**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

MARTIN KUMECKÝ

**Návrh off-line webovej aplikácie pre riadenie ventilačnej techniky**

Vedúci práce: Ing. Matej Meško, PhD.

Registračné číslo: 57/2017

Ministerské číslo: 28360720181057

Žilina, 2018

**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

ŠTUDIJNÝ ODBOR: Informatika

MARTIN KUMECKÝ

**Návrh off-line webovej aplikácie pre riadenie ventilačnej techniky**

Žilinská univerzita v Žiline

Fakulta riadenia a informatiky

Žilina, 2018

#### [Čestné vyhlásenie](Pokyny_pre_vypracovanie_ZP.docx#Čestné_vyhlásenie)

Čestne prehlasujem, že som bakalársku prácu *Návrh off-line webovej aplikácie pre riadenie ventilačnej techniky* vypracoval samostatne na základe vlastných teoretických a praktických poznatkov, pod odborným vedením vedúceho bakalárskej práce a používal som len uvedenú odbornú literatúru nachádzajúcu sa v zozname použitej literatúry.

V Žiline Dňa:................................. Podpis:................................

#### [Poďakovanie](Pokyny_pre_vypracovanie_ZP.docx#Poďakovanie)

Chcem sa poďakovať ...

[ABSTRAKT V ŠTÁTNOM JAZYKU](Pokyny_pre_vypracovanie_ZP.docx" \l "Abstrakt)

KUMECKÝ, Martin: *Návrh off-line webovej aplikácie pre riadenie ventilačnej techniky*. [Bakalárska práca]. – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky; Študijný program: Informatika. – Vedúci práce: Ing. Matej Meško, PhD. – Stupeň odbornej kvalifikácie: Bakalár informatiky – Mesto: FRI ŽU v Žiline, 2018. Počet strán (napr. 35 s.)

[Vysvetlenie a príklad.](Pokyny_pre_vypracovanie_ZP.docx#Príklad_abstrakt)

[**Kľúčové slová**](Pokyny_pre_vypracovanie_ZP.docx#Kľúčové_slová)**:** aplikácia, web

[ABSTRAKT V CUDZOM JAZYKU](Pokyny_pre_vypracovanie_ZP.docx" \l "Abstrakt_CJ)

[Vysvetlenie.](Pokyny_pre_vypracovanie_ZP.docx#Abstrakt_CJ)

**Key words:**

[Obsah](Pokyny_pre_vypracovanie_ZP.docx#Obsah)

Zoznam obrázkov 9

Zoznam tabuliek 10

Zoznam skratiek 11

Úvod 12

1 Najpoužívanejšie reaktívne frameworky 15

1.1 Angular 15

1.1.1 História verzií 16

1.1.2 TypeScript 17

1.1.3 Inštalácia a používanie 19

1.2 React 22

1.2.1 JSX 22

1.2.2 Používanie frameworku 23

1.3 Vue 26

1.3.1 Zmeny medzi verziami 26

1.3.2 Inštalácia a používanie 27

1.4 Porovnanie 27

2 Zber požiadaviek 28

3 Analýza požiadaviek 29

4 Implementácia 30

Záver 31

Zoznam použitej literatúry 32

Zoznam príloh 33

Prílohy 34

Príloha A: Názov prílohy 35

Príloha B: Obsah DVD 36

# [Zoznam obrázkov](Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_obrázkov)

[Obrázok 1 Schéma deklaratívneho a reakčného renderovania 13](#_Toc508834004)

[Obrázok 2 Ukážka kódu v jazyku TypeScript 15](#_Toc508834005)

[Obrázok 3 Ukážka rovnakého kódu v jazyku JavaScript 16](#_Toc508834006)

# [Zoznam tabuliek](Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_obrázkov)

[Tabuľka 1 Základné príkazy v príkazovom riadku 19](#_Toc508833016)

[Tabuľka 2 Životný cyklus komponentu 20](#_Toc508833017)

# [Zoznam skratiek](Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_skratiek)

SPA Single page application

DOM Document Object Model

HTML Hypertext markup language

# [Úvod](Pokyny_pre_vypracovanie_ZP.docx#Úvod)

Kedysi, pri začiatkoch samotného internetu boli webové stránky jednoduché dokumenty, zobrazujúce poväčšine statický text, doplnený obrázkami. No postupom času sa tento statický text zmenil na dynamický, vznikli prehrávače videí, do stránok bolo možné vložiť najrôznejšie animácie a celkovo komplexnosť webovej stránky rástla. Avšak neskôr, webový prehliadač už nebol iba prostriedok, pomocou ktorého sa dostaneme k informáciám na internete. Stal sa aj zobrazovacím nástrojom pre rôzne aplikácie, využívané v rôznych odvetviach priemyslu, účtovníctve, štátnej správe a to práve z dôvodu, že aplikácie nebolo nutné inštalovať jednotlivo na každú pracovnú stanicu, ale aplikácie boli dostupné všade, kde ich bolo potreba.

Vývoj takýchto webových aplikácií bolo možné dosiahnuť aj za pomoci skriptovacieho jazyka JavaScript, ktorého popularita od svojho vzniku neustále rástla. Taktiež rástla aj komplexnosť webových aplikácií, a preto boli nároky na vývojárov, či už časové alebo vedomostné, čoraz väčšie. To v súčasnosti vyústilo do vzniku mnohých JavaScriptových frameworkov, ktoré majú za úlohu zjednodušiť a urýchliť vývoj webových stránok.

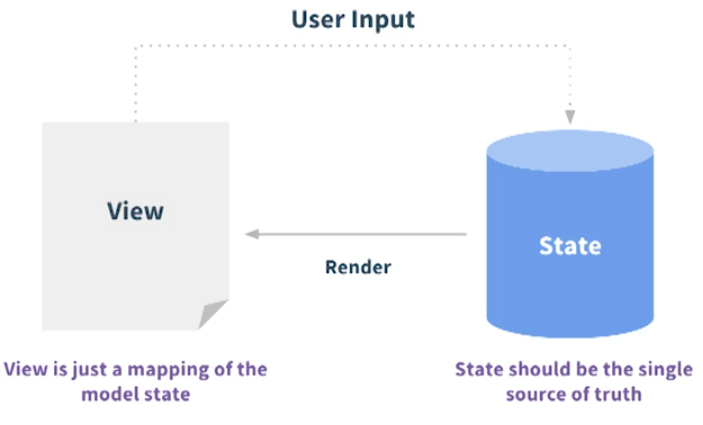
V prvej kapitole preskúmam možnosti v súčasnosti najpoužívanejších reaktívnych frameworkov. Pri skúmaní sa zameriam hlavne na zložitosť daného frameworku, jeho silné a slabé stránky. V závere kapitoly pomocou tabuľky zhrniem a porovnám hlavné poznatky, ako aj výhody jednotlivých frameworkov.

Hlavnou úlohou tejto práce však je vytvoriť funkčnú webovú aplikáciu pre diaľkové ovládanie ventilačného systému za pomoci jedného z reaktívnych frameworkov. Okrem základného ovládania je aplikácia rozšírená aj o možnosť sledovania dôležitých atribútov, rovnako ako aj ich štatistických vyhodnotení, hlavne v podobe grafu. Nasadenie výslednej aplikácie v praxi bude mať za následok zvýšenie komfortu pri prevádzke objektu.

# [Najpoužívanejšie](Pokyny_pre_vypracovanie_ZP.docx" \l "Súčasný_stav_riešenia) reaktívne frameworky

V súčasnosti existuje nespočetné množstvo frameworkov, ktorými je možné uľahčiť si prácu s front-end časťou webových aplikácií. Ich hlavnou výhodou oproti používaniu obyčajnej kombinácie HTML, JavaScriptu a CSS je rýchlosť vývoja aplikácií.

Zároveň však tieto frameworky majú rýchlejšie renderovanie stránok a lepšiu odozvu. Prečo je tomu tak? Pokiaľ si vezmeme pred seba ľubovoľný nereaktívny framework, napr. Laravel, zistíme, že všetky pohľady sa renderujú na strane servera a server pošle webovému klientovi už vyrenderovanú HTML stránku. Pokiaľ klient požiada o iné údaje, tento renderovací proces sa deje znova. Reaktívne frameworky však pracujú na princípe deklaratívneho a reakčného renderovania. To znamená, že namiesto toho, aby sa pri akejkoľvek akcii používateľa, či už ide o presmerovanie alebo odoslanie formulára, renderoval DOM znova, zmení sa iba stav objektov a iba tieto zmeny budú znova renderované. Je teda zrejmé, že deklaratívne renderovanie rozdeľuje klientskú aplikáciu na časť stavov, kde sú držané informácie o stavoch objektov a pohľady, ktoré tieto stavy zobrazujú. Inými slovami, stav objektov sa dá popísať aj ako jediný zdroj pravdy a pohľad ako mapovanie stavu modelu.



Obrázok 1 Schéma deklaratívneho a reakčného renderovania

## Angular

Angular je JavaScriptový framework, vyvíjaný a podporovaný spoločnosťou Google. V súčasnosti ide o plnohodnotný framework, navrhnutý podľa architektúry service/controller, čo znamená, že na rozdiel od knižníc, ako napr. jQuery, vie ponúknuť presmerovania pomocou vlastného router modulu. Pomocou Agularu je možné vytvoriť rôzne SPA a na dnešnom trhu je vývoj práve týchto aplikácií žiadaný. Pre svoj objektovo orientovaný prístup je vo veľkej miere využívaný vo väčších projektoch, kde je potrebné, aby na vývoji pracoval tím viacerých vývojárov. Dôležité je tiež spomenúť fakt, že spomedzi vybraných frameworkov je Angular najzložitejší z pohľadu učenia sa.

### História verzií

Angular vznikol v roku 2009 pod názvom AngularJS. Bol v podstate úvodným frameworkom pre SPA, teda rozšíril HTML o možnosť vývoja aplikácií, ktoré by sa za iných okolností programovali pre desktopové operačné systémy. Do vývojárskeho sveta priniesol nové pojmy, ako napríklad Two Way Data-Binding, vlasntú implementáciu Depencency Injection alebo znovupoužiteľnosť komponentov pri vývoji. Zároveň bol však navrhnutý podľa archtektúry MVC – model, view, controller. Uvedené pojmy si vysvetlíme neskôr v tejto kapitole.

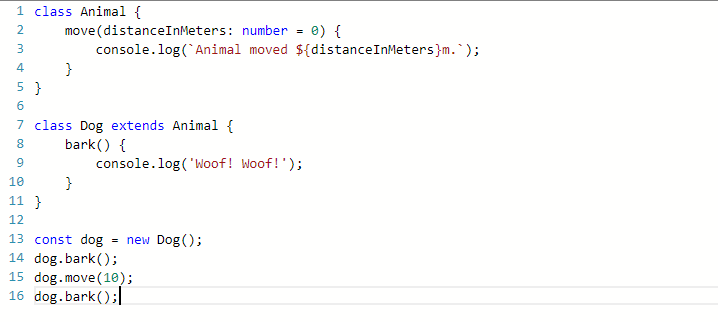
Jeho nástupcom bol Angular 2, ktorý sa od svojho predchodcu významne líšil. V tejto verzii už naďalej nebola používaná architektúra MVC, ale nahradila ju doteraz používaná service/controller architektúra. Zmena architektúry zapríčinila aj nemožnosť spätnej kompatibility medzi verziou 1 a 2, čo vo väčšine prípadov zapríčinilo, že vývojári boli nútení prepísať aplikačný kód odznova. Ďalšia veľká zmena nastala aj v používanom programovacom jazyku – JavaScript bol nahradený jazykom TypeScript, ktorý má pomáhať pri písaní robustného a štruktúrovaného kódu. Vzhľadom ku rozrastajúcemu sa počtu mobilných zariadení, Angular 2 sa tiež prispôsobil tejto platforme vylepšeným renderovaním komponentov.

Po verzii 2 nasledovala Angular verzia 4. Toto zdanlivo nelogické číslovanie je spôsobené tým, že už Angular 2 používal komponent Angular Router s verziou 3.x a teda vývojári chceli jednoducho zosynchronizovať číslovanie. Verzia 4 priniesla zmeny vo View Engine, čo viedlo k redukcii veľkosti generovaného kódu pre jednotlivé komponenty až okolo 60%. Druhou veľkou zmenou bolo osamostatnenie animácií, ktoré boli vytiahnuté z hlavného balíčka @anglular/core a bol im priradený balík @angular/platform-browser/animations. Ak teda vývojár nechcel poúžívať v aplikácii animácie, nemuseli sa zbytočne k nej pripájať. Uvedené zmeny praxi znamenali zrýchlenie celého frameworku. Výhodou bolo tiež zachovanie spätnej kompatibility s verziou 2.

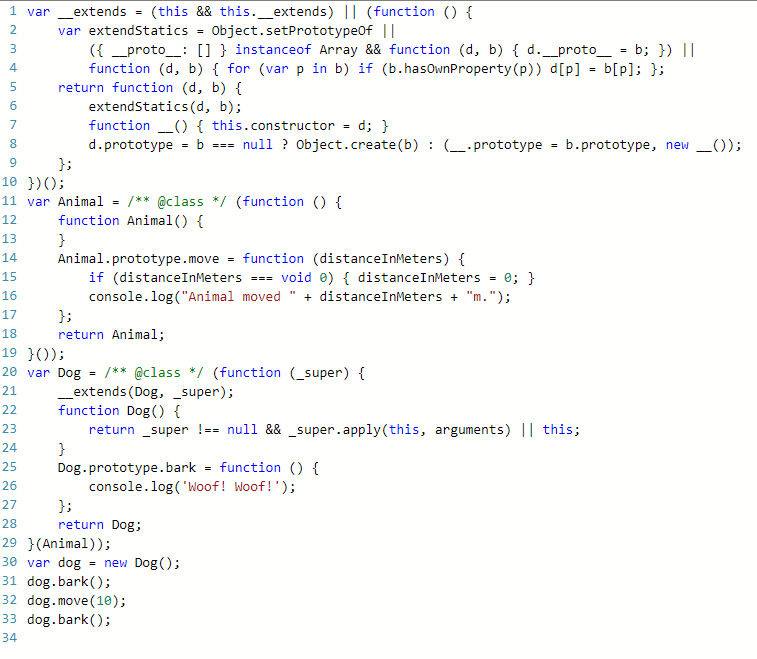
V čase písania tejto práce existuje najnovšia verzia s verziovým číslom 5.2. Medzi najdôležitejšie zmeny, ktoré priniesol Angular 5 patrí aj možnosť viacnásobného pomenovania komponentov aplikácie. Výhodu tohto spôsobu pomenúvania zdôvodňuje Otemuyiwa vo svojom článku: „Export komponentov s viacerými názvami môže používateľom pomôcť pri migrácii bez porušenia zmien.“[[1]](#footnote-1). Okrem toho Angular 5 obsahuje medzinárodné formáty čísel alebo dátumov a ako aj u predošlých verzií prebehla aj optimalizácia frameworku.

### TypeScript

Ako som už vyššie spomenul, Angular využíva TypeScript ako svoj hlavný programovací jazyk. TypeScript funguje ako nadstavba JavaScriptu a jeho kód sa do JavaScriptu aj kompiluje. Vznikol pod záštitou spoločnosti Microsoft a od svojej prvej verzie prešiel viacerými zmenami. Pre jednoduchší vývoj aplikácií, Microsoft pridal podporu jazyka TypeScript aj do svojho vývojového prostredia Visual Studio.

Angular ho však využíva pre jeho objektový prístup. TypeScript pozná pojmy ako trieda, rozhranie alebo dedičnosť. Rovnako sa stretneme aj s typovou kontrolou, ktorá v JavaScripte chýba. Nižšie sa nachádza ukážka zdrojového kódu, písaného jazykom TypeScript a pre porovnanie, uvádzame aj ukážku rovnakého kódu v jazyku JavaScript.

Obrázok 2 Ukážka kódu v jazyku TypeScript



Obrázok 3 Ukážka rovnakého kódu v jazyku JavaScript

### Inštalácia a používanie

Inštalácia frameworku prebieha pomocou príkazového riadku. Nevyhnutným základom pre používanie Angularu je inštalácia Node.js a npm, čo je vlastne veľký register balíčkov od rôznych vývojárov, ktorí chcú zdieľať svoje open-source projekty. V nasledujúcej tabuľke sú uvedené základné príkazy pre používanie Angularu.

Tabuľka 1 Základné príkazy v príkazovom riadku

|  |  |
| --- | --- |
| npm install –g @angular/cli | Inštalácia balíka Angular globálne na pevný disk. Príkaz sa zadáva iba raz pri samotnej inštalácii frameworku |
| ng new nazov-aplikacie | Vytvorenie nového projektu, kde nazov-aplikacie je ľubovoľný názov novej aplikácie. |
| ng serve | Spustenie servera. Častý parameter príkazu je –open (skrátene -o). Server je spúšťaný na porte 4200, pokiaľ nie je určené inak. |
| ng generate component nazov | Vygenerovanie nového komponentu. Názov môže byť ľubovoľný |
| ng generate service nazov | Vygenerovanie nového servisu. Názov môže byť ľubovoľný |

Základnými časťami Angular aplikácie sú component a service. Component je základným stavebným blokom aplikácií. Pomocou neho je možné zobraziť dáta na obrazovku, získavať vstupy od používateľa a následne može component na tieto vstupy reagovať. Component je definovaný nasledujúcimi súbormi:

1. app.component.ts – kód tried componentu, napísaný v TypeScripte. Obsahuje logiku komponentu alebo výpočty
2. app.component.html – šablóna komponentu, písaná v HTML
3. app.component.css - štýlovanie, týkajúce sa len daného komponentu

Vzhľad HTML a CSS dokumentov je takmer identický s ostatnými dokumentami, avšak TypeScript dokument komponentu vyzerá nasledovne:

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-heroes',

templateUrl: './heroes.component.html',

styleUrls: ['./heroes.component.css']

})

export class HeroesComponent implements OnInit {

constructor() { }

ngOnInit() {

}

}

Poďme si vyššie uvedený kód postupne rozobrať. Ako prvé sa nachádzajú v dokumente importujú komponenty z jadra Angularu. Ďalej je definovaný komponent samotný pomocou povinných parametrov selector, templateUrl a styleUrls. Parameter selector určuje názov selektora komponentu, ktorý bude použitý v HTML časti iných komponentov. TemplateUrl definuje cestu ku HTML dokumentu pre daný komponent a syleUrls obsahuje pole ciest k dokumentom určených pre štýlovanie. K jednému komponentu je teda možné priradiť viac štýlovacích dokumentov.

Komponent má aj svoj vlastný životný cyklus definovaný metódami:

Tabuľka 2 Životný cyklus komponentu

|  |  |
| --- | --- |
| **Metóda** | **Význam** |
| ngOnChanges() | Je volaná pred ngOnInit() a vždy ak sa zmení niektorý z data-bound prvkov |
| ngOnInit() | Inicializácia komponentu |
| ngDoCheck() | Volaná počas každej kontroly zmien hneď po ngOnChanges() a ngOnInit() |
| gnAfterContentInit() | Volaná raz po prvom ngDoCheck() keď vlastnosti zobrazovaného komponentu sú inicializované |
| ngAfterContentChecked() | Volaná po ngAfterContentInit() a každom nasleduúcom ngDoCheck() |
| ngAfterViewInit() | Volaná raz hneď po ngAfterContentInit() keď vlastnosti child templatu boli inicializované |
| ngAfterViewChecked() | Volaná ak sa zmenia vlastnosti child komponentu |
| ngOnDestroy() | Volaná tesne pred zničením komponentu. Najlepšie miesto pre uvoľnenie premenných alebo udalostí, aby nevznikli úniky pamäte. |

Vzhľadom k faktu, že pracujeme s komponentmi na objektovo orientovanej úrovni, máme k dispozícii aj konštruktor triedy. Avšak podľa odporúčaných postupov by sa konštruktor mal používať hlavne kvôli dependency injection, kedy Angular analyzuje parametre konštruktora a na základe nich vloží do komponentu potrebné inštancie objektov.

Na zobrazovanie obsahu premenných využíva Angular tzv. data-binding. V rámci práce s premennými je možné využiť direktívy ako ngIf – podmienka, ngFor – cyklus, ktoré sú určené pre ľahšiu prácu s premennými. Data-binding môže fungovať jednosmerne, kedy je iba vypísaný obsah premennej v šablóne komponentu alebo obojsmerne, pomocou direktívy ng-model, kedy sa aj v šablóne, napr. pomocou selektora input vo formulári dá meniť hodnota premennej za behu aplikácie. Rovnako je možné pomocou direktív preniesť hodnotu premennej z rodiča na potomka.

Ako ďalší stavebný prvok si opíšeme Service. Komponenty by nemali pristupovať priamo k dátam. Preto je táto úloha prenechaná Servisu. Servis teda zbiera informácie zo servera a predáva ich komponentom pomocou dependency injection. Servis zároveň funguje aj ako entita, ktorá uchováva dáta počas behu aplikácie a môže ich poskytnúť každému komponentu, ktorý si o ne požiada.

Angular ako plnohodnotný framework obsahuje aj Routing modul, slúžiaci na navigáciu medzi jedotlivými komponentmi. Pre lepšiu organizáciu zdrojových kódov sa odporúča, aby bol Routing modul implementovaný ako samostatný top-level modul, ktorý je následne importovaný do hlavného modulu aplikácie. Samotný modul obsahuje definíciu ciest, ktorá sa skladá z povinných parametrov path a component a ďalších nepovinných parametrov, ako napr. children alebo data. V module hlavnej aplikácie je možné routing modul vložiť do šablóny pomocou selektora <routing-outlet>.

## React

React nie je síce framework v pravom slova zmysle, je iba JavaScriptovou knižnicou, avšak má vlastnosti reaktívnych frameworkov, preto je zahrnutý aj v tejto bakalárskej práci.

React je vyvíjaný a udržiavaný hlavne spoločnosťou Facebook, avšak rovnako ako pri Angulari, aj tu majú vývojári možnosť prispieť k obohateniu frameworku.

Od svojho prvého vydania v roku 2013 React prešiel viacerými verziami a zmenami, ktoré neboli veľmi výrazné, ako na príklad v prípade Angularu a v súčasnosti je jeho verziové číslo v16.x.x. Zmeny vo frameworku sa týkali hlavne zrýchľovania renderovania aplikácie a do frameworku bolo pridaných viacero nových elementov (dialog, picture, atď.).

### JSX

Podobne ako Angular, aj React má svoje vlastné rozšírenie JavaScriptu v podobe JSX. Avšak na rozdiel od TypeScriptu JSX funguje skôr ako spojenie JavaScriptu s HTML, teda v jednom dokumente je možné písať aj logiku programu a zároveň aj spôsob, akým sa budú komponenty renderovať na obrazovke. Nie je nutnosťou toto rozšírenie používať v aplikáciách, avšak ako uvádzajú samotní autori frameworku React, „Veľa ľudí ho považuje za užitočný pri práci s používateľským rozhraním vo vnútri JavaScript kódu.“.[[2]](#footnote-2) Deklarácia premennej pomocou JSX môže vyzerať nasledovne:

const element = (

<h1>

Hello, {formatName(user)}!

</h1>

);

V prípade, že sa vo výraze bude nachádzať aj kód JavaScriptu (zvyčajne funkcia alebo premenná), je možné ho vložiť medzi množinové zátvorky a tým sa zabezpečí zároveň aj dynamickosť renderovaného komponentu. Ďalšou výhodou využívania JSX je spôsob ochrany aplikácie pred injection útokmi. Predtým ako JSX vyrenderuje element aplikácie, všetky premenné sú skonvertované na string – reťazec.

### Používanie frameworku

React aplikácia je tiež zložená z istých stavebných kociek. Tou najmenšou je element. Tieto elementy podliehajú istej hierarchii, kde na vrchole tejto hierarchie existuje základný element, pomocou ktorého sa renderujú všetci potomkovia tohto elementu. Renderovanie sa môže uskutočňovať pomocou funkcie ReactDOM.render();

ReactDOM.render(element, document.getElementById('root'));

Ďalšou stavebnou kockou sú komponenty. Koncept komponentov je založený hlavne na znovupoužiteľnosti rovnakého kódu. V Reacte môžu byť definované dvojako. Najjednoduchším spôsobom je definovania komponentu je pomocou JavaScript funkcie, ktorá obsahuje parameter props – properties ( v preklade vlastnosti ) predstavujúci dáta komponentu a návratovou hodnotou funkcie je React element.

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

Druhým spôsobom je definovanie pomocou triedy:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

Výhodou tried oproti funkciám je možnosť úplnej izolácie atribútov komponentu a využitie životného cyklu komponentu. O životnom cykle komponentu sa dozvieme viac neskôr v kapitole.

V prípade, že chceme vytvoriť element, obsahujúci náš komponent, je možné tento element definovať nasledovne:

const element = <Welcome name="Sara" />;

Selektor Welcome je v tomto prípade názov nášho komponentu a atribút name je JSX atribút komponentu, ktorý je vložený do objektu props. Takýto element sa potom renderuje rovnako pomocou funkcie ReactDOM.render();. Takýmto spôsobom je možné aj vnárať do seba rôzne komponenty a teda môžu spoločne vytvárať hierarchiu s viacerými úrovňami. Dôležité je spomenúť, že komponenty vytvorené vývojárom by mali začínať veľkým začiatočným písmenom, v opačnom prípade ich React bude považovať za DOM značky, napr. <div />, <a />, <input /> a nebude prebiehať správna typová kontrola komponentu.

Okrem atribútu props React komponent obsahuje aj atribút state. Ako už názov napovedá, state bude v sebe uchovávať stav komponentu. Tento atribút vznikol za účelom toho, aby sa vlastnosti, teda props komponentu nemenili počas existencie komponentu.

#### Životný cyklus komponentu

Životný cyklus je založený na volaní metód komponentu v určitom poradí. Komponent volá rôzne metódy v závislosti na tom, či sa komponent pripája, aktualizuje alebo odpája.

Pri pripájaní komponentu do DOM sa volajú metódy:

* Konštruktor – inicializácia atribútov triedy
* componentWillMount() – volaná pred metódou render()
* render() – povinná metóda. Táto metóda by nemala meniť stav komponentu a jej návratovou hodnotu je buď React element, čísla, reťazce, boolean hodnoty alebo hodnotu null
* componentDidMount() – vhodné miesto pre vykonávanie dopytov na server

Akutalizácia komponentu sa deje zapomoci metód:

* componentWillRecieveProps()
* shouldComponentUpdate() – metóda dáva Reactu vedieť, či nastali zmeny v props alebo state premenných. Ak nie, preskočí sa volanie všetkých zostávajúcich metód
* componentWillUpdate() – metódu je vhodné využiť na prípravu atribútov pred aktualizáciou
* render()
* componentDidUpdate() – na tomto mieste je znovu vhodné vykonať dopyty na server, avšak je nutné skontrolovať aj hodnoty props a state premenných pred zmenou a po zmene

V prípade, že je komponent mazaný z DOM, je volaná metóda componentWillUnmount(). Na tomto mieste je vhodné ukončiť všetky dopyty a vyčistiť pamäť.

#### Udalosti

Spracovávanie udalostí v Reacte je veľmi podobné tomu v DOM. Sú tu však malé syntaktické rozdiely. Udalosti sú pomenované tzv. ťaviou notáciou a s JSX sa obsluhy udalostí zapisujú ako funkcie, na rozdiel od JavaScriptu, kde sú zapisované ako reťazce. Nižšie je uvedený príklad obsluhy udalosti, zapísaný pomocou JSX, onClick – akcia po kliknutí na element.

<button onClick={this.activateLasers}>

Udalosť this.activateLasers je najčastejšie implementovaná ako metóda triedy a kľúčové slovo this odkazuje na triedu komponentu. Aby však táto udalosť mohla nastať, je nutné ju previazať s inštanciou objektu v jeho konštruktore pomocou metódy bind: this.activateLasers.bind(this); . Toto správanie metód alebo funkcií je určené samotným JavaScriptom.

Podobne ako Angular, aj React dokáže renderovať komponenty na základe určitej podmienky alebo tiež vyrenderuje viacero elementov pomocou cyklov.

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

if (isLoggedIn) {

return <UserGreeting />;

}

return <GuestGreeting />;

}

ReactDOM.render(

<Greeting isLoggedIn={false} />,

document.getElementById('root')

);

Na tomto príklade je možné vidieť komponent Greeting, ktorý na základe podmienky, isLoggedIn vykreslí buď komponent UserGreeting alebo GuestGreeting. Okrem klasickej if-else podmienky je možné použíť aj ternárny operátor.

Cyklus je v Reacte možné vytvoriť pomocou funkcie map(). Teda, ak máme určité pole prvkov, je možné pre každý prvok poľa vytvoriť jeho obraz, ktorý môže obsahovať hodnotu prvku alebo aj iné elementy alebo komponenty. Konktrétne, je v nasledujúcom príklade možné vytvoriť a zobraziť zoznam čísel nasledovne:

const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) =>

<li>{number}</li>

);

Po vykonaní funkcie map() bude element listItems obsahovať prvky zoznamu pre všetkých 5 čísel a tento element je možné renderovať. Taktiež je možné vo funkcii map() využívať kľúče (musia byť jedinečné iba v rámci funkcie map()), ktorými vieme zabezpečiť identitu prvku v rámci cyklu. Tento kľúč môže byť index poľa alebo číslo ID, ak ide o dáta z databázy.

Pri každom vyvíjanom systéme vzniká otázka, či je lepšie použiť kompozíciu alebo dedičnosť. Vývojári frameworku však odporúčajú uprednostňovať práve kompozíciu z dôvodu opätovného použitia komponentov a kódu. Rovnako aj oni uvádzajú, že pri tisíckach komponentov, ktoré samotný Facebook obsahuje, nenašli jediný prípad, kedy by uprednostnili dedičnosť pred kompozíciou.

## Vue

Vue je jeden z reakčných frameworkov, ktoré sú v súčasnosti veľmi využívané pre tvorbu SPA. Zakladateľom tohto frameworku je bývalý zamestnanec spoločnosti Google, Evan You. Dnes sa však na projekte podieľa väčšie množstvo vývojárov. Vue vznikol práve s úmyslom, aby zachytil deklaratívnu povahu data-binding, práve tak, ako Angular, ale s jednoduchším a prístupnejším API. Práve preto sa aj jednoduchšie učí a je odporúčaný pre začínajúcich vývojárov alebo pre vývojárov, pracujúcich na menších projektoch.

### Zmeny medzi verziami

Tak isto ako aj ostatné frameworky, aj Vue prešiel od svojho vzniku istými zmenami. Počiatočné verzie od 0.9 až po verziu 1.0 boli len na experimentálnej báze. Verzia 1.0 bola prvou hlavnou verziou. V súčasnosti sa však používa verzia 2.x, ktorá sa od prvých verzií líši hlavne v prepracovaní metód životného cyklu aplikácie, boli prepracované rôzne direktívy alebo boli zmenené parametre niektorých vstupov.

Zmien bolo mnoho a aby bol vývojárom uľahčený prechod zo starších verzií na novšie, vznikol modul vue-migration-helper. Tento modul skenuje súbory a hľadá v projekte zastaraný zdrojový kód a tým pomáha vývojárom prepracovať kód na novšiu verziu. Síce nie je schopný nájsť všetky zastarané vzory, avšak nájde ich podstatnú väčšinu.

### Inštalácia a používanie

Inštalácia frameworku môže prebehnúť buď vložením odkazu na skript už do vytvoreného projektu alebo je možné využiť príkazový riadok a za predpokladu, že vývojár má nainštalovaný Node.js a npm, je možné vytvoriť nový projekt s vygenerovanou adresárovou štruktúrou, stavanou pre Vue.

Každá Vue aplikácia vzniká vytvorením základnej, koreňovej Vue inštancie. K nej je možné pripojiť inštancie ďalších komponentov a spoločne vytvoria stromovú štruktúru. Pri inicializácii, každý komponent musí prejsť sériou funkcií – začne sa jeho žitovný cyklus.

Začiatok životného cyklu je tvorený metódami:

* beforeCreate() – úplne prvotná funkcia. Data objekt ešte nebol naviazaný na reakčný systém a taktiež neboli nastavené ani udalosti
* created() – na tomto mieste je už možné pristupovať k reakčným dátam aplikácie a udalosti sú už tiež aktívne. DOM však ešte nebol renderovaný

Cykly beforeCreate() a created() sú inicializačné a dávajú vývojárovi priestor na inicializáciu premenných alebo na vykonanie akcií s premennými predtým, ako budú nasadené do DOM. Nasledujúce dve funkcie sa nazývajú mounting hooks – upevňovacie funkcie.

* beforeMount() – funkcia je volaná pred prvým renderom DOM, avšak nie je volaná, pokiaľ renderovanie prebieha na strane servera. V praxi nie je veľmi využívaná.
* mounted() – v tejto funkcii má vývojár plný prístup ku kompomentu, jeho šablóne a renderovanému DOM objektu.

Cykly beforeUpdate() a updated() sú volané vždy, ak sa akákoľvek vlastnosť komponentu zmení, alebo nastane udalosť, ktorá vyvolá nové renderovanie komponentu. Tieto funkcice by však nemali slúžiť na zisťovanie, či sa niektorá z vlastností komponentu zmenila. Na tento účel vo Vue existujú objekty computed a watch, o ktroých si viac povieme neskôr.

* beforeUpdate() – funkcia má za úlohu zistiť všetky zmeny, ktoré v komponente nastali, tesne predtým, ako bude komponent znovu renderovaný.
* updated() – nové zmeny boli renderované; ak chce vývojár pristúpiť ku DOM, po tom, čo boli vlastnosti komponentu zmenené, tu je na to najvhodnejšie miesto.

Posledné dve funkcie slúžia na vykonanie akcií tesne pred zničením komponentu. Je to teda vhodné miesto na čistenie pamäte. Pri volaní funkcie destroyed() už je objekt zničený a funkcia môže byť využitá napr. na informovanie serveru o zničení komponentu.

* beforeDestroy()
* destroyed()

Vue využíva v šablónach systax založenú na HTML. Vzhľadom v faktu, že Vue vychádza z Angularu, tak väčšina direktív má podobnú syntax. Jednosmerný data-binding je rovnako ako v šablónach Angularu vyjadrený pomocou dvoch množinových zátvoriek, alebo ako ich sami vývojári nazývajú – fúzy:

<span>Message: {{ msg }}</span>

Vue však obsahuje aj direktívu v-once, ktorá zabezpečí, že dáta v elemente budú renderované iba raz. Fúzy však nie je možné použiť v HTML atribútoch elementu. V taktom prípade je data-binding zabezpečený pomocou direktívy v-bind. Medzi fúzy je možné vložiť nielen premenné, ale aj plnohodnotné JavaScript výrazy – je teda možné volanie funkcií, vykonávanie aritmetických operácií alebo je možné vložiť podmienku pomocou ternárneho operátora. No príliš veľa aplikačnej logiky v šablónach môže spôsobiť veľkú neprehľadnosť kódu. Riešením sú tzv. computed properties. Ide o objekt vo vnútri Vue komponentu, kde je možné definovať funkciu, ktorá bude vykonávať potrebnú logiku a vráti hodnotu, ktorá sa môže vyrenderovať na ozbrazovku. Vzniká tu však aj otázka, prečo by sme mali používať práve objekt computed a prečo nestačí iba zavolať obyčajnú funkciu? Rozdiel je v tom, že zatiaľ čo funkcia by musela vykonávať tú istú logiku neustále dokola, objekt computed bude vykonávať logiku svojej funkcie iba vtedy, ak sa zmení niektorá zo závislostí funkcie, inak iba vráti vypočítanú hodnotu bez nutnosti vykonávania funkcie. Cieľom je teda dosiahnuť lepšiu odozvu aplikácie znížením počtu potrebných výpočtov.

## Porovnanie

# [Zber](Pokyny_pre_vypracovanie_ZP.docx#Ciele_práce) požiadaviek

# Analýza požiadaviek

# Implementácia

# [Záver](Pokyny_pre_vypracovanie_ZP.docx#Záver)

# [Zoznam použitej literatúry](Pokyny_pre_vypracovanie_ZP.docx#Zoznam_použitej_literatúry)

<https://www.angularminds.com/blog/article/comparison-difference-between-angular1-vs-angular2-vs-angular4.html>

https://blog.angularjs.org/2017/03/angular-400-now-available.html

<https://blog.angular.io/version-5-0-0-of-angular-now-available-37e414935ced>

https://auth0.com/blog/whats-new-in-angular5/

<https://www.gurustop.net/newsletter/19>

https://www.typescriptlang.org/docs/handbook/classes.html

# [Zoznam príloh](Pokyny_pre_vypracovanie_ZP.docx" \l "Prílohy)

**Príloha A** Názov

**Príloha B** Názov

# 

# Prílohy

## Príloha A: Názov prílohy

Každá ďalšia príloha začína na novej strane.

## Príloha B: Obsah DVD

Priložené DVD obsahuje:

* Práca v elektronickej podobe (formát PDF)

1. https://auth0.com/blog/whats-new-in-angular5/ [↑](#footnote-ref-1)
2. https://reactjs.org/docs/introducing-jsx.html [↑](#footnote-ref-2)