zhlédnutí diagramu na obr. 6 je vidět, že první vykreslení má Angular relativně rychlé, ale všechna ostatní vykreslení jsou znatelně pomalejší. Výjimkou je v tomto případě prohlížeč Safari, který drží takřka stejné hodnoty během všech měření a zároveň byl mezi testovanými prohlížeči nejrychlejší. Nyní se podíváme, jak za stejných podmínek bude fungovat React.

8.1.2 Výsledky měření vykreslení Reactem

Po připravení všech vykreslovacích komponent knihovny React byla připravena také direktiva, která danou tabulku s daty zobrazí. Ta slouží pouze jako „most" mezi Angularem a Reactem, resp. slouží k předávání dat z Angularu do komponent Reactu a zároveň volá knihovnu Reactu a požaduje po ní překreslení komponent v případě, že se změní data.

Čas vykreslení byl měřen stejným mechanismem, jako u předchozího frameworku.

Tabulka 7 Měření rychlosti vykreslení aplikace React / Google Chrome.

	Renderování	**Vyhledávání**	**Výsledek**
Prvotní vykreslení	11821ms	133ms	11688ms
2. Test data	4392ms	123ms	4269ms
3. Test data	5967ms	138ms	5829ms
4. Test data	3834ms	96ms	3738ms
5. Test data	6034ms	115ms	5919ms

Zdroj: vlastní zpracování

Tabulka 8 Měření rychlosti vykreslení aplikace React / Mozilla Firefox.

	Renderování	**Vyhledávání**	**Výsledek**
Prvotní vykreslení	10176ms	121ms	10055ms
2. Test data	7169ms	135ms	7034ms
3. Test data	6426ms	126ms	6300ms
4. Test data	6625ms	136ms	6489ms
5. Test data	4740ms	136ms	4604ms

Zdroj: vlastní zpracování

Tabulka 9 Měření rychlosti vykreslení aplikace React / Internet Explorer.

	Renderování	**Vyhledávání**	**Výsledek**
Prvotní vykreslení	32644ms	229ms	32415ms
2. Test data	31718ms	270ms	31448ms
3. Test data	31996ms	257ms	31739ms
4. Test data	31747ms	268ms	31479ms
5. Test data	31832ms	433ms	31399ms

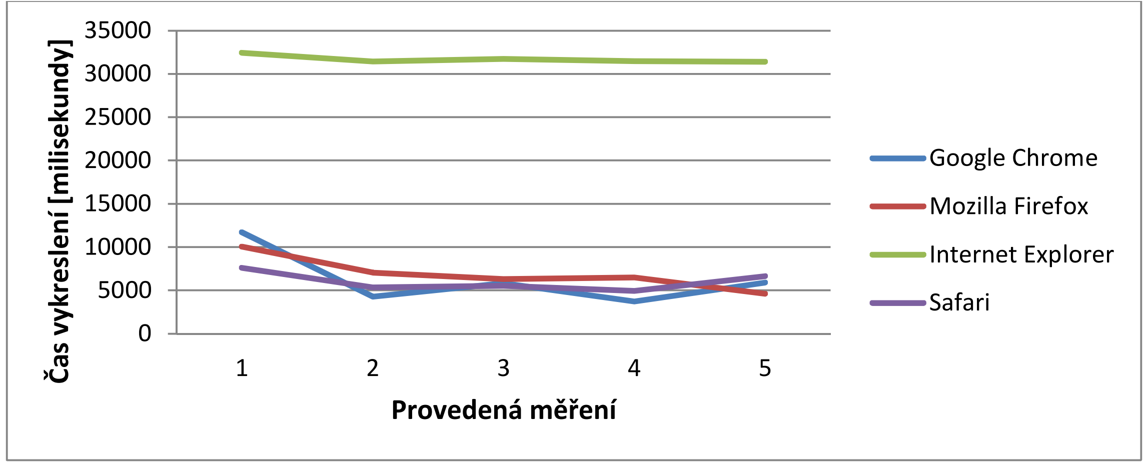
Zdroj: vlastní zpracování

Tabulka 10 Měření rychlosti vykreslení aplikace React / Safari.

	Renderování	**Vyhledávání**	**Výsledek**
Prvotní vykreslení	7640ms	45ms	7595ms
2. Test data	5388ms	47ms	5341ms
3. Test data	5587ms	48ms	5539ms

4. Test data	5024ms	68ms	4956ms
5. Test data	6731ms	77ms	6654ms

Zdroj: vlastní zpracování

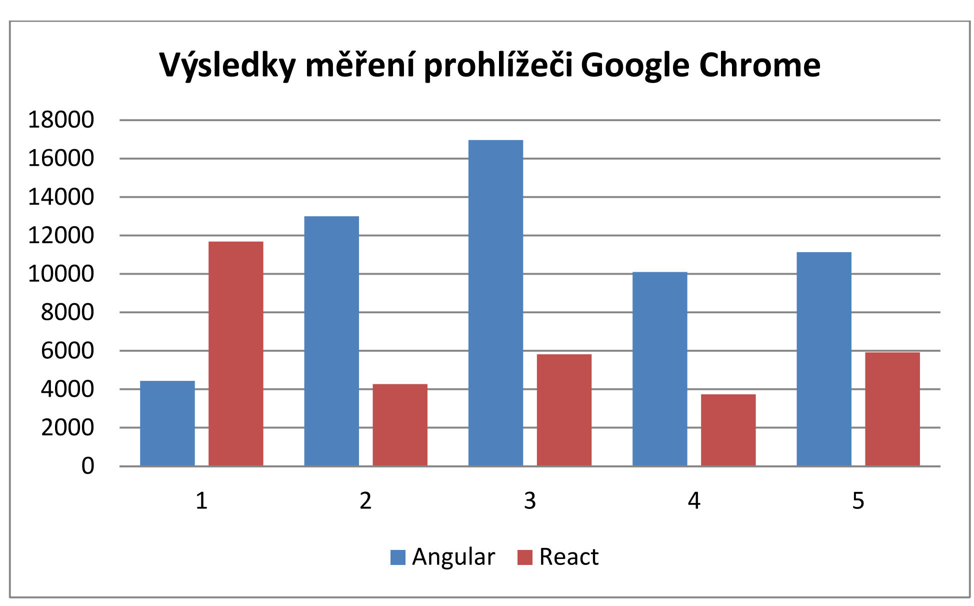


Obr. 8 Výsledky měření rychlosti vykreslování v Reactu.
Zdroj: vlastní zpracování

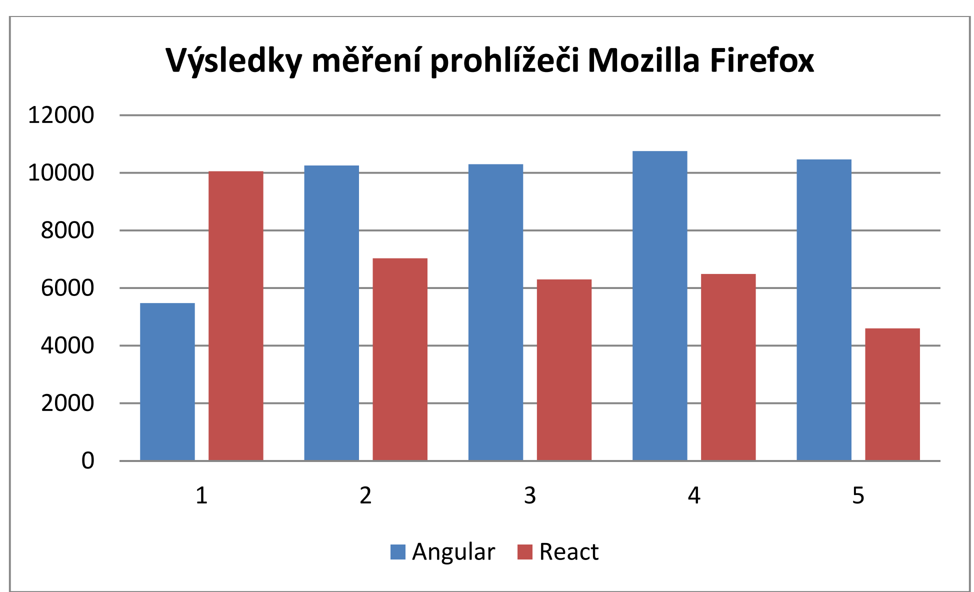
Graf na obr. 8 poukazuje na skutečnost, že první vykreslení v Reactu bývá stejně dlouhé, ne-li v některých případech delší, než u ostatních prohlížečů. Na rozdíl od předešlého frameworku se s každým dalším vykreslením dostáváme na mnohem nižší čísla, než byla ta pro prvotní vykreslení. V tomto případě je jednou výjimkou Internet Explorer, který je při každém překreslení aplikace stejně pomalý jako během prvního vykreslení.

8.1.3 Porovnání výsledků

Po srovnání jednotlivých výsledků měření Angularu a Reactu je viditelné, že prvotní vykreslení vede jednoznačně Angular. V některých případech je i více jak dvakrát rychlejší, než React. Na rozdíl od Reactu je ale s každým dalším překreslením celého DOMu tento čas až několikanásobně větší.

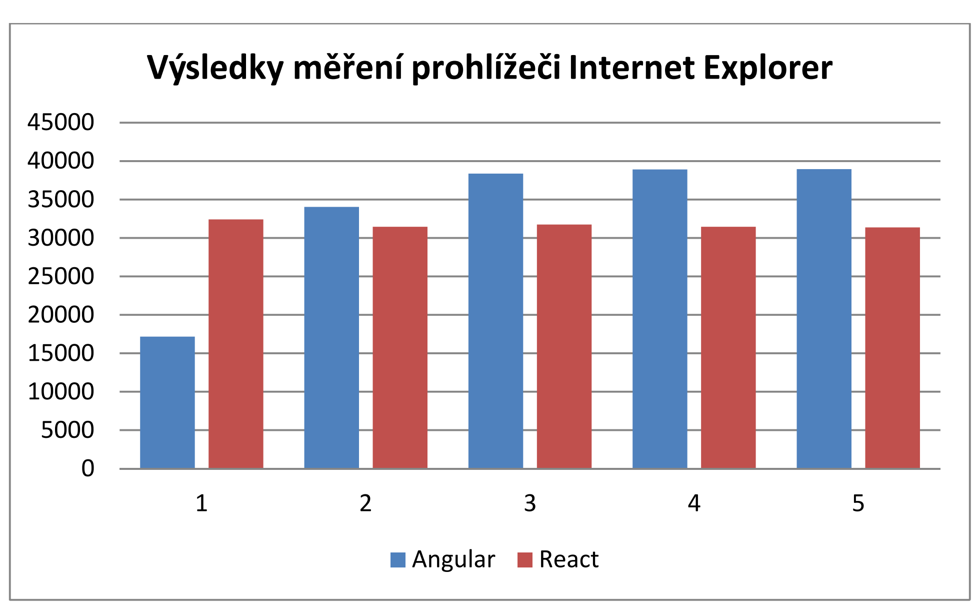


Obr. 9 Porovnání výsledků měření vykreslení pro prohlížeč Google Chrome.
Zdroj: vlastní zpracování



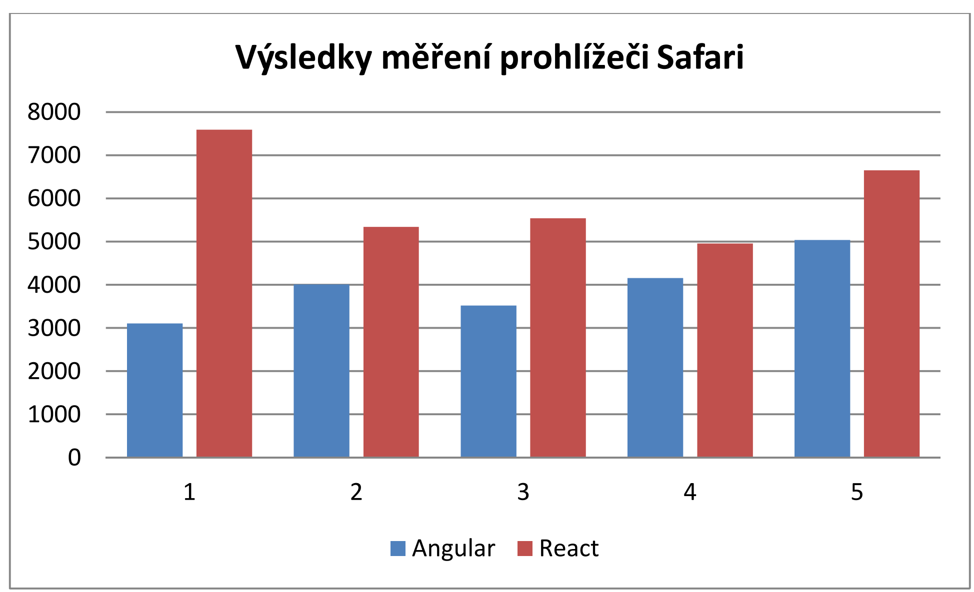
Obr. 10 Porovnání výsledků měření vykreslení pro prohlížeč Mozilla Firefox.
Zdroj: vlastní zpracování

Výsledky měření v prohlížečích Google Chrome a Mozilla Firefox jsou velmi podobné. Prvotní vykreslení vede Angular, všechna ostatní naopak React. To značí výhodu Reactu pro aplikace, které se v průběhu času často mění.



Obr. 11 Porovnání výsledků měření vykreslení pro prohlížeč Internet Explorer.
Zdroj: vlastní zpracování

U prohlížeče Internet Explorer nastal rozdíl oproti předchozím prohlížečům především u Reactu. Všechna vykreslení jsou na stejných hodnotách, včetně prvního. Celkový čas vykreslení oproti ostatním prohlížečům je mnohonásobně větší.



Obr. 12 Porovnání výsledků měření vykreslení pro prohlížeč Safari.
Zdroj: vlastní zpracování

Překvapením byl prohlížeč Safari dostupný pro počítače s operačním systémem Apple OS X. Vykreslování webové aplikace bylo v tomto prohlížeči

nejrychlejší ze všech testovaných prohlížečů a Angular v rychlosti vykreslení jednoznačně vedl a u dalších vykreslení nenastaly žádné velké rozdíly v naměřených hodnotách, jako u ostatních prohlížečů. React pro prvním vykreslení svou rychlost zlepšil, ale ani tak to na naměřené hodnoty Angularu nestačilo.

8.2 Syntaktické rozdíly mezi jednotlivými architekturami

Pokud budeme porovnávat rozdíly ve zdrojových kódech aplikace pro Angular a Angular s nahrazenou vrstvou pro uživatelské rozhraní Reactem, lze očekávat, že Angular bude mít jednodušší (jednotnější) styl zápisu, jelikož vše vychází z jednoho frameworku. Podívejme se ale, za jakých podmínek, jsme schopni zrychlit webovou aplikaci napsanou v Angularu.

Použití komponenty (direktivy) můžeme z hlediska složitosti a přehlednosti úplně zanedbat, jelikož použití je vlastně stejné. V obou případech máme nějakou direktivu a v obou případech do ní předáváme načtená data ze Scope hlavního Controlleru. Zajímat nás bude hlavně definice direktivy, která má v sobě veškeré informace o tom, jak budou data vykreslena.

V jednom případě používáme čistý Angular, výsledný kód definice je přiložen v příloze číslo 2 a 3. Definice direktivy je relativně jednoduchá, definujeme pouze data, která mají být do direktivy předána a adresu souboru, který obsahuje šablonu. Šablona je připravena, jako samostatný HTML kód s několika Angular direktivami, které řídí průběh vypisování položek v tabulce.

V druhém případě, kde vykreslovací vrstvu nahrazujeme Reactem, představuje direktiva pouze obálku, která zavolá knihovnu React. Tato obálka je tak o něco složitější, jelikož musíme načíst komponenty, které máme někde definované a zavolat knihovnu Reactu, aby nám tyto komponenty začala renderovat. Abychom zachovali standardní chování, musíme řídit překreslování dat v případě, že se změní. Běžně tuto operaci za nás dělá přímo Angular, ale vzhledem k tomu, že jsme se ho rozhodli nahradit, musíme při každé změně dat informovat React sami. Další věc, která se změní, je způsob zápisu šablony. Pro React komponenty můžeme využít standardní JavaScript, nebo JSX.

React je tedy o něco složitější, co se týče stylu zápisu a přehlednosti ve zdrojovém kódu. Je na individuálním zvážení, jestli zrychlení překreslování aplikace za tu cenu stojí.

9 Závěry a doporučení

V této bakalářské práci byly představeny Single Page Aplikace, jejich hlavní princip a historie, která umožnila vyvíjení webových aplikací na této architektuře. Předvedeny byly nejrozšířenější knihovny, které slouží k tvorbě těchto webových aplikací, z toho dvě z nich podrobně.

- Angular – kompletní MVC framework.
- React – knihovna pro tvorbu uživatelského rozhraní.

Obě dvě knihovny byly podrobně popsány, včetně principu a možností, které nabízejí. Tyto dvě knihovny jako celky nebylo možné porovnat, protože React zastupuje pouze jednu vrstvu v modelu MVC, vrstvu uživatelského rozhraní (View). Zásadní rozdíl Reactu oproti vrstvě uživatelského rozhraní v Angularu je v použití Virtual DOM algoritmu, pro nahrazování elementů v reálném DOMu webové aplikace.

Cílem této práce byla záměna vrstvy uživatelského rozhraní Angularu Reactem. Po nahrazení této vrstvy byla změřena a porovnána rychlost vykreslování testovacích dat, v pěti opakováních, pro nejpoužívanější webové prohlížeče. Výsledky nám následně ukázaly, v jakých případech je z hlediska výkonnosti výhodnější Angular, a kdy React.

Po porovnání výsledků je zřejmě, že samostatný Angular umožní snadněji a rychleji napsat menší webové aplikace. Pro velké aplikace, které se v čase často mění je z hlediska udržitelnosti výkonu výhodnější React. Nicméně díky možnosti využití Reactu z Angularu může být tato změna znatelná až ve chvíli, kdy narazíme na výkonnostní limit Angularu.

10 Seznam použité literatury

[1] Single Page Applications. Wikipedia: Free Encyclopedia [online]. 2015 [cit. 2015-07-21]. Dostupné z: https://en.wikipedia.org/wiki/Single-page_application

[2] SOTELO, Caleb. Evolution of the Single Page Application. Paislee.io: Software in context. [online]. 2015, (2) [cit. 2015-08-11]. Dostupné z: http://paislee.io/evolution-of-the-single-page-application-2-of-2/

[3] AngularJS. Wikipedia: Free Encyclopedia [online]. 2015 [cit. 2015-08-11]. Dostupné z: https://en.wikipedia.org/wiki/AngularJS

[4] React (JavaScript library). Wikipedia: Free Encyclopedia [online]. 2015 [cit. 2015-08-11]. Dostupné z: https://en.wikipedia.org/wiki/React_(JavaScript_library)

[5] Meteor (web framework). Wikipedia: Free Encyclopedia [online]. 2015 [cit. 2015-08-11]. Dostupné z: https://en.wikipedia.org/wiki/Meteor_(web_framework)

[6] Ember.js. Wikipedia: Free Encyclopedia [online]. 2015 [cit. 2015-08-11]. Dostupné z: https://en.wikipedia.org/wiki/Ember.js

[7] GOOGLE A.S. AngularJS API Docs [online]. 2014 [cit. 2015-07-20]. Dostupné z: https://docs.angularjs.org

[8] KOZLOWSKI, P. Darwin; P. AngularJS Web Application Development. New Edition. Birmingham [u.a.]: Packt Publ, 2013. ISBN 978-178-2161-820.

[9] GREEN, Brad a Shyam SESHADRI. AngularJS. First edition. x, 183 pages. ISBN 978-144-9344-856.

[10] FACEBOOK A.S. ReactJS API Docs [online]. 2013 [cit. 2015-07-20]. Dostupné z: https://facebook.github.io/react/docs/

[11] FREED, Tony. What is Virtual DOM. Tony Freed Blog [online]. [cit. 2015-07-20]. Dostupné z: http://tonyfreed.com/blog/what_is_virtual_dom

[12] NICOLA, Thierry. Faster AngularJS Rendering (AngularJS and ReactJS). William Brown Street [online]. 2014 [cit. 2015-07-20]. Dostupné z: http://www.williambrownstreet.net/blog/2014/04/faster-angularjs-rendering-angularjs-and-reactjs/

11 Seznam příloh

1) app/app.js
2) app/ng/dataTable.html
3) app/ng/dataTable.js
4) app/rt/dataTable.js
5) index-ng.html
6) index-rt.html
7) Zdrojové kódy

12 Přílohy

1) app/app.js

```
var app = angular.module("angularWithReactApp", []);

function measureRendering(searchText) {
    var count = 0;
    var renderingStart = new Date().getTime();
    var measuringFunction = setInterval(function() {
        var start = new Date().getTime();
        var source = document.getElementsByTagName('html')[0].innerHTML;
        var found = source.search(searchText);
        if (found != -1) {
            var end = new Date().getTime();
            var renderingDom = (end - renderingStart);
            var lookupDom = (end - start);
            console.log(renderingDom + "ms - rendering DOM");
            console.log(lookupDom + "ms - lookup in DOM");
            console.log((renderingDom - lookupDom) + "ms - result");
            console.log("------------------------------------------");

            count++;
            if (count >= 1) clearInterval(measuringFunction);
        }
    }, 100);
};

app.controller("applicationCtrl", function ($scope, $http) {
    var dataIndex = 1;
    $scope.switchData = function() {
        var startLoadingData = new Date().getTime();
        $http.get('sampleData/data'+ dataIndex
+'.json').success(function(response) {
            var endLoadingData = new Date().getTime();
            console.log((endLoadingData - startLoadingData) + "ms - loading
DATA");
            $scope.content = response;
            var searchText = "";
            switch (dataIndex) {
                case 1: searchText = "Dixon Griffith"; break;
                case 2: searchText = "Melendez Hicks"; break;
                case 3: searchText = "Eaton Griffin"; break;
                case 4: searchText = "Hinton George"; break;
                case 5: searchText = "Jasmine Davidson"; break;
            }
            measureRendering(searchText);
            if (dataIndex <= 4) dataIndex++; else dataIndex = 1;
        });
    };
    $scope.switchData();
});
```

2) app/ng/dataTable.html

```
<table class="usersDataTable">
   <thead>
          <th>Name</th>
          <th>Age</th>
          <th>Gender</th>
          <th>Eyes</th>
          <th>Company</th>
          <th>Email</th>
          <th>Telephone</th>
   </thead>
   <tr ng-repeat-start="item in items" class="overview-info" ng-class="{'is-
active': item.isActive, 'is-not-active': !item.isActive}">
          <td class="name">{{item.name}}</td>
          <td>{{item.age}}</td>
          <td>{{item.gender}}</td>
          <td>{{item.eyeColor}}</td>
          <td>{{item.company}}</td>
          <td>{{item.email}}</td>
          <td>{{item.phone}}</td>
   </tr>
   <tr>
          <td>Friends: </td>
          <td colspan="6"> <div ng-repeat="friend in item.friends">
{{friend.name}} <br /></div> </td>
   </tr>
   <tr class="about">
          <td>About:</td>
          <td colspan="6" class="text"> {{item.about}} </td>
   </tr>
   <tr ng-repeat-end>
          <td colspan="7"><hr /></td>
   </tr>
</table>
```

3) app/ng/dataTable.js

```
app.directive('ngDataTable', function() {
    return {
            scope: {
                    items: '='
            },
            templateUrl: 'app/ng/dataTable.html'
    };
});
```

4) app/rt/dataTable.js

```
var theadComponent = React.createClass({
    render: function () {
        return React.DOM.thead(null,
                        React.DOM.th(null, 'Name'),
                        React.DOM.th(null, 'Age'),
                        React.DOM.th(null, 'Gender'),
                        React.DOM.th(null, 'Eyes'),
                        React.DOM.th(null, 'Company'),
                        React.DOM.th(null, 'Email'),
                        React.DOM.th(null, 'Telephone')
              );
    }
});

var tableComponent = React.createClass({
    render: function () {
        var rowsData = this.props.content;
        var rows = rowsData.map(function (data) {
            var friends = data.friends.map(function (friend) {
                return [React.DOM.div(null, friend.name)];
            });

            var classString = "overview-info ";
            if (data.isActive) classString += 'is-active';
            if (!data.isActive) classString += 'is-not-active';

            return [
                  React.DOM.tr({ className: classString },
                       React.DOM.td({ className: 'name' },
data['name']),
                       React.DOM.td(null, data['age']),
                       React.DOM.td(null, data['gender']),
                       React.DOM.td(null, data['eyeColor']),
                       React.DOM.td(null, data['company']),
                       React.DOM.td(null, data['email']),
                       React.DOM.td(null, data['phone'])
                  ),
                  React.DOM.tr(null,
                       React.DOM.td(null, 'Friends:'),
                       React.DOM.td({ colSpan: 6 }, friends)
                  ),
                  React.DOM.tr({ className: "about" },
                       React.DOM.td(null, 'About:'),
                       React.DOM.td({ colSpan: 6 }, data['about'])
                  ),
                  React.DOM.tr(null,
                       React.DOM.td({ colSpan: 7 }, React.DOM.hr(null))
                  )
            ];
        });
        return (
          React.DOM.table(
            { className: "usersDataTable" },
            React.createElement(theadComponent),
            rows
          )
        );
```

```
    }
});

app.directive('rtDataTable', function () {
    return {
        scope: {
            content: '='
        },
        link: function (scope, el, attrs) {
            scope.$watchCollection('content', function (newValue, oldValue) {
                if (newValue != undefined) {
                    React.render(React.createElement(tableComponent, {
content: newValue }), el[0]);
                }
            })
        }
    }
});
```

5) index-ng.html

```
<!DOCTYPE html>
<html>
   <head>
          <meta charset="utf-8">
          <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <title>Angular with React Bachelor Thesis</title>
          <meta name="description" content="">
          <meta name="viewport" content="width=device-width">
          <link rel="stylesheet" href="app/app.css" />
          <script src="node_modules/angular/angular.js"></script>
          <script src="app/app.js"></script>
        <script src="app/ng/dataTable.js"></script>
   </head>
   <body ng-app="angularWithReactApp">
          <div ng-controller="applicationCtrl">
                 <button ng-click="switchData()">Reload Data</button>
                 <ng-data-table items="content" />
          </div>
   </body>

</html>
```

6) index-rt.html

```
<!DOCTYPE html>
<html>
    <head>
            <meta charset="utf-8">
            <meta http-equiv="X-UA-Compatible" content="IE=edge">
            <title>Angular with React Bachelor Thesis</title>
            <meta name="description" content="">
            <meta name="viewport" content="width=device-width">
            <link rel="stylesheet" href="app/app.css" />
            <script src="node_modules/angular/angular.js"></script>
            <script src="node_modules/react/dist/JSXTransformer.js"></script>
            <script src="node_modules/react/dist/react.js"></script>
            <script src="app/app.js"></script>
            <script src="app/rt/dataTable.js"></script>
    </head>
    <body ng-app="angularWithReactApp">
            <div ng-controller="applicationCtrl">
                    <button ng-click="switchData()">Reload Data</button>
                    <rt-data-table content="content" />
            </div>
    </body>

</html>
```

7) Zdrojové kódy

Zdrojové kódy jsou přiloženy na datovém DVD.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Marek Horyna**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Lukáš Vacek**

Název práce:
Single Page Aplikace s využitím frameworků AngularJS a ReactJS

Název práce v AJ:
Single Page Apps and comparing frameworks AngularJS and ReactJS

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Prozkoumat možnosti a výhody Single Page Aplikací (SPA) pro vývoj dnešních webových aplikací. Představení frameworků nejčastěji používaných pro vývoj SPA. Porovnání dvou nejpoužívanějších frameworků.

Osnova práce:
1. Úvod
2. Co je to Single Page Aplikace ?
3. Druhy frameworků
4. AngularJS
5. ReactJS
6. Syntaxe a použití
7. Rychlost renderování a porovnání výkonu
8. Další rozdíly mezi frameworky
9. Závěr

Projednáno dne:

Podpis studenta Podpis vedoucího práce