# Moderní JavaScriptové frameworky z pohledu vytváření single-page aplikací

Bc. Filip Ivaška

Diplomová práce
2019



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Filip Ivaška** |
| Osobní číslo: | **A17732** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Forma studia: | **kombinovaná** |
| Téma práce: | **Moderní JavaScriptové frameworky z pohledu vytváření single-page aplikací** |
| Téma anglicky: | **Modern JavaScript Frameworks from the Perspective of Creating Single-Page Applications** |

Zásady pro vypracování:

1. **Vypracujte literární rešerši na dané téma.**
2. **Vysvětlete princip Single-page aplikací a nastudujte potřebné technologie pro jejich vývoj.**
3. **Charakterizujte Javascript frameworky "Angular", "React" a "Vue" v nejnovějších verzích.**
4. **Vytvořte vhodné ukázkové aplikace ve všech třech Javascript technologiích a popište postup jejich vývoje.**
5. **Proveďte porovnání vytvořených aplikací z pohledu jejich použitelnosti, náročnosti na vývoj a jejich rychlosti.**
6. **Navrhněte a vypracujte vhodné edukační materiály z této oblasti pro výuku studentů v informatických studijních programech.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:


1. FAIN, Yakov a Anton MOISEEV. Angular 2 development with TypeScript. Shelter Island, NY: Manning Publications Co., 2017. ISBN 9781617293122.
2. SCOTT, Emmit A. SPA design and architecture: understanding single-page web applications. Shelter Island, NY: Manning, 2016. ISBN 978-1617292439.
3. CHAU, Guillaume. Vue.js 2 Web Development Projects: Learn Vue.js by building 6 web apps. Packt Publishing, 2017. ISBN 978-1787127463.
4. WIERUCH, Robin. The Road to learn React: Your journey to master plain yet pragmatic React.js. CreateSpace Independent Publishing Platform, 2018. ISBN 978-1986338820.
5. FREEMAN, Adam. Pro angular 6. New York, NY: Springer Science Business Media, 2018. ISBN 978-1484236482.
6. JANSEN Remo. Learning TypeScript. Birmingham, Spojené království: Packt Publishing, 2015. ISBN 1783985550.


| | |
|---|---|
| Vedoucí diplomové práce: | **Ing. Milan Navrátil, Ph.D.**<br>Ústav elektroniky a měření |
| Konzultant: | **Ing. Hynek Petrla**<br>Holešov, Martinice |
| Datum zadání diplomové práce: | **3. prosince 2018** |
| Termín odevzdání diplomové práce: | **15. května 2019** |

Ve Zlíně dne 7. prosince 2018

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

doc. Mgr. Milan Adámek, Ph.D.
*děkan*

prof. Mgr. Roman Jašek, Ph.D.
*garant oboru*

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Filip Ivaška, v.r.

Ve Zlíně, dne …….

podpis diplomanta

## ABSTRAKT

Diplomová práca sa zaoberá single page aplikáciami a frameworkami, ktoré umožňujú tvorbu tohto typu aplikácií. Hlavným cieľom práce je popísať dôvody tvorby tohto typu aplikácií, postup vývoja v jednotlivých technológiách na vzorových aplikáciách, a porovnať rozdiely medzi nimi. V teoretickej časti sú vysvetlené základné pojmy a technológie potrebné pre vývoj single page aplikácií. V ďalších kapitolách sú popísané jednotlivé frameworky, ich charakteristické vlastnosti, funkcie a ich stručná história. Súčasťou praktickej časti práce sú štyri aplikácie. Jedna predstavuje server, na ktorý sa aplikácie pripájajú, a tri single page aplikácie, v každom SPA frameworku jedna. V nasledujúcej kapitole sú tieto frameworky porovnané. Posledná kapitola práce popisuje výukové prezentácie, ktoré sú súčasťou práce.



Klíčová slova: SPA, Single-page aplikace, JavaScript, Angular, React, Vue



## ABSTRACT

This master's thesis deals with single page applications and frameworks, which enable development of these apps. Main focus of this thesis is to describe reasons to develop this type of appications, development process in each of the frameworks by creating demo applications, and compare differences between them. The theoretical part contains description of basic terms and technologies needed for developing single page applications. Next chapters describe each of the frameworks, their charateristics, functions and their short history. Practial part contains four applications. One as a server, to which other apps connect, and then three applications, one in each of the frameworks. Next chapter contains comparison of these technologies. Last chapter describes presentations, which are part of this thesis.



Keywords: SPA, Single-page application, JavaScript, Angular, React, Vue

Ďakujem vedúcemu mojej diplomovej práce, pánovi Ing. Milanovi Navrátilovi, Ph.D., za odborné vedenie práce, cenné pripomienky, rady a návrhy na zlepšenie práce.

Ďalšie poďakovanie patrí konzultantovi tejto diplomovej práce, pánovi Ing. Hynkovi Petrlovi, za rady pri vypracovávaní praktickej časti práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

# OBSAH

# ÚVOD

Od roku 1991, kedy sa verejnosť dostala k používaniu WWW – World Wide Web prešiel internet obrovskými zmenami. Od prvej statickej stránky ktorá obsahovala pár riadkov textu, vytvorenej Timom Berners-Lee počas toho ako pracoval v CERN-e, až po komplexné informačné systémy, ktoré v našich prehliadačoch bežia dnes.

Tradičné fungovanie webových stránok a prehliadačov týchto stránok spočívalo v získaní dát zo serveru, ich zobrazenia, prípadnej úprave, a poslaní naspäť na server. Dnes sa prehliadače zmenili na komplexné prostredia, ktoré dokážu spúšťať aplikácie, bez akejkoľvek nutnosti inštalácie, bez ohľadu na operačný systém, na ktorom bežia. Táto schopnosť spôsobila migráciu rôznych aplikácií priamo web.

Z veľkej časti za to môžeme ďakovať popularite JavaScriptu, v ktorom sú tieto aplikácie, často nazývané single-page aplikácie, napísané. Single-page aplikácie ponúkajú radu rôznych výhod, napríklad komplexné aplikácie dostupné pre užívateľa kdekoľvek, bez inštalácie, bez ohľadu na zariadenie a operačný systém z ktorého sa užívateľ pripája. Pre vývojárov to prinieslo možnosť vytvoriť jednu aplikáciu v jednej technológií, ktorá beží kdekoľvek, bez nutnosti úprav, prípadne úplného prepísania aplikácie do technológie podporovanej iným operačným systémom. Preto tento prístup môže významne znížiť náklady na vývoj, prevádzku a údržbu systému, a to finančné, aj časové.

Single-page aplikácie sa dajú vyvíjať aj za použitia samotného JavaScriptu, tento proces je však pomerne náročný a zdĺhavý. Drvivá väčšina vývojárov siaha pri nutnosti vytvoriť single-page aplikáciu po niektorom z frameworkov, ktoré nám ponúkajú základné nástroje pre vytvorenie takejto aplikácie bez nutnosti ich od začiatku programovať. Tri najpopulárnejšie JavaScript frameworky pre vytvorenie single-page aplikáci sú v momentálne Angular, React a Vue, a práve nim sa budem venovať aj v tejto práci.

# I. TEORETICKÁ ČÁST

# 1 SINGLE-PAGE APLIKÁCIE

Pokusy o presun desktopových aplikácií začali už dávno. Skúšali sa rôzne riešenia a technológie, s rôznym stupňom úspechu. Jednalo sa napríklad o Java applety, IFrame, Adobe Flash a ďalšie. Síce sa jednalo o diametrálne odlišné technológie, mali za úlohu spoločný cieľ – pokus vytvoriť medzi platformovú aplikáciu podobnú desktopovej, ktorá bude bežať v prehliadači.

O tento cieľ sa snažia aj single-page aplikácie, avšak s tým rozdielom že sa používajú len technológie natívne podporované priamo prehliadačom, teda bez nutnosti doinštalovať do prehliadača pluginy.

Prvou technológiou, ktorá zmenila myslenie o tom ako mlžu webové aplikácie fungovať, bola technológia AJAX – Asynchronous JavaScript and XML. Táto technológia umožňovala posielať požiadavky na server na pozadí, bez nutnosti znovu načítania stránky. Tieto dáta získane zo serveru mohli byť pomocou JavaScriptu vložené do Document Object Modelu (DOM), teda do objektového modelu, ktorý tvorí stránku. Vďaka tomu bol možné dynamicky meniť obsah stránky.[1]

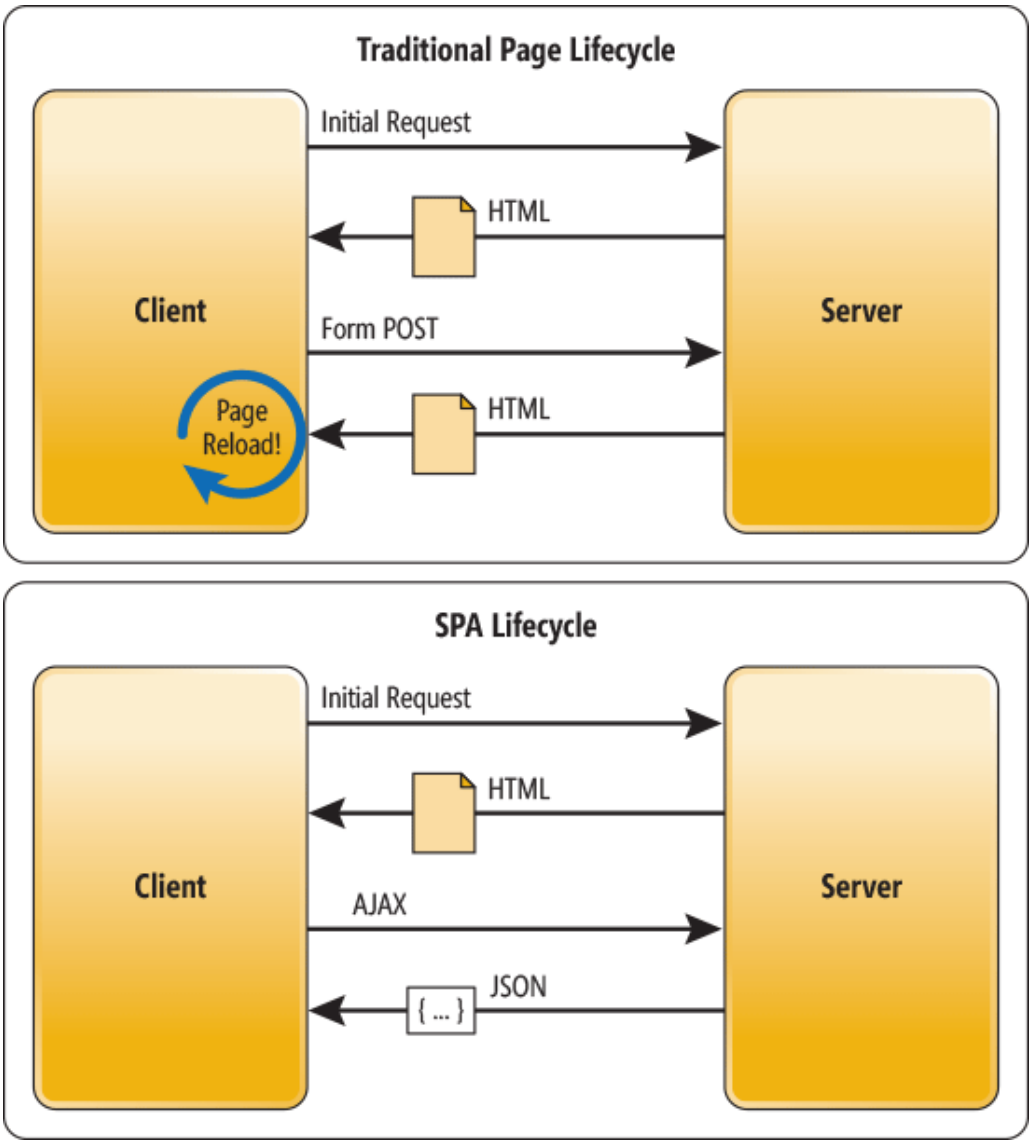
## 1.1 Princíp single-page aplikácií

V prípade použitia single-page aplikácie (SPA), beží celá web stránka ako jedna aplikácia. V takomto prípade sa úplne oddelí prezentačná a serverová vrstva aplikácie, tzv front-end a back-end, kde SPA sa starajú práve o front-endovú časť.

Pri štandardnom dizajne web stránok, sa pri potrebe aktualizovať dáta zobrazené na stránke pošle na server požiadavok, tzv. request. Server tento request spracuje, vygeneruje výsledný HTML kód, ktorý odošle naspäť prehliadaču. Prehliadač tento kód vyrenderuje a zobrazí užívateľovi. Server sa teda stará aj o prezentačnú vrstvu.

V prípade SPA sa prezentačná vrstva nachádza v klientovom prehliadači. Pri prvotnom načítaní stránky sa stiahnu nástroje potrebné na renderovanie stránok, a vyrenderuje sa prvé zobrazenie, tzv. view. Ak je potreba vygenerovať nové view, napríklad užívateľ klikol na odkaz, ktorý ho presmeroval na iné miesto v aplikácií, je toto view vyrenderované lokálne a dynamicky vložené do DOM stránky, zatiaľ čo pôvodné, už nepotrebné sa z DOM odstráni. Spolu s vyrenderovaním nového view sa pošle request na server (ak je potrebný). Dáta sa zo serveru zvyčajne vrátia vo formáte XML, alebo JSON. Následne sa spracujú

a doplnia do view. Výsledkom je presmerovanie a vyrenderovanie novej časti stránky s aktualizovanými dátami, bez nutnosti plného prenačítania v prehliadači. [1]




Obrázok 1. Popis komunikácie medzi klientom a serverom v rámci štandartnej a single-page aplikácie [2]

## 1.2 Technológie pre tvorbu SPA

Cieľom SPA bolo vyhnúť sa nutnosti doinštalovania pluginov do prehliadača – musia teda bežať natívne v každom prehliadači. Preto na vytvorenie SPA stačia základné technológie, ktoré sa využívajú aj na tvorbu štandardných stránok – HTML, CSS a Javascript.

### 1.2.1 HTML

HTML je skratka pre Hyper Text Markup Language. Tento jazyk je používaný na vytváranie štruktúry webových stránok. Na vytvorenie tejto štruktúry sa používajú HTML elementy, ktoré sú tvorené tagmi. Tagy môžu byť párové alebo nepárové. Jednotlivými elementami oddeľujeme časti obsahu, ako napríklad hlavičku, odsek tabuľky a podobne. Tieto elementy sú potom prehliadačom vyrenderované do grafického zobrazenia, ktoré vidí užívateľ. Momentálne najnovšou verziu je verzia 5.2. [3][4]

### 1.2.2 CSS

CSS – Cascade Style Sheets, sa používajú na štýlovanie HTML elementov. Fungujú na princípe selektorov, pomocou ktorých vyberieme žiadaný element, a ten potom dizajnujeme pomocou jednotlivých CSS vlastnosti, ako napríklad *font-size*, pre veľkosť textu, alebo *backgorund-color* pre farbu pozadia elementu. Dnes naberajú na popularite CSS preprocesory, ako napríklad Sass alebo LESS, ktoré pridávajú do štandardného CSS jazyka novú funkcionalitu, ako napríklad používanie premenných, dedičnosť a prehľadnejší zápis vnorených selektorov. Aktuálna verzia CSS je verzia 3. [5][6]

### 1.2.3 Javascript

Javascript je skriptovací programovací jazyk, používaný pri tvorbe webstránok. Hlavnou úlohou JavaScriptu je definovať a dynamicky upravovať správanie stránky. Pomocou tohto jazyku dokážeme manipulovať s HTML elementami, meniť ich rozmery, pozíciu, obsah alebo CSS vlastnosti. Bez JavaScriptu by nebolo možné vytvoriť iné ako obyčajné statické webstránky. Vďaka tomuto jazyku môžeme dnes vytvárať veľmi zložité aplikácie, na nerozoznanie od tých desktopových. Javascript začal ako čisto klientský jazyk, dnes už existujú technológie, vďaka ktorým môžeme použiť Javascript a na napísanie kódu ktorý beží na serveri, príkladom je technológia Node.js.

Prvá verzia jazyka vznikla v roku 1997, pod oficiálnym názvom ECMAScript 1. Najvýraznejšie zmeny prišli vo verziách ECMAScript 5 (skrátene ES5), v roku 2009 a potom vo verzií ECMAScript 6 (ES6), ktoré z JavaScriptu spravili najpopulárnejší jazyk, ktorý dnes poznáme. [7][8]

# 2 ANGULAR

Angular je framework pre vytváranie klientskej (front-end) časti webových aplikácií. Jeho charakteristickým znakom je, že na rozdiel od väčšiny ostatných frameworkov je sám napísaný, a aj pri vytváraní samotných aplikácií sa používa Typescript. Typescript, technológia od Microsoftu, je typovaný superset JavaScriptu, ktorý sa následne kompiluje do čistého JavaScriptu. [9]

## 2.1 Typescript

Javascript je slabo typový jazyk. To znamená, že pri jeho použití sa nevykonáva žiadna kontrola použitého typu objektu alebo premennej. Znamená to, že sú povolené operácie medzi premennými rôzneho typu. Tento spôsob programovania má síce za výsledok kompaktnejší kód, ale môže pri ňom nastať situácie, ktoré sa veľmi ťažko odlaďujú. Typickým príkladom môže byť napríklad sčítanie typu string a typu integer.

```
1    var x = "15";
2    var y = 15;
3    var z = x + y; // "1515"
```

V tomto prípade bude výsledok sčítania hodnota „1515“, ako string. Je to chyba, ktorá sa stáva pomerne často, a neraz trvá dlho kým sa ju podarí odstrániť. Práve tu prináša veľkú výhodu práve Typescript. Ten do JavaScriptu prináša typovú kontrolu. Počas kompilácie Typescriptu, kompilátor kontroluje aj jednotlivé typy premenných, a teda hore uvedený kód (zapísaný v Typescripte) by neprešiel kompiláciou. A vďaka rôznym pluginom, ktoré sa dajú nainštalovať do vývojových prostredí, by sme na nevalidnú operáciu dostali upozornenie už hneď pri napísaní kódu. Použitie silného typovania v Typescripte je však dobrovoľné. Nemusíme ho tak použiť vôbec, alebo ho použiť len na niektorých miestach. [10]

## 2.2 Architektúra Angularu

Základnými stavebnými blokmi Angularu sú tzv. NgModuly, ktoré slúžia ako kontext kompilácie komponentov. To znamená, že v moduloch sú deklarované jednotlivé komponenty, a rozdeľujú aplikáciu do funkčných blokov. Komponenty predstavujú jednotlivé

views, z ktorých skladáme aplikáciu. Komponenty majú k dispozícií služby, ktoré sa neviažu na konkrétne view, a môžu sa používať v rámci celej aplikácie. Tieto služby sa volajú service. Tieto služby sa používajú v komponentoch pomocou Dependecy Injection. Na navigáciu po aplikácií sa využíva AngularRouter. [9][11]

## 2.3 NgModuly a komponenty

NgModuly zabezpečujú modulárnosť každej Angular aplikácie. Moduly sú kontainery, do ktorých rozdeľujeme súvisiace časti kódy alebo jednotlivé časti stránky, kde napríklad každá položka v menu má vlastný modul. Každá aplikácia obsahuje minimálne jeden koreňový modul, zvyčajne nazývaný *AppModule*. Tento modul sa načíta ako prvý pri spustení aplikácie.

V moduloch sa deklarujú rôzne vlastnosti:

- *declarations:* deklarácacie jednotlivých komponentov, direktív alebo pipe
- *exports*: deklarácie, ktoré sú viditeľné pri importovaní modulu druhým modulom
- *imports*: miesto, kde pridáme moduly, z ktorých chceme použiť jeho exportované komponenty.
- *providers*: deklarácia služieb.
- *bootstrap*: dekláracia hlavného komponentu. Tento parameter by sa mal použiť len raz v hlavnom module. [12]

Komponenty tvoria jednotlivé views. Celé stránky sa skladajú z komponentov, kde sa každý komponent stará o časť zobrazenia. Môže sa jednať o komponenty, ktoré majú len jednu úlohu, napríklad sa starajú o zobrazenie jednej položky listu, alebo o komponenty, ktoré slúžia ako hlavný komponent na stránke a pozostávajú z väčšieho množstva iných komponentov.

Komponent je Typescript trieda, ktorá je označená dekorátorom *@Component*. Komponent sa stará o logiku zobrazenia a spracovanie dát. Každý komponent k sebe musí mať priradený HTML súbor, ktorý slúži ako šablóna. Tento HTML súbor môže mať každý komponent vlastný, prípadne mu môže byť priradený aj už existujúci súbor iného komponentu. Ku komponentu môže byť priradený aj CSS súbor, ktorý obsahuje štýly ku šablóne. Komponent spolu so šablónou tvoria view. [13][14]

Angular komponenty majú po svojom vytvorení životný cyklus, tzv. Lifecycle hooks. Jednotlivé metódy sa volajú v určitých momentoch život daného komponentu. Jednotlivé metódy sa volajú nasledovne:

- *ngOnChanges()*: volá sa raz pred *ngOnInit()*, a vždy potom ako sa zmenia niektoré z dát, ktoré sú poslané do komponentu z nadradeného komponentu pomocou *@Input()*.
- *ngOnInit()*: zavolané jedenkrát po inicializácií komponentu potom, ako sa prvý krát načítajú dáta poslané do komponentu pomocou *@Input().*
- *ngDoCheck()*: voláne vždy potom, ako Angular spustí svoju detekciu zmien, vždy po *ngOnChanges()* a *ngOnInit().*
- *ngAfterContentInit():* spustené raz, potom ako sú načítané všetky externé dáta do šablóny komponentu.
- *ngAfterContentChecked*(): spustené vždy po spustenej kontrole, ktorá nastala po zmene dát viazaných na komponent.
- *ngAfterViewInit()*: spustené jeden krát potom, ako sa nainicializuje šablóna dokumentu.
- *ngAfterViewChecked*(): spustené vždy po spustenej kontrole a zmene v šablóne komponentu.
- *ngOnDestroy()*: spustené jedenkrát predtým ako Angular odstráni referenciu na daný komponent. [13][15]

## 2.4 Service, Dependency Injection

Service, alebo služba, je objekt ponúkajúci spoločnú funkcionalitu, ktorá podporuje výstavbu ďalších blokov aplikácie – napríklad komponentov alebo direktív. Zjednodušujú štruktúru aplikácie, zabraňujú zbytočnému duplikovaniu kódu, zlepšujú znovu využiteľnosť kódu a zľahčuje následné úpravy, keďže stačí, keď sa vykonajú na jednom mieste.

Príkladom môže byť service, ktorá sa stará o zistenie aktuálneho kurzu medzi CZK a EUR. Keďže je možné, že túto funkcionalitu budeme potrebovať vo viacerých komponentoch, je vhodné vytvoriť service, a neimplementovať túto funkcionalitu do každého komponentu zvlášť. V service vytvoríme metódu, ktorá bude ako parametre brať do akej meny chceme z CZK zmeniť. Ako druhý parameter bude suma, ktorú potrebuje zameniť. Táto metóda pošle dotaz na server, ktorý dokáže vrátiť aktuálne kurzy, túto hodnotu spracuje, a vráti

správny výsledok. Vďaka tomu, že je táto funkcionalita ako service, je použiteľná v rámci celej aplikácie, a nie je nutnosť ju definovať stále znovu. Služby sa označujú dekorátorom *@Injectable*.

V komponentoch dokážeme service použiť, vďaka technike zvanej dependency injection. Angular má túto funkčnosť zabudovanú ako svoju základnú súčasť. Pri tejto technike vyjadrujeme závislosť komponentu na service, ktorú sme vložili do konštruktoru daného komponentu. Angular pri vytváraní každého komponentu skontroluje jeho konštruktor, a vytvorí objekty, na ktorých je konštruktor závislý. V našom prípade je tento objekt práve service. Tento objekt sa však vytvorí len v prípade že ešte neexistuje. To znamená že service je tzv. singleton. Pre celú aplikáciu vždy existuje len jedna instancia danej service. Táto vlastnosť je veľmi užitočná, napríklad keď chceme zdieľať dáta naprieč aplikáciou medzi komponentami. [13][15]

## 2.5 Routing

Väčšina aplikácií potrebuje ukazovať užívateľovi rôzny typ obsahu. Tento prípad sa rieši pomocou routingu, kedy sa na základe URL adresy v prehliadači mení zobrazený obsah. O obsluhu ciest sa stará modul *RouterModule*, preto každý modul, ktorý pracuje s cestami, musí importovať práve *RouterModule*. V Angulari sa registrujú jednotlivé cesty v príslušných moduloch. Pri vytvorení nového projektu, je vytvorený okrem *AppModule* v súbore *app.module.ts* aj súbor *app.routing.ts*. V tomto súbore sú definované jednotlivé cesty k modulom. Povedzme že naša aplikácia obsahuje cestu */auth*. V *app.routing.ts* zadefinujeme že táto cesta odkazuje na modul *AuthModule*. Samotná obsluha cesty sa už rieši v module *AuthModule*. V ňom môže zadefinovať ktorý komponent sa má spustiť na danej adrese. Ak chceme na adrese */auth/login* spustiť *LoginComponent*, vytvoríme konštantu v tvare

```
const routing: ModuleWithProviders = RouterModule.forChild([
  {
    path: 'login',
    component: LoginComponent,
  }
]);
```

Túto konštantu potom importujeme do *AuthModule*, a nová cesta bude prístupná v prehliadači. [13][17]

## 2.6 Data-binding

Data-binding je mechanizmus ktorý sa stará o koordináciu zobrazených dát, a dát ktoré sa nachádzajú v aplikácií. Existuje možnosť priamo meniť HTML kód, ale lepším a prehľadnejším riešením je použiť niektorú z väzieb ktoré ponúka Angular. Angular podporuje 3 typy väzby dát.

- *Jednosmerná zo zdroju do cieľa:* Dáta sa pošlú do view alebo podradeného komponentu. Dáta v aplikácií nereagujú na zmenu ktorá prišla z cieľu tejto väzby. Napríklad vypísanie premennej do šablóny
- *Jednosmerná z cieľu do zdroja:* používa sa napríklad pri spustení metódy po udalosti, ktorá nastala v šablóne – kliknutie na tlačidlo, interakcia so vstupom...
- *Obojsmerná:* Dáta reagujú na zmeny bez ohľadu kde zmena nastala [18]

## 2.7 Šablóny, pipe

Šablóny tvoria viditeľnú časť aplikácie. V architektúre MVC, ktorú Angular používa, sú šablóny časť View, zatiaľ čo komponent tvorí časť Model a Controller. V Angular šablónach sa používa klasický HTML jazyk obohatený o rôzne direktívy, ktoré umožňujú pokročilejšiu prácu v rámci HTML.

Dôležitou funkciou Angular šablón je interpolácia. Pomocou nejak dokážeme vypisovať hodnoty premenných do HTML kódu. Používa sa obalením názvu premennej do dvojitých zložených zátvoriek.

```
<div>{{ variable }}</div>
```

Uvedený kód vypíše hodnotu premennej *variable* ako text do elementu <*div*>. V rámci interpoláciu sú validné aj matematické výrazy.

Ďalšou veľmi používanou funkciou šablón sú štrukturálne direktívy. Tie prenášajú funkčnosť zobrazovať obsah na základe podmienok, prípadne vypisovať elementy z polí. Tieto direktívy mimikujú podmienky a cykly, ktoré poznáme z programovacích jazykov, a prinášajú ich do HTML kódu. Pre použitie podmienečného zobrazenia sa používa direktíva **ngIf* alebo **ngSwitch*. Do parametru direktívy **ngIf* vložíme podmienku, a daný element sa zobrazí iba v prípade, že podmienka je splnená.

```
<div *ngIf="condition">{{ variable }}</div>
```

Pri tomto zápise sa zobrazí daný element iba ak podmienka *condition*, bude vyhodnotená ako *true*.

Pri nutnosti vypísať list sa používa direktíva **ngFor*. Táto direktíva je obrazom funkcie cyklu *foreach*. Pre použite tejto direktívy zadefinujeme ako má vyzerať jeden prvok z listu. Šablóna potom použije tento element, a vypíše ho toľko krát, koľko je v liste položiek

```
<div *ngFor="let item of array">{{ item }}</div>
```

Tento zápis hovorí- vyrenederuj div element toľko krát, koľko je prvkov v liste *array*. Pri každom jednotlivom výpise ulož práve vypisovaný prvok do lokálnej premennej *item*. Pomocou tejto premennej dokážeme pristúpiť ku jednotlivým položkám listu, a vypísať ich.[13]

Súčasťou syntaxe používanej v šablónach sú aj pipe. Pipe sú používané na dynamickú transformáciu dát, do žiadaného formátu. Klasickým príkladom je zmena zápisu dátumu z formátu JavaScrip Date objektu, do formátu dátumu, tak ako ho poznáme. Ak vytvoríme nový Date objekt pomocou *new Date(),* a pokúsime sa ho vypísať do šablóny pomocou interpolácie, výsledná hodnota bude vo formáte

*Thu May 02 2019 13:26:58 GMT+0200 (Central European Summer Time)*

Stačí však pridať pipe s názvom date, ktorú automaticky ponúka Angular. Pre pridanie pipe stačí pridať jej názov za znak |.

```
<div>{{ variable | date }}</div>
```

V tomto prípade bude výsledná hodnota

*2.5.2019*

Pipe dokážu prijímať aj parametre. Ak chceme spomínaný dátum vypísať aj s časom, stačí pridať parameter ‚*medium*'. [14] Zápis v tom prípade bude vyzerať nasledovne:

```
<div>{{ variable | date:'long' }}</div>
```

Väčšie množstvo pipe ponúka priamo Angular, napríklad pipe pre formátovanie čísiel, meny, písma a podobne. Ďalšie sa dajú doinštalovať ako balíčky, prípadne je možnosť naprogramovať si vlastnú. [21]

## 2.8 História frameworku

Angular je open source framework udržiavaný spoločnosťou Google. Jeho prvá verzia bola vyvinutá slovenským programátorom Michalom Hevérym v roku 2009, keď pracoval vo firme Brat Tech LLC.

Vo svojej prvej verzií sa framework nazýval AngularJS, a na vývoj sa používal Javascript, nie Typescript, a začal ako platená služba. Po počiatočnom neúspechu sa však tím okolo Hevéryho rozhodol vypustiť AngularJS ako open source knižnicu. [22]

Angular od verzie 2, ktorá bola oznámená v roku 2014, a vyššie, prešiel úplným prepísaním. Začal sa používať Typescript, bol kladený veľký dôraz na modularitu a komponenty predstavujú základnú charakteristiku architektúry frameworku. Vzhľadom na veľké zmeny nebola táto nová verzia spätne kompatibilná s verziou AngularJS. Tento krok vzbudil veľkú kontroverziu medzi developermi. Finálna verzia 2.0 oficiálne vyšla v roku 2016 [23][24]

Framework túto počiatočnú nepriazeň prežil a nachádza vo verzií 7.2 a teší sa veľkej popularite – na portáli github.com kde je hostovaný má vyše 47 600 hviezidčiek. [25]

# 3 REACT

React je JavaScriptová knižnica, ktorá sa sústreďuje na View, v klasickom MVC modeli. Jeden view je poskladaný ako hierarchia komponentov. React sám o sebe nie je plnohodnotný framework. Plnohodnotným frameworkom sa stáva po pridaní ďalších súčastí, ktoré z neho robia kompletnú technológiu pre tvorbu SPA. Meno React je odvodené od slova reactive, čo naznačuje filozofiu frameworku -jednotlivé komponenty reagujú na aktuálny stav aplikácie. Aplikácia ako celok neudržiava svoj stav, stavy si udržiavajú len jednotlivé komponenty.

Ak chceme pridať funkcionalitu na udržiavanie globálneho stavu, je k tomu potreba externý plugin. React takisto zo základu neposkytuje veci ako router, prípadne obsluhu http requestov. O tom čo bude React podporovať rozhodne užívateľ tým, ktoré balíčky doinštaluje. Tento princíp umožňuje zanechať celkovú veľkosť aplikácie nízko, keďže obsahuje len funkcionalitu, ktorú programátor naozaj potrebuje. [26][27]

## 3.1 Komponent

Komponenty ponúkajú spôsob rozdelenia UI do nezávislých, znovu použiteľných kódu. Koncepčne je komponent v podstate JavaScriptová funkcia, ktorá prijíma vstupné parametre ako *props* (skrátene od slova properties), a vracia React element popisujúci čo sa ma v prehliadači vyrenderovať. [27][28]

Komponenty môžu byť v Reacte zadefinované ako triedy, alebo funkcie. Väčšiu funkcionalitu ponúkajú komponenty ako triedy. Pre zadefinovanie triedy ako React komponentu, musí táto trieda rozširovať triedu *React.Component.* [29]

Každý komponent si udržiava svoj vlastný lokálny stav, ktorý nám umožňuje ukladať, upravovať alebo vymazávať vlastnosti uložené v komponente. Ukážkový komponent v Reacte by mohol vyzerať nasledovne.

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
```

```
          {this.state.date.toLocaleTimeString()}
        </div>
      );
    }
}
```

Vytvorená trieda Example rozširuje triedu React.Component. Do konštruktoru komponentu je poslaný parameter *props*, čím sa inicializujú vlastnosti komponentu. Do objektu *this.state* priradíme začiatočný stav komponentu, konkrétne do vlastnosti *date* priradíme aktuálny dátum. Následne v metóde *render()* zadefinuje ako chceme aby tento stav videl užívateľ vo svojom prehliadači. [26][28]

Podobne ako Angular komponenty, aj React komponenty majú svoj životný cyklus.

- *render()*: jediná povinná metóda v rámci React komponentu. Volá sa vždy pri aktualizácií komponentu
- *componentDidMount()*: zavolá sa jeden krát po tom ako bol výstup z komponentu vyrenderovaný.
- *componentWillUnmount()*: zavolá sa jeden krát predtým, ako je komponent odstránený z DOM. [26][30]

## 3.2 JSX

JSX (skrátene JavaScript Extension) je odporúčaná technológia na popis dizajnu v Reacte. Jedna sa o rozšírenie syntaxu JavaScriptu. React nemá filozofiu separovania technológií do vlastných súborov, ako je to napríklad pri Angulari, kde je Typescript a šablóna písala v HTML v dvoch separátnych súboroch. Všetok kód vrátane JSX sa pri Reacte píše priamo do komponentu. Pri prvom pohľade nápadne pripomína HTML jazyk. Vo výsledku sa však JSX kompiluje do JavaScriptového kódu. Ako príklad môžeme uviesť výpis nadpisu do *render()* metódy v komponente.

```
<h1 className='large'>Hello World</h1>
```

Tento kód bude vo výsledku preložený do JavaScriptu ako

```
React.createElement(
   'h1',
   {className: 'large'},
  'Hello World'
  )
```

Používanie JSX nie je nutné, je možne písať priamo JavaScript kód. Používanie JSX však výrazne zvyšuje prehľadnosť kódu, a znižuje učiacu krivku, keďže je pravdepodobné že väčšina web programátorov sa už stretla s HTML kódom, ktorému sa JSX veľmi podobá. [31][32]

## 3.3 Redux

Redux je open-source JavaScriptová knižnica, ktorá umožňuje globálne manažovanie stavu aplikácie. Redux prináša jednosmerný tok dát, a drží sa niekoľkých zásad.

- Aplikácia má globálny stav
- Zmena tohto stavu spustí aktualizáciu dotknutých komponentov
- Iba niektoré funkcie dokážu meniť stav
- Tieto funkcie sú spustiteľné len užívateľom
- Vždy prebieha iba jedna zmena stavu

Tento globálny stav nedokáže spúšťať sám o sebe žiadne ďalšie akcie, to dokáže len vstup od používateľa. To spôsobuje že tento stav je omnoho ľahšie udržiavateľný a zabraňuje to jeho nechceným zmenám. [33]

V tomto globálnom stave môžu byť uložené napríklad odpovede zo serveru, nacachované dáta, alebo dát vytvorené lokálne. Redux umožňuje manipuláciu a udržiavanie práve týchto dát.

## 3.4 Virtual DOM

Použitie Virtual DOM nacachovanú verziu DOM aplikácie, ktorú uloží do pamäti zariadenia na ktorom beží. O aktuálnosť tohto zobrazenia v pamäti s reálne vyrenderovaným zobrazením sa stará knižnica ReactDOM. Výhodou tohto princípu je to, že vždy sa aktualizujú len tie časti stránky/komponenty, u ktorých je to nutné, teda došlo pri nich ku zmene stavu. Tým sa odstraňuje nutnosť znovu renderovať celú stránku.

## 3.5 Aplikovanie štýlov do komponentov

React umožňuje veľkú voľnosť pri výbere ako budeme štýlovať jednotlivé komponenty. Prvou možnosťou je vytvorenie súboru, ktorý obsahuje kód na štýlovanie, a jeho následne importovanie do komponentu. Tento súbor môže byť typu CSS, kde používame klasický

CSS kód, alebo je možnosť použiť niektorý zo preprocesorov, ako napríklad LESS alebo SASS.

Ďalšou možnosťou je vytvorenie štýlov ako JavaScript objektu, a následné priradzovanie štýlov z tohto objektu priamo do šablóny JSX. Toto riešenie prináša výhody, napríklad máme kontrolu nad tým kde sa ktoré štýly používajú pomocou počtu referencií na objekt v ktorom sa štýly nachádzajú, alebo jednoduché zmeny štýlov na základe hodnoty premenných v komponente. Pri tomto použití však aj prichádzame o výhody, ktoré prinášajú preprocesory, ako napríklad CSS premenné, alebo skladanie jednotlivých selektorov.

## 3.6 História frameworku

React je open source knižnica vytvorená a udržiavaná spoločnosťou Facebook. Hlavnou motiváciou bola enormná expanzia Facebooku, čo sa týka počtu užívateľov, tak aj narastajúcou funkcionalitou sociálnej siete. Časom začala byť ťažšie udržiavateľná.

React bol predstavený ako open source projekt na konferencií JS ConfUS v roku 2013. Spočiatku boli o tejto technológií pochybnosti, keďže išla proti dovtedy všetkým zaužívaným postupom.[34]

React neprešiel tak veľkými zmenami ako napríklad Angular, a drží sa svojej pôvodnej filozofie, a po pôvodných pochybnostiach je to dnes najpoužívanejší a najpopulárnejší JavaScriptový SPA framework, ktorý má na GitHube viac ako 128 000 hviezdičiek

# 4 VUE

Vue je Javascriptová knižnica pre tvorbu SPA. Z troch spomínaných frameworkov je najmenšia. Zhruba sa drží návrhového vzoru MVVM – Model-View-ViewModel. Tento vzor určuje rozdelenie View (užívateľského rozhrania) a Modelu (dát, s ktorými užívateľské prostredie pracuje). Komunikáciu medzi týmito vrstvami ViewModel, v tomto prípade knižnica Vue. Na rozdiel od Angularu, React aj Vue je možné nasadiť do existujúceho projektu. Vďaka relatívnej jednoduchosti základnej verzie knižnice, je možné vytvoriť jednoduché komponenty, ktoré obsluhujú len časť už existujúcej web stránky. Je však možné vytvoriť aj komplexné SPA vďaka doplnenia základnej knižnice o oficiálne pluginy, ktoré z knižnice tvoria plnohodnotný framework. Tieto pluginy sa starajú o routing, udržiavanie globálneho stavu a pod. V tomto si je Vue veľmi podobné s Reactom, požičiava si však aj niektoré schopnosti Angularu (ako napríklad obojsmernú väzbu dát). [36]

## 4.1 Komponent

Komponenty sú znovu použiteľné instancie knižnice Vue. Každý komponent môže obsahovať niekoľko metód alebo nastavení. Jedná sa napríklad o jednotlivé metódy životného cyklu, funkcia pre vytvorenie vlastných metód v rámci komponentu, alebo metóda pre uloženie dát.

```
export default {
  name: 'app',
  components: {
    Sidebar
  },
  data: () => ({
      showNavigation: false,
      showSidepanel: false
    })
}
```

V hore uvedenom príklade môžeme vidieť príklad Vue komponentu. Tento komponent je v aplikácií dostupný pod menom *app*. V objekte *components* sa registrujú podriadené komponenty, ktoré sú zobrazené v rámci daného komponentu. Funkcia *data()* obsahuje deklarácie premenných s ktorými sa v komponente pracuje. Dáta sa musia uchovávať v rámci funkcie z dôvodu uchovávania jedinečnej instacie týchto dát pre každú instanciu komponentu. V prípade že by sme mali viac krát použitý ten istý komponent, a dáta by boli

uložené ako jednoduchý objekt, zmena dát v jednom komponente, by ovplyvnila dáta aj v ostatných instanciách. [37]

Podobne ako pri Reacte, do komponentu môžeme poslať z nadradeného komponentu dáta pomocou props. V Angulare docielime podobné zdieľanie dát pomocou funkcie *@Input()*.

Pri Vue komponentoch sa píše JavaScriptový kód, kód HTML a takisto aj štýlovanie pomocou CSS, alebo ľubovoľného preprocesora, ako napríklad SASS alebo LESS.

Komponenty majú aj svoj už spomenutý životný cyklus. Tento životný cyklus sa skladá z niekoľkých metód.

- *beforeCreate():* Prebehne jedenkrát, pred inicializáciou komponentu
- *created():* Prebehne jedenkrát po inicializácií dát z objektu *data*
- *beforeMount():* Prebhen jedenkrát predtým ako dôjde k prvému vyrenderovaniu šablóny
- *mounted():* Prebehne jedenkrát po kompletnom vyrenderovaní komponentu
- *beforeUpdate():* Prebehne vždy po zmene dát v komponente, predtým ako dôjde k opätovnému vyrenderovaniu šablóny komponentu
- *updated():* Prebehne po vyrenderovaní po zmene dát v komponente.
- *beforeDestroy():* Prebehne pred odstránením komponenty z DOM pokým sú dáta a šablóna komponentu ešte plne dostupné
- *destroyed():* Prebehne jedenkrát potom, ako bol komponent odstránený [38]

## 4.2 Šablóny

Vue poskytuje slobodu v tom, akým spôsobom sa píše užívateľské rozhranie. Je možnosť použiť technológiu JSX, podobne ako je to pri technológií React. Je však možné použiť aj tradičný jazyk HTML, spolu s direktívami a ďalšími funkciami, ktoré ponúka framework.

Vue direktívy poskytujú bohatú funkcionalitu priamo v rámci HTML jazyka, podobne, ako je to pri frameworku Angular. Tieto direktívy umožňujú ovládať interakciu s užívateľom, vypisovať hodnoty premenných pomocou interpolácie, vypisovať listy pomocou ekvivalentu funkcie foreach a podobne. Direktívy začínajú označením *v-,* za čím nasleduje názov direktívy a prípadný parameter. Ako príklad môže byť direktíva obsluhujúca klik užívateľom na nejaký element na stránke.

```
<button v-on:click="counter += 1">Add 1</button>
```

Na elemente tlačidla sa použila direktíva *v-on* s parametrom *click*. Tá zaručuje, že po kliknutí na tlačidlo sa vykoná kód v úvodzovkách. Môže obsahovať zavolanie funkcie, alebo, ako v tomto prípade, pripočíta jednotku k premennej *counter*.

Pre vypísanie listu a simulácie funkcie foreach sa dá použiť direktíva v-for.

```
<ul>
    <li v-for="item in items">
      {{ item.message }}
    </li>
</ul>
```

Pomocou tejto direktívy prejdeme všetky prvky listu *items* a pre každú iteráciu vypíšme položku *message*.

## 4.3 Sledovanie zmien

Vue ponúka funkciu sledovania zmien dát v komponente. Túto funkciu umožňujú tzv *watchers*, ktoré definujeme v objekte *watch* v jednotlivých komponentoch.

```
watch: {
    variable: function (newVar, oldVar) {
      console.log(newVar, oldVar)
    }
  }
```

V príklade kódu vyššie zadefinujeme funkciu, ktorá sa spustí vždy po zmene premennej *variable*. Ako parametre tejto funkcie sú dve premenné, v prvom parametri je uložená nová hodnota po zmene, v druhom parametri je pôvodná hodnota. Táto funkcionalita je vhodná napríklad vtedy, ak potrebujeme vykonať akciu po zmene dát užívateľom, napríklad ju chceme poslať na server, a uložiť ju.[36]

## 4.4 Vuex

Vuex je oficiálna knižnica na udržiavanie globálneho stavu ako aplikácie, podobná Reduxu pri použití s Reactom. Jedná sa o centrálny úložný priestor pre všetky komponenty v aplikácií, zabezpečujúci zmenu stavu predvídateľným spôsobom. Používa sa v prípadoch, kedy viacero komponentov závisí na jednom stave dát aplikácie, alebo napríklad ak akcie z rôznych častí aplikácie potrebujú zmeniť tie isté dáta. Pri použití tejto knižnice odpadáva nutnosť zdĺhavo posielať dáta do podriadených komponentov pomocou

props. Tento systém prenášania dát napríklad vôbec nefunguje pre komponenty na rovnakej úrovni. Vďaka Vuex vyjmeme tieto spoločné dáta nad komponenty, a vytvoríme z neho jeden singleton ku ktorému majú komponenty prístup a môžu v ňom meniť dáta pomocou mutácií (dáta sa nedajú meniť priamo). Na rozdiel od Reduxu je však Vuex vytvorený špeciálne pre framework Vue.

Táto knižnica funguje na základe pár konceptov.

- State: objekt, ktorý obsahuje všetky dáta, ktoré majú byť dostupné v rámci aplikácie
- Getter: funkcia používaná na prístup ku dátam uloženým v State.
- Mutations: slúžia na obsluhu zmeny stavu v State
- Actions: Funkcie využívané na vyvolanie mutácií
- Modules: umožňuje rozdeliť jeden globálny stav aplikácie na viacero menších, čím sa pri veľkých aplikácia umožňuje lepšie udržať prehľadnosť kódu

## 4.5 História frameworku

Prvá verzia frameworku začala byť vyvíjaná v roku 2013 programátorom Evanom You, ktorý bol v tom čase zamestnancom spoločnosti Google. Pôvodný plán bol zanechať všetky obľúbené schopnosti Angularu, a odstrániť koncepty, ktoré vyžadujú dlhé štúdium pre ich používanie, a znížiť jeho veľkosť a robustnosť.

Verzia 1.0 vyšla v roku 2015, a na rozdiel od Angularu a Reactu zaznamenala okamžitý úspech. Krátko po vydaní si ako svoju oficiálnu frontendovú technológiu vybral Vue aj veľmi obľúbený framework Laravel.

Verzia 2.0 vyšla v roku 2016, a najvýraznejšou zmenou bola pridanie renderovania pomocou technológie Virtual DOM. [36]

V súčasnej dobe sa Vue nachádza vo verzií 2.6, a na hosťujúcom portáli github.com má vyše 138 tisíc hviezdičiek.

# II. PRAKTICKÁ ČÁST

# 5 UKÁŽKOVÉ APLIKÁCIE

Pre demonštrovanie používania a funkcií každého z frameworkov a ich následné porovnanie, boli vytvorené tri aplikácie s totožnou funkčnosťou. Cieľom troch rovnakých aplikácií je prehľadné porovnanie riešení obdobných problémov. Jedná sa o dochádzkový systém, ktorý by mohol byť použiteľný v malých firmách. Tento software umožňuje uchovávanie základných informácií o dochádzke zamestnanca. Súčasťou vývoja softwaru by mali byť zadefinované požiadavky na výsledný produkt.

## 5.1 Funkčné a nefunkčné požiadavky

Požiadavky sa delia na funkčné a nefunkčné. Funkčné požiadavky popisujú správanie a funkcie softwaru. Nefunkčné požiadavky určujú vlastnosti a obmedzujúce podmienky aplikácie.

### 5.1.1 Funkčné požiadavky

- Aplikácia bude umožňovať registráciu užívateľov
- Vytvorený užívateľ sa bude môcť do aplikácie prihlásiť
- Užívateľ bude môcť administrovať stav svojej prítomnosti v práci - bude môcť zaznačiť príchod do práce, odchod na prestávku, alebo odchod z práce
- Budú dostupné informácie o aktuálnom stave prítomnosti na základe posledné vykonanej akcie.
- Užívateľ bude vidieť aktuálny stav ostatných registrovaných užívateľov v práci.
- Budú dostupné informácie o odpracovanom čase za daný mesiac, a informácia o zostávajúcom čase do splnenia úväzku.
- Užívateľ bude mať prehľad o svojich posledných vykonaných akciách v rôznych zobrazeniach – listovom alebo kalendárnom
- Stránka bude poskytovať rýchly prehľad o uvedených informáciách na domovskej stránke, pričom každá sekcia bude mať vlastnú stránku, kde budú tieto informácie zobrazené podrobne.

### 5.1.2 Nefunkčné požiadavky

- Bude sa jednať o single page aplikáciu.
- Na vývoj bude použitý JavaScript framework na vývoj SPA.
- Každý z frameworkov (Angular, React a Vue) bude vo svojej najnovšej verzií.

- Aplikácia bude komunikovať z externým serverom pomocou REST API
- Na vytvorenie externého serveru bude použitá technológia .NET Core a SQL. Tento server sa bude starať o ukladanie dát.

## 5.2 Externá serverová aplikácia

Pri vytváraní single page aplikácie sa táto aplikácia stará o tzv. frontendovú časť. Frontendová časť nedokáže priamo spolupracovať a komunikovať s databázou, a teda ani nedokáže dlhodobo uchovávať žiadne dáta. Preto budú všetky tri vytvorené aplikácie napojené na spoločnú serverovú aplikáciu, ktorá má na starosti komunikáciou s databázou.

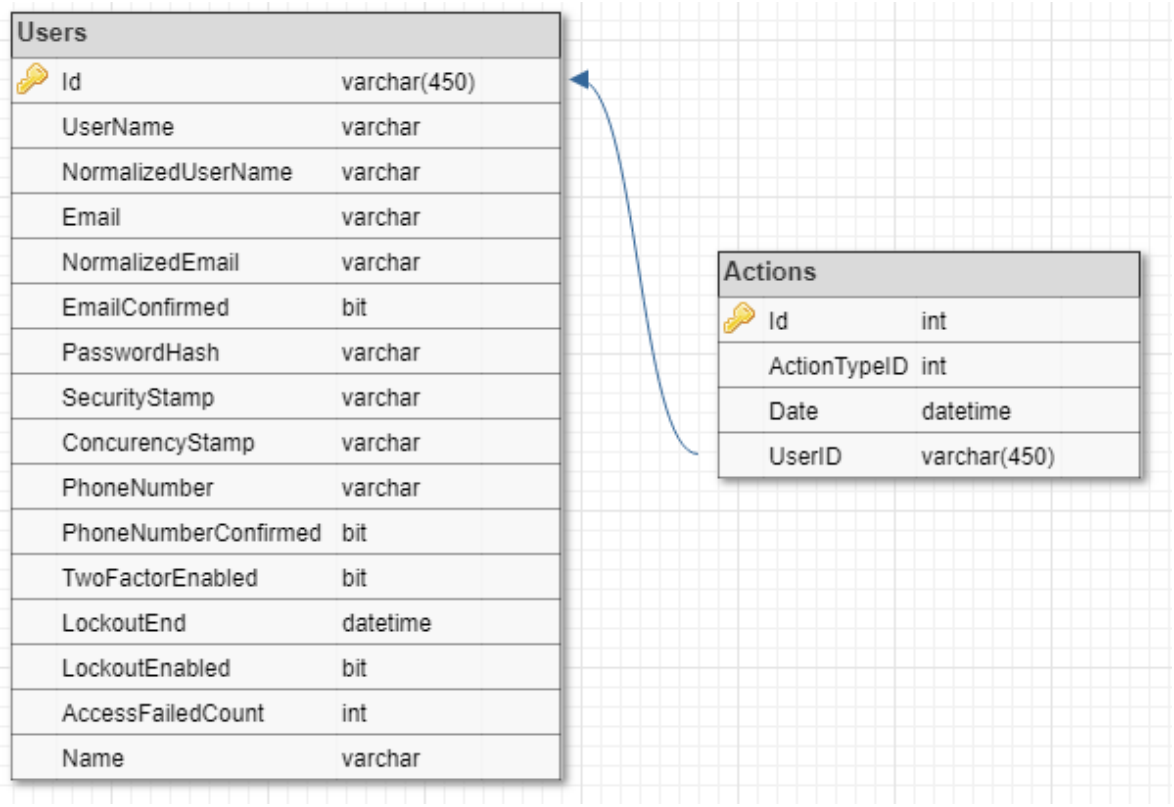
### 5.2.1 Použité technológie

Serverová aplikácia bola vytvorená v technológií .NET Core vo verzií 2.2 s použitím Entity Framework. Entity Framework je object-relational mapping (ORM) technológia. Stará sa o vytvorenie databázy na základe vytvorených modelov, a následnú komunikáciu s ňou. Pomocou EF vytvárame aj jednotlivé dotazy do databázy. Z výsledkov týchto dotazov potom EF automaticky vytvorí objekt, s ktorým sa môže pracovať v kóde bez zložitého mapovania. Keďže všetky dotazy je možné písať pomocou technológie LINQ (Language Integrated Query), odpadá nutnosť používania jazyka SQL.

### 5.2.2 Databáza

Databáza sa skladá z dvoch tabuliek. Tabuľka Actions, kde sa ukladajú jednotlivé záznamy zamestnancov. Teda napríklad ich príchody a odchody. Ukladajú sa informácie o type akcie, čase vykonania tejto akcie, a užívateľ, ktorý túto akciu vytvoril.

V tabuľke Users sa ukladajú zaregistrovaní užívatelia. Pre registráciu a prihlasovanie používateľov používam vstavanú funkcionalitu .NET Core frameworku, ktorá uľahčuje autorizáciu na základe tzv. JSON Web Token, alebo JWT. Táto funkcionalita ovplyvnila aj vytvorenie tabuľky Users. Vytvorený model, na základe ktorého je táto tabuľka vytvorená, rozširuje model *IdentityUser* z knižnice *Microsoft.AspNetCore.Identity*. Tento model je nutný ku správnemu fungovaniu autorizácie od .NET Core.




Obrázok 2. Schéma databáze serverovej aplikácie

### 5.2.3 REST API

Serverová aplikácia vytvorí REST API, pomocou ktorého bude komunikovať s jednotlivými SPA aplikáciami.

Skratka REST predstavuje pojem Representational State Transfer. Je to architektúra rozhrania, ktorá definuje komunikáciu medzi nezávislými aplikáciami. REST implementuje štyri základné metódy, pomocou ktorými si tieto aplikácie vymieňajú dáta, v prípade tejto aplikácie pomocou HTTP protokolu. Tieto metódy sa označujú skratkou CRUD. Jednotlivé metódy sú:

- Create – vytvorenie dát pomocou metódy POST
- Retrieve – získanie dát pomocou metódy GET
- Update – aktualizácia dát pomocou metódy PUT
- Delete – zmazanie dát pomocou metódy DELETE

V aplikácií sú vytvorené kontrolery, ktoré obsluhujú prácu s dátami z jednotlivých tabuliek, existuje teda *UserController* a *ActionController*. V týchto kontroleroch sú zadefinované endpointy, pomocou ktorých komunikujú SPA aplikácie so serverom. Z SPA aplikácie

cie vytvoríme request na daný endpoint. Ten tento request spracuje a zapíše alebo upraví zadané dáta do databázy, alebo vráti žiadané dáta vo formáte JSON.

Endpointy z *ActionsController* sú:

- GET *api/Actions* – výpis všetkých akcií
- GET *api/Action/User/{id}* – výpis akcií ktorý vytvoril užívateľ so zadaným id
- GET *api/Action/{id}* – výpis detailu akcie so zadaným id
- PUT *api/Action/{id}* – úprava akcie so zadaným id
- POST *api/Actions* – vytvorenie novej akcie
- DELETE *api/Action/{id}* – zmazanie akcie s daným id

Endpointy z UsersControlle sú:

- GET *api/Users* – výpis všetkých užívateľov
- GET *api/Users/{id}* – detail užívateľa so zadaným id
- GET *api/Users/Actions* – výpis všetkých užívteľov spolu s poslednou akciou ktorú vykonali
- PUT *api/Users/{id}* – úprava užívateľa so zadaným id
- POST *api/Users* – vytvorenie nového užívateľa, používané na registráciu
- DELETE *api/Users/{id}* – vymazanie užívateľa so zadaným id
- POST *api/Users/Login* – používa sa pre prihlásenie užívateľa. Prijíma prihlasovacie údaje (login, heslo) a vráti detail prihláseného užívateľa spolu s autorizačným tokenom, ktorý sa používa na overenie identity pri ďalších volaných endpointoch.

## 5.3 Nutné nástroje pre vývoj SPA

Pred začatím vývoja je nutné nainštalovať program Node.js. Node.js je runtime prostredie, v ktorom budú bežať aplikácie lokálne v počítači na ktorom sa aplikácia vyvíja. Momentálne sa nachádza vo verzií 10.15. Pomocou tohto programu sa inštalujú do aplikácií aj Node moduly, teda rôzne rozšírenia a knižnice.

Na písanie samotnej aplikácie je nutný textový editor. Existujú mnohé možnosti, platené aj bezplatné. Jedná sa o editory napr. Sublime Text, Brackets, Atom a ďalšie. Momentálne najobľúbenejší, s najbohatším výberom rôznych pluginov, ktoré uľahčujú vývoj je však Visual Studio Code. Preto bol použitý práve ten.

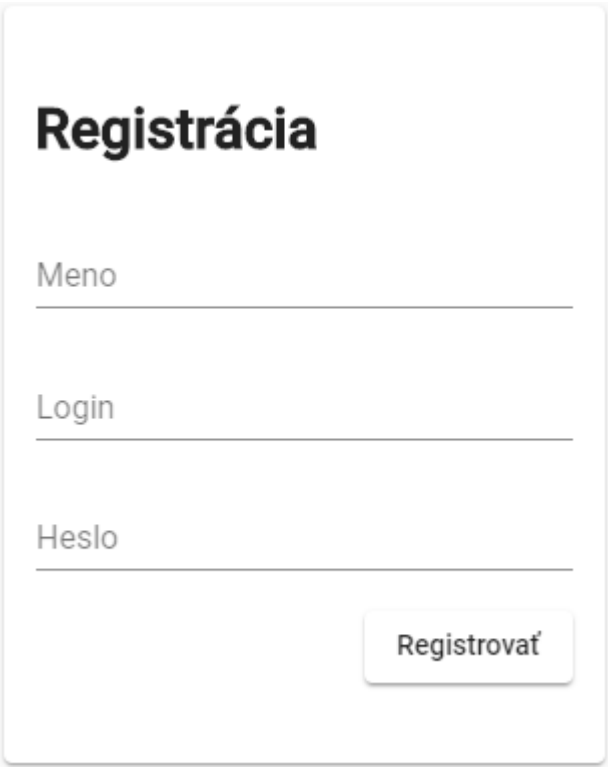
Pri písaní aplikácií bol využitý aj verzovací systém Git. Významne zjednodušuje prácu v tíme, ale aj pri práci jednotlivca, a ponúka prehľad o vykonaných zmenách. Na prácu s Gitom bol použitý program SourceTree, a git repozitár bol hostovaný na serveri GitHub.com

## 5.4 Funkcionalita aplikácie

V tejto časti sú popísané jednotlivé časti aplikácie a ich funkcionalita. Pre zachovanie čo najväčšej podobnosti všetkých troch aplikácií, a pre ukážku práce z externými knižnicami, používajú všetky aplikácie tzv. Material design. Je to knižnica, ktorá v sebe obsahuje rôzne UI elementy, ako napríklad inputy, dropdowny a pod. Dokážeme pomocou nej ovládať aj layout aplikácie.

### 5.4.1 Registrácia

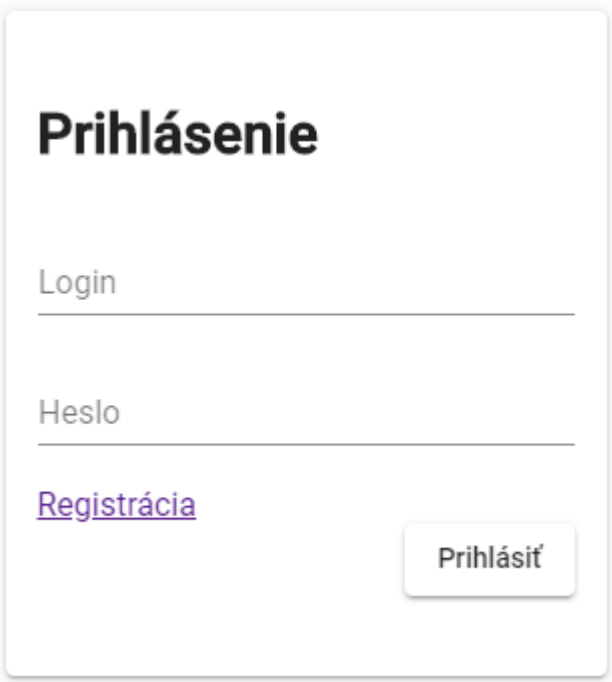
Užívateľ si môže založiť účet v aplikácií pomocou registračného formuláru. Do políčka meno vloží svoje celé meno. Do políčka Login zadá užívateľ svoj email, pomocou ktorého sa bude aplikácie prihlasovať. Do políčka Heslo zadá užívateľ svoje minimálne šesť miestne heslo. Po úspešnej registrácií je presmerovaný na formulár prihlásenia.




Obrázok 3. Registračný formulár

### 5.4.2 Prihlásenie

Ak užívateľ už vlastní účet, alebo sa práve zaregistroval, môže sa do aplikácie prihlásiť pomocou prihlasovacieho formuláru. Do políčka Login zadá užívateľ svoj email, ktorý slúži aj ako prihlasovacie meno. Do políčka Heslo zadá heslo, ktoré si zvolil pri registrácii. Po úspešnom prihlásení je užívateľ presmerovaný na domovskú stránku aplikácie, ktorá sa v aplikácií nazýva Dashboard. V prípade že užívateľ nie je prihlásený, žiadne ďalšie časti aplikácie nie sú dostupné. Na formulári registrácie a prihlasovania nie sú dostupné žiadne ďalšie odkazy, a užívateľ sa nedostane ďalej ani manuálnym prepísaním adresy v prehliadači.




Obrázok 4. Prihlasovací formulár

### 5.4.3 Dashboard

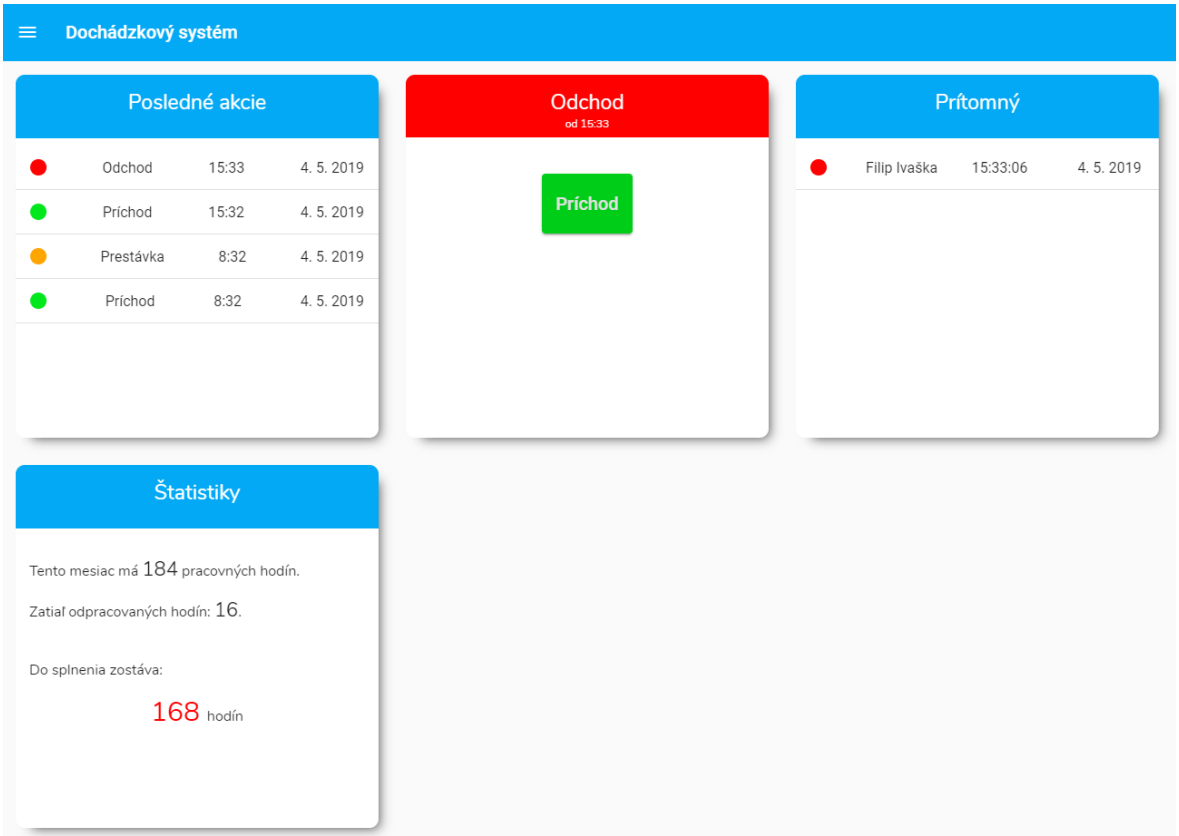
Po prihlásení sa užívateľovi otvoria ďalšie časti aplikácie. Ako prvou je domovská stránka – Dashboard. Ten je navrhnutý pomocou systému kariet, kde každá karta zobrazuje náhľad na nejaký typ informácií.

- Karta, ktorá označuje aktuálny stav prítomnosti v práci, a na základe tohto stavu mení farbu svojho záhlavia. Umožňuje takisto vykonať akcie, teda príchod, odchod a odchod na prestávku.
- Karta ukazuje informácie o posledných šiestich vykonaných akciách.
- Karta ukazujúca posledných šiestich užívateľov, ktorým sa zmenil stav prítomnosti

- Karta s informáciou o počte pracovných hodín v aktuálnom mesiace, počet odpracovaných hodín užívateľom, a zostávajúci počet hodín do splnenia úväzku

Tento štýl dizajnu bol zvolený kvôli jeho jednoduchej škálovateľnosti. V prípade pridania ďalšej funkcionality, stačí pridať ďalšiu kartu, bez akéhokoľvek zásahu do už hotového užívateľského rozhrania.

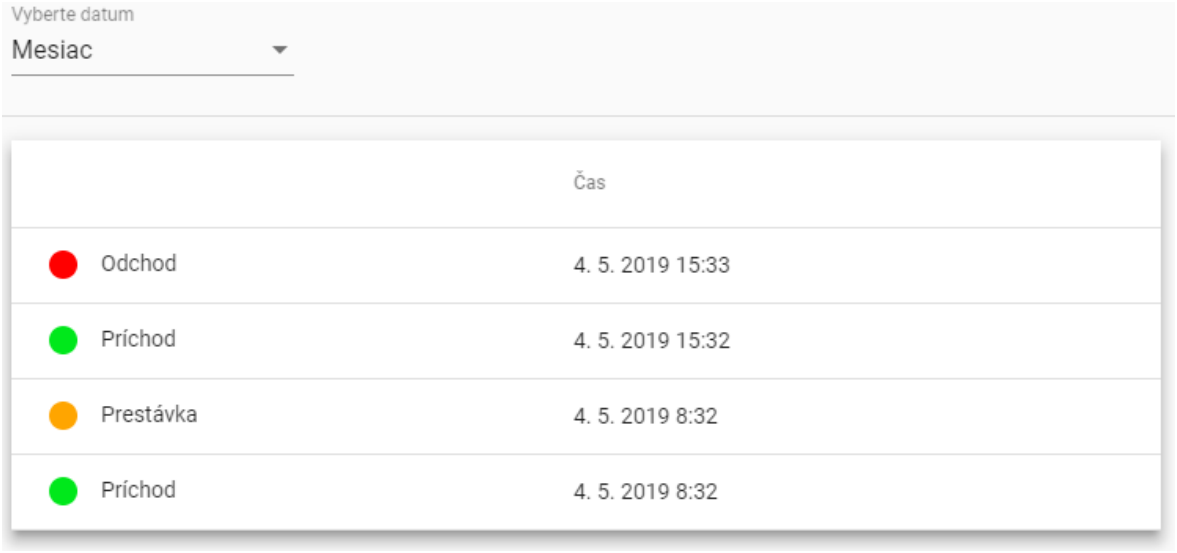
Tlačidlom v ľavom hornom rohu je možné rozbaliť menu. Toto menu obsahuje odkazy na ďalšie sekcie stránky. Tieto sekcie rozširujú informácie už poskytnuté v kartách, ukazujú ich však bez obmedzenia na posledné záznamy, prípadne sa dajú informácie zobraziť v iných pohľadoch.




Obrázok 5. Domovská stránka aplikácie

### 5.4.4 Záznamy

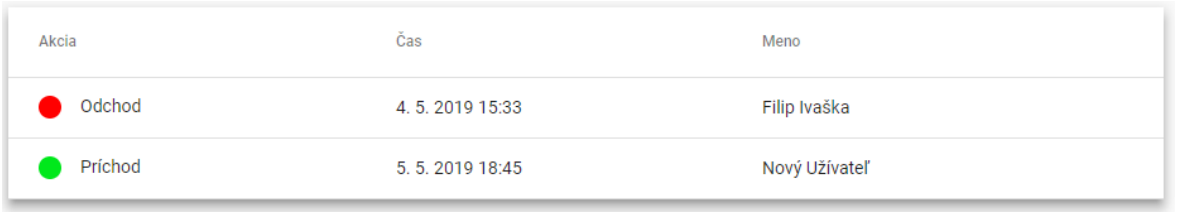
Na tejto stránke má užívateľ možnosť prehliadať si sebou vytvorené záznamy za určený časový interval. Tento interval môže byť dnešný deň, aktuálny týždeň, alebo aktuálny mesiac. Jednotlivé záznamy informujú o type akcie, a čase jej vykonania. Každý typ akcie je farebne oddelený pre väčšiu prehľadnosť.




Obrázok 6. Listové zobrazenie vykonaných akcií za posledný mesiac

### 5.4.5 Prítomný

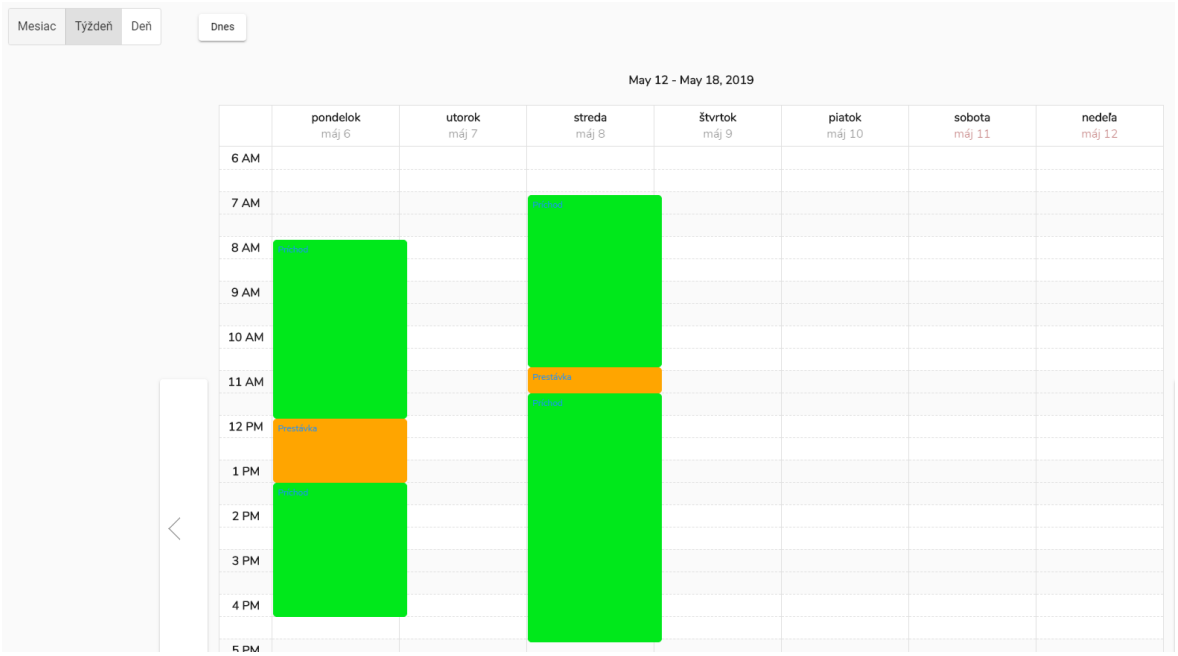
Stránka prítomnosť obsahuje výpis všetkých registrovaných užívateľov v systéme. Ku každému záznamu sú informácie o mene užívateľa, čase poslednej vykonanej akcie, a o type vykonanej akcie. Podobne ako pri stránke so záznamami, aj v tomto prípade sú jednotlivé akcie oddelené farebne. Na prvý pohľad je tak jasné, kto práve v práci je, a kto je momentálne neprítomný.




Obrázok 7. Výpis registrovaných užívateľov a ich aktuálny stav

### 5.4.6 Kalendár

Kalendár slúži na prehľadné grafické zobrazenie stavu prítomnosti užívateľa v práci. Toto zobrazenie podporuje výpis cez mesačný kalendár, alebo pomocou podrobného rozpisu dní. A to buď v rámci jednotlivých dní, alebo celého týždňa. Toto zobrazenie dá na prvý pohľad predstavu o dochádzke za vybraný časový úsek.




Obrázok 8. Kalendár v týždennom zobrazení s udalosťami

# 6 ANGULAR APLIKÁCIA

Pre nainštalovanie frameworku a správu projektu sa používa nástroj Angular CLI. V prvom kroku vytvoríme projekt. Na mieste na disku, kde chceme projekt vytvoriť otvoríme konzolu a príkazom *ng new AttendantAngular* vytvoríme projekt s názvom AttendantAngular. Angular CLI ponúka aj ďalšie pomôcky, napríklad automatické generovanie komponentov, služieb, pipe a pod. Napríklad príkaz *ng g c home* vytvorí zložku s názvom *home*, a v ňom 3 súbory – pre komponent, štýly a šablónu.

## 6.1 Zdieľané časti aplikácie

V rámci prehľadného členenia kódu je vhodné vytvoriť vlastný modul pre kusy kódu, ktoré sa používajú naprieč celou aplikáciou. V tomto module budú zadeklarované komponenty, služby, pipe, modely a konštanty ktoré chcem aby boli viditeľné pre celu aplikáciu. Vzhľadom na to, že pre tieto spoločné veci vyčleníme jeden modul, do ostatných modulov potom stačí importovať tento spoločný, a zdieľané veci budú automaticky použiteľné, bez nutnosti importovať každú časť kódu zvlášť. Preto bol pomocou príkazu *ng g m shared* vytvorený modul *shared.module.ts*, ktorý bude slúžiť na tieto účely.

### 6.1.1 Services

Services tvoria dôležitú časť Angular aplikácie. Dôvod ich používania je podrobnejšie popísaný v teoretickej časti tejto práce.

**DataService** – táto služba bude využívaná na uľahčenie práce s HTTP requestami. Jej existencia nie je nutná, HttpClient sa môže volať priamo z každého komponentu. Pomocou tejto service však dokážeme zjednodušiť zápis a volanie jednotlivých requestov.

```
@Injectable({
  providedIn: 'root'
})
export class DataService {
constructor(private http: HttpClient) { }
public post<T>(entityName: string, body: Object = {}):
Observable<T> {
  const promise =
  this.http.post<T>(`${environment.api_url}/${entityName}`,
JSON.stringify(body)).pipe(share());
  return promise;
  }
}
```

Keďže sa jedná o službu, ktorá musí byť prístupná komponentom pomocou dependecy injection, musí byť jej deklarácia označená dekorátorom *@Injectable()*. Do konštruktoru tejto služby pomocou dependency injection posielame *HttpClient*, modul ponúkany Angularom na správu http requestov, a jeho instanciu si uložíme do premennej s názvom *http*.

V service je deklarovaná metóda post, ktorá ako parametre prijíma *entityName* a parameter *body*. *EntityName* je typu *string* a predstavuje adresu, na ktorej sa žiadaný endpoint nachádza. *Body* je typu *Object*, a predstavuje telo requestu. Návratový typ metódy je Observable<T>. V tomto kontexte je <T> typový parameter. Doň pri volaní metódy pošleme očakávaný návratový typ. V samotnom tele metódy, sa pomocou instancie HttpClienta zavolá žiadaný endopint. Do tela tohto requestu sa pošle objekt, ktorý sme zadali ako parameter funkcie, a ten sa prekonvertuje do JSON formátu. Obdobným spôsobom sú vytvorené metódy pre všetky potrebné typy requestov.

V samotnom komponente potom zavoláme niektorú z metód service nasledovne

```
this.dataService.get<UserAction[]>(`${Constants.Users}/actions`).s
ubscribe(result => {
  this.userActions = result.slice(0, 6);
});
```

V tomto volaní posielame GET request na adresu pre získanie všetkých vykonaných akcií. Vzhľadom na to, že návratový typ funkcie je *Observable*, teda asynchrónna premenná, musíme k takejto premennej vykonať subscribe. Telo tejto metódy sa vykoná potom, ako do asynchrónnej premennej príde hodnota. V tomto konkrétnom prípade sa do premennej *userActions* zapíše prvých 6 prvkov listu z výsledku, ktorý nám poslal server.

**JwtService** – Service pre spravovanie autentifikačného tokenu, ktorý nám zaslal server po prihlásení. Pomocou tejto service je možné tento token uložiť do lokálnej pamäte prehliadača, a pracovať s ním pri vytváraní requestov.

**AuthService** – Slúži pre obsluhu prihlásenia a uloženie informácií o aktuálne prihlásenom užívateľovi.

**AuthGuard** – Špeciálny typ service, ktorý sa používa pre ochranu jednotlivých adries v aplikácií, ktoré nemá užívateľ vidieť, pokým nie je prihlásený. Táto služba sa zavolá pri každej aktivácií novej cesty v prehliadači, a skontroluje či má na ňu užívateľ právo. Ak to právo nemá, tak je automaticky odkázaný na stránku pre prihlásenie.

### 6.1.2 Components

V tejto zložke sa nachádzajú komponenty ktoré sú zdieľané a viditeľné na celej stránke, ako napríklad bočný panel, ktorý plní funkciu menu.

### 6.1.3 Models

Slúži pre uloženie modelov, ktoré sa používajú v rámci aplikácie

### 6.1.4 Interceptors

Interceptory sú vo frameworku Angular špeciálny typ service. Prerušia každý odchádzajúci request z modulu, kde je tento interceptor deklarovaný, a dokážu tento request modifikovať. Používa sa hlavne pri autentifikácií, kedy do hlavičky requestu vloží autorizačný token. Bez tohto tokenu by nám server nevrátil žiadané dáta, a slúži ako overenie že užívateľ je prihlásený.

```
let headers = new HttpHeaders({
  'Authorization': `Bearer ${window.localStorage['jwtToken']}`,
});
```

### 6.1.5 Enums

Slúži pre uloženie konštánt. Je vhodné používať konštanty na čo najväčšie množstvo veci ktoré nemenia svoj stav na základe zmeny v aplikácií. Ak je potrebná zmena, napríklad sa zmení adresa pre endpointy kontroleru *UsersController*, stačí túto zmenu vykonať len na jednom mieste.

### 6.1.6 Pipes

Miesto pre programátorom vytvorené pipe. Pipe sa pri implementácií musia označiť dekorátorom *@Pipe* a musia implementovať interface *PipeTransform*.

```
@Pipe({
  name: 'actionType'
})
export class ActionTypePipe implements PipeTransform {

  transform(value: any, translated = true): string {
    if (value === ActionTypes.In) {
      return translated ? 'Príchod' : 'in';
    }
```

```
    if (value === ActionTypes.Out) {
      return translated ? 'Odchod' : 'out';
    }
    if (value === ActionTypes.Break) {
      return translated ? 'Prestávka' : 'break';
    }
  }
}
```

Samotnú úpravu definuje vo funkcií transform. Konkrétne v pipe s názvom *ActionTypePipe* sa prijíma ako hodnota číselný typ akcie, a vráti sa vyjadrenie tejto akcie v textovej podobe. Parameter *translated* určuje, či chceme aby sa toto vyjadrenie vrátilo preložené, alebo po anglicky. Preložený text sa môže použiť pri výpise akcie na obrazovku, anglický text sa môže použiť napríklad ak chceme nejakému elementu na základe typu akcie priradiť určitú HTML triedu.

## 6.2 Routing

Router je priamo súčasťou Angularu, nie je teda nutné doinštalovať žiadne balíčky, ani knižnice. Pre vytvorenie jednotlivých ciest je nutné vytvoriť konštantu, pozostávajúcu z listu adries ktoré chceme zaregistrovať. Každá položka listu musí byť typu Routes. Obsahuje informácie o adrese, na ktorá cestu aktivuje, na ktorý modul má odkazovať a parameter, ktorý určuje, či je cesta chránená pomocou nejakej služby. V prípade tejto aplikácie sú chránené všetky cesty, okrem cesty ktoré odkazujú k *AuthModulu*, aby boli dostupné bez prihlásenia. Táto konštanta je potom importovaná do hlavného *AppModule*. Moduly a komponenty ktoré boli sputené po aktivácií nejakej cesty sa vypisujú pomocou komponentu *RouterOutlet*.

## 6.3 Autentifkácia

O autentifikáciu sa stará AuthModule. Ten v sebe obsahuje deklaráciu komponentov pre registráciu, a prihlásenie používateľa. Na vytvorenie týchto formulárov som použil UI komponenty, ktoré ponúka knižnica Angular Material. Využil som ich pre jednotlivé textové polia. Vďaka inputom z tejto knižnice je uľahčené aj zobrazenie validačných hlášok.

Na prenos dát medzi šablónou a komponentom som využil obojsmernú väzbu dát. Pri prihlásení je vytvorený objekt *credentails* typu *Credentials*, ktorý obsahuje vlastnosti Email a Password. Tieto vlastnosti sú namapované na jednotlivé polia pomocou property

*[(ngModel)]*. Vďaka tomu, akonáhle užívateľ zmení hodnotu v týchto poliach, sa táto zmena prejaví aj v objekte *credentials*. Všetky polia v tomto formulári sú povinné.

Po kliknutí na prihlásenie sa pomocou *DataService* zavolá endpoint ktorý obsluhuje prihlásenie. Po úspešnom prihlásení sa zo serverovej aplikácie vráti JSON objekt obsahujúci informácie o overenom používateľovi a autorizačný token. Tento výsledný objekt sa pošle do *AuthService*, kde sa prihlásenie spracuje. Autorizačný token sa uloží do lokálnej pamäti prehliadača. Ten sa potom následne pomocou *AuthIntercepor* vloží do hlavičky každého requestu, ktorý vyžiada aplikácia. Bez neho by server odmietol request obslúžiť, a vrátil by chybovú hlášku 401 Unauthorized.

```
token: "eyJhbGciOi-
JIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySUQiOiIzYjA2YmU5My01ZDUyLTQ2
YTMt…"
user: {
   accessFailedCount: 0
   concurrencyStamp: "69e93766-2e19-439d-968c-3933a6234ef7"
   email: "filip.ivaska@gmail.com"
   emailConfirmed: false
   id: "3b06be93-5d52-46a3-ae87-9f31eefdb16f"
   lockoutEnabled: true
   lockoutEnd: null
   name: "Filip Ivaška"
   normalizedEmail: "FILIP.IVASKA@GMAIL.COM"
   normalizedUserName: "FILIP.IVASKA@GMAIL.COM"
   password: null
   passwordHash: "AQAAAAEAA…"
   phoneNumber: null
   phoneNumberConfirmed: false
   securityStamp: "3TZYGUZGPJEPMI5CPNT4M3YMJIEVLNUR"
   twoFactorEnabled: false
   userName: filip.ivaska@gmail.com
}
```

V ukážke kódu vyššie je názorne ukázaný objekt, ktorý sa vráti zo serverovej aplikácie po úspešnom prihlásení. JSON objekt obsahuje dve časti. Prvou je *token*, ktorá obsahuje autorizačný token, a položka *user*, ktorá obsahuje dáta o užívateľovi.

## 6.4 Dashboard

Modul obsluhujúci domovskú stránku aplikácie. Sú v ňom deklarované komponenty jednotlivých kariet. Obsahuje komponent DashboardComponent, ktorý slúži ako nadradený komponent pre jednotlivé karty, a stará sa o ich zobrazenie a jednotné štýlovanie.

### 6.4.1 Komponent domovskej stránky

Stará sa o zobrazenie kariet na domovskej obrazovke. V prípade, že užívateľ vykoná akciu, napríklad zaznamená príchod do práce, stará sa o aktualizáciu stavu v ostatných kartách. Táto aktualizácia nastane potom, ako sa v kontroleri pre obsluhu akcií *DashboardActionsController*, spustí event s názvom *actionUpdated*.

```
<div class="widget-container">
  <div class="widget">
    <last-actions></last-actions>
  </div>
  <div class="widget">
     <dashboard-actions (actionUpda-
     ted)="updateLastActions()"></dashboard-actions>
  </div>
  <div class="widget">
    <dashboard-attendance></dashboard-attendance>
  </div>
  <div class="widget">
    <dashboard-stats></dashboard-stats>
  </div>
</div>
```

## 6.5 Prítomný a Záznamy

Tieto moduly sa starajú o zobrazenie prítomných zamestnancov a tabuľkové zobrazenie akcií vykonaných užívateľom. Na zobrazenie je použitá tabuľka z Material UI. Táto tabuľka nie je klasickým komponentom, ktorý prijíma *Input()* parametre. Jedná sa o direktívu, ktorá transformuje klasický HTML element *table* do žiadanej podoby. Pre zobrazenie každého stĺpca je použitý Angular element *ng-container*, ktorý sa používa v prípade, ak chceme obaliť časť HTML kódu do elementu, na ktorý môžeme priradiť direktívy alebo štrukturálne direktívy (napr. **ngIf*), ale nechceme zbytočne pridávať ďalší level renderovaných elementov. Element *ng-container* je totižto nerenderovaný.

Tabuľky pracujú s dvomi premennými, a to je list, ktorý obsahuje kľúče jednotlivých stĺpcov, a samotné dáta ktoré sa do tabuľky majú vypísať.

## 6.6 Kalendár

V kalendárnom zobrazení je možnosť pozrieť si svoje akcie v prehľadnom kalendárnom zobrazení. Kalendár podporuje mesačné, týždenné a denné zobrazenie týchto udalostí.

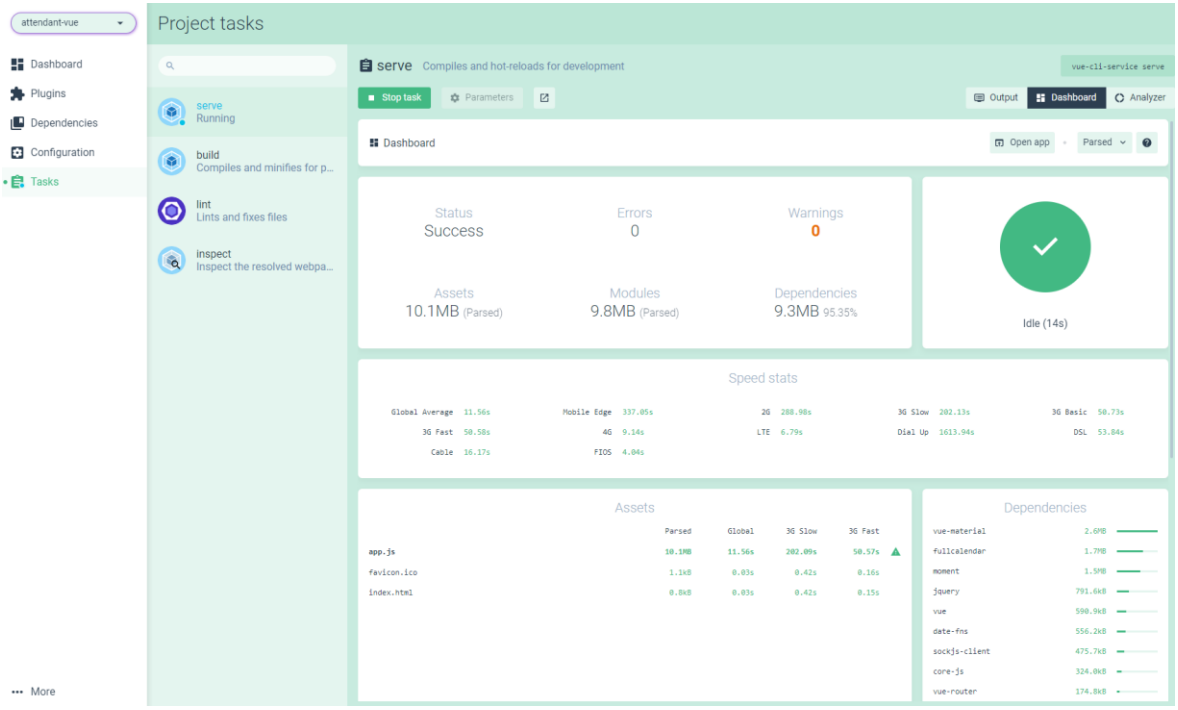
Pre toto zobrazenie bola použitá knižnica dostupná z adresy https://mattlewis92.github.io/angular-calendar/docs/index.html. Kalendár funguje na princípe stiahnutia akcií vykonaných užívateľom, a ich následnú transformáciu tak, aby boli použiteľne ako udalosti v kalendári. Rôzne zobrazenia sa vyžadujú rôzne transformácie. Pri mesačnom výpise sú akcie zobrazené ako jednoduché udalosti po rozkliknutí žiadaného dátumu. Pri týždennom a dennom výpise je nutnosť pretransformovať jednotlivé akcie tak, aby spolu tvorili jednotlivé udalosti, ktoré majú svoj začiatok a koniec.

Každé zobrazenie sa zobrazuje pomocou vlastného komponentu, je teda možnosť formátovať každé zvlášť. Prepínanie týchto zobrazení je nutné naprogramovať vývojárom, nestará sa o to žiaden komponent z knižnice. Prepínanie je riešené pomocou Material UI komponenty *mat-button-toggle-group*, ktorá vytvorí element typu radio-button.

# 7 VUE APLIKÁCIA

Pre inštalovanie a správu Vue aplikácií sa používa Vue CLI. Používa sa pre vytvorenie aplikácie, prípadne pre pridanie rôznych pluginov do nej. Pre vytvorenie aplikácie sa používa príkaz *vue create*. Ak teda chceme vytvoriť aplikáciu, použijeme napríklad príkaz *vue create attendant-vue*. Tento príkaz vytvorí aplikáciu v zložke, v ktorej sa práve terminál nachádza.

Veľmi užitočnou funkciou Vue CLI je aj Vue UI. Táto funkcia umožňuje spustiť Vue CLI v grafickom rozhraní v prehliadači. Je teda možné ovládať celú funkcionalitu pomocou CLI s grafického rozhrania. Jedná sa teda o vytvorenie nového projektu, alebo aj spúšťanie jednotlivých úloh v existujúcom projekte, alebo aj štatistiky existujúceho projektu.




Obrázok 9. Štatistika projektu zobrazená pomocou Vue UI

## 7.1 Zdieľané časti aplikácie

V rámci zdieľaného kódu aplikácie vo frameworku Vue sa nachádzajú napríklad konštanty, alebo plugin Vuex, ktorý sa stará o udržiavanie globálneho stavu aplikácie.

### 7.1.1 Axios

Plugin axios sa stará o prácu s http requestami. Obsahuje metódy pre všetky typy requestov -GET, POST, PUT aj DELETE. Môžeme vďaka nemu nastaviť základnú adresu URL, kto-

rá sa nemení alebo aj upravovať hlavičky všetkých odchádzajúcich requestov, a pridať napríklad autorizačný token. Podobnú funkcionalitu vo frameworku Angular zastávajú interceptory.

```
Vue.prototype.$http = Axios;
const token = localStorage.getItem('jwtToken')
if (token) {
  Vue.prototype.$http.defaults.headers.common['Authorization']
= `Bearer ${token}`
}
```

V kóde vyššie registrujeme novú instanciu pluginu Axios, získame autorizačný token z lokálnej pamäte prehliadača, a ak tento token existuje, vložíme ho do každej hlavičky requestu pod kľúčom *Authorization*.

### 7.1.2 Vuex

Vuex je knižnica starajúca sa o manažovanie stavu aplikácie. V rámci tejto aplikácie je použitá na udržanie stavu prihlásenia, a na zobecnenie funkcií, ktoré sa starajú o prepojenie s API. Nachádza sa v zložke *store*. Je importovaná v súbore main.js, a zaregistrovaná pri registrovaní aplikácie.

```
import { store } from './store/store'

new Vue({
  render: h => h(App),
  router: router,
  store: store
}).$mount('#app')
```

V kóde vyššie môžeme vidieť import súboru *store.js* do premennej *store*, a poslanie tejto premennej do parametru *store* v príkaze new *Vue()*, ktorá sa stará o vytvorenie aplikácie.

**State** – V tomto objekte sú udržiavané stavy o dátach dôležitých pre chod aplikácie. Sú to dáta o autorizačnom tokene, uloženom *userID* a objekt, ktorý ukladá informácie o užívateľovi. Ukážka použitia je v nasledujúcom kóde, kde sa počiatočný stav hodnoty token incializuje načítaním z lokálnej pamäte prehliadača. Ak prehliadač túto hodnotu uloženú nemá, do premennej sa uloží hodnota *null*, a znamená to, že užívateľ nie je prihlásený.

```
state: {
    token: localStorage.getItem('jwtToken') || null,
  },
```

**Getters** – Pomocou tu deklarovaných funkcií môžeme pristupovať ku stavu aplikácie z komponentov. Vďaka prístupu pomocou funkcie zabraňujeme zmene dát počas čítania týchto dát, napríklad keď sa k dátam snažia pristúpiť dva komponenty naraz.

```
getters: {
    loggedIn(state) {
      return state.token !== null
    }
  }
```

Getter *loggedIn()* vráti hodnotu, ktorá značí či je užívateľ prihlásený, na základe toho, či existuje hodnota v *state.token*.

**Mutations** – Zmena stavu aplikácie nie je povolená na priamo. Stav sa musí meniť pomocou tzv. mutácií. Jedná sa o jednotlivé funkcie, kde prvým parametrom je vždy stav aplikácie, a ďalšie parametre sú dobrovoľné, ak sú nutné ku správnemu vykonaniu mutácie.

```
mutations: {
    setToken(state, token) {
      state.token = token
    }
  },
```

Mutácia *setToken()*, preberá ako prvý povinný parameter stav aplikácie, a ako druhý parameter *token*, ktorý následné uloží do globálneho stavu aplikácie v premennej *token*.

**Actions** – Akcie sú funkcie, ktoré majú povolenie vyvolávať mutácie, a môžu sa v nich vykonávať asynchrónne operácie, ako napríklad poslanie requestu na server, čakanie na odpoveď, a jej spracovanie. Parametrom jednotlivých akcií je parameter *context*, ktorý obsahuje kontext aktuálneho stavu, vďaka ktorému je možné volať všetky mutácie v rámci Vuex. Mutácie sa teda nevolajú priamo, ale pomocou príkazu *commit(),* ktorý a ako prvý parameter prijíma názov mutácie, a ako druhý parameter dáta ktoré chceme poslať do ďalších parametrov mutácie.

```
actions: {
    loadUserDetail(context) {
        axios.get(`/users/${context.state.userID}`)
          .then(response => {
            context.commit('setUser', response.data)
          })
    }
  }
```

Akcia *loadUserDetail()* vysiela pomocou pluginu axios GET request na zadanú adresu. Po obdržaní výsledku je uložený do premennej *response*, a pomocou príkazu *context.commit()* je spustená mutácia *setUser()*, ktorá uloží výsledné dáta zo serveru do globálneho stavu aplikácie.

### 7.1.3 EventBus

Jednou z možností ako obsluhovať eventy vo Vue aplikácií je použitie EventBus. Pre Vytvorenie EventBusu, stačí vytvoriť jednoduchý súbor, ktorý potom naimportujeme do komponentov, kde ho chceme používať. Obsah súboru obsahuje len dva riadky kódu. Je nutné importovanie Vue, a následné vytvorenie novej instancie, a jej exportu.

```
import Vue from 'vue'
export default new Vue()
```

Pre vytvorenie eventu potom stačí vytvoriť event s ľubovoľným názvom, po vykonaní nejakej akcie, napríklad po kliknutí na tlačidlo.

```
EventBus.$emit('userLoggedOut')
```

Po spustení kódu vyššie sa vyvolá event s názvom *userLoggedOut*, a je prístupný v rámci celej aplikácie. Pre zachytenie tohto eventu v inom komponente stačí použiť funkciu *$on()*.

```
EventBus.$on('userLoggedOut', () => {
     this.showNavigation = false;
})
```

Po vyvolaní eventu sa spustí kód ktorý je v tzv. arrow funkcií, v tomto prípade sa nastaví hodnota premennej *showNavigation* na *false*.

### 7.1.4 Filtre

Filtre sú veľmi podobné pipe z frameworku Angular. Majú v podstate rovnakú funkciu, a veľmi podobný majú aj zápis. Používajú sa teda na formátovanie výpisu dát. Filter pre preklad typu akcie z číselného vyjadrenia na textovú podobu, má úplne identické telo samotnej transformácie. Líšia sa len v deklarácií a mieste kde sa deklaruje. Filter je vo frameworku Vue zadeklarovaný priamo v súbore *main.js*.

```
Vue.filter('actionFilter', function (value, translated = true)
{
  if (!value) return ''
  if (value === ActionTypes.In) {
    return translated ? 'Príchod' : 'in';
  }
  if (value === ActionTypes.Out) {
    return translated ? 'Odchod' : 'out';
  }
  if (value === ActionTypes.Break) {
    return translated ? 'Prestávka' : 'break';
  }
})
```

## 7.2 Routing

Cesty sa vo Vue aplikácií registrujú v súbore *main.js*. Je nutné vytvoriť konštantu so žiadanými cestami. Táto konštanta sa použije pri vytvorení novej instancie objektu VueRouter ktorá sa uloží do premennej. Pomocou tejto premennej dokážeme funkciou beforeEach zadefinovať spustenie ľubovoľného kódu pred aktivovaním cesty. Používa sa hlavne pre ochranu jednotlivých ciest, ak majú byť dostupné až po prihlásení.

```
router.beforeEach((to, from, next) => {
  if(openRoutes.includes(to.path)) {
      next()
  } else if (store.getters.loggedIn) {
      next()
  } else {
      next('/login')
  }
  })
```

Pri takejto implementácií sa pred aktivovaním cesty skontroluje stav prihlásenia. Ak je užívateľ prihlásený, cesta sa aktivuje, ak nie je, je presmerovaný na prihlasovací formulár.

## 7.3 Autentifikácia

Autentifikácia prebieha pomocou komponentu *Login.vue*. Obsahuje šablónu pre prihlasovací formulár, a logiku ktorá obsluhuje prihlásenie a uloženie tohto prihlásenia pomocou Vuex. Pre štýlovanie v rámci celej aplikácie je použitý CSS preprocessor SASS. Na dosiahnutie konzistentnosti UI je použitý Material UI.

## 7.4 Dashboard

Zložka Home obsahuje deklaráciu komponentov potrebných pre vykreslenie domovskej stránky. Pre každú kartu je vytvorený vlastný komponent. Karty, ktorých sa to týka, dynamicky reagujú na zaregistrovanie akcie. Ak teda zamestnanec zaregistruje svoj príchod, karty sú automaticky aktualizované na najnovšie dáta. Toto správanie je dosiahnuté pomocou funkcionality EventBus.

## 7.5 Prítomný a Záznamy

Pre tabuľkový výpis dát v týchto komponentoch je použitý komponent *md-table*. Ten sa používa trochu tradičnejšie, ako komponent pre tabuľky vo frameworku Angular. Jeden nadradený komponent obsahuje podriadené komponenty pre výpis jednotlivých elementov tabuľky -jednotlivé riadky, hlavičku, bunky a pod. Pre výpis všetkých riadkov sa cez dáta iteruje pomocou direktívy *v-for*.

Pre zjednodušenie práce s dátumami je použitá knižnica *date-fns*, ktorá obsahuje metódy pre zistenie, či je zadaný dátum súčasťou aktuálneho týždňa, mesiaca a podobne. Ušetrí veľke množstvo kódu a zbytočnej logiky, ktorou by sa každý komponent musel zaťazovať.

## 7.6 Kalendár

Kalendár je obsluhovaný komponentom Full Calendar dostupným z https://fullcalendar.io/. Tento kalendár umožňuje zobraziť kalendár v mesačnom týždennom alebo dennom zobrazení. Dokáže zobrazovať aj udalosti, a práve táto funkcionalita je v aplikácií použitá. Je nutnosť transformovať jednotlivé akcie na súvislé udalosti, ktoré potom komponent prijíma ako prop. O prepínanie zobrazenia sa stará samotný komponent, nebolo nutné túto funkcionalitu programovať.

# 8 REACT APLIKÁCIA

Na inštaláciu React projektu sa využíva balíček *create-react-app*. Nemá teda žiaden dedikovaný CLI ako ostatné dva frameworky. Preto pre vygenerovanie novej prázdnej aplikácie potrebujeme stiahnuť najskôr ten. Nainštaluje sa pomocou príkazu npm install -g create-react-app. Po stiahnutí a nainštalovaní ho môžeme použiť pre vytvorenie aplikácie pomocou príkazu *create-react-app attendant-react*. Lokálne potom spustíme aplikáciu príkazom *npm start.*

## 8.1 Zdieľané časti aplikácie

V zdieľanej časti aplikácie sa nachádzajú konštanty, jednotlivé služby a funkcie pre obsluhu autentifikácie.

### 8.1.1 Axios

Podobne ako Vue, ani React nemá implementovanú funkcionalitu pre obsluhu http requestov, je nutné túto funkcionalitu doplniť externou knižnicou. Knižnica, ktorá bola použitá pre Vue, je dostupná aj pre React, preto je použitá aj v tejto aplikácií. Ich funkcionalita je totožná.

### 8.1.2 Service

Service, alebo služby, v Reacte neexistujú ako dedikovaná funkcionalita frameworku, a nepoužívame ich v jednotlivých komponentoch pomocou dependency injection. Je však možnosť vytvoriť, a sčasti nahradiť funkcionalitu služieb vytvorením štandardných JavaScript funkcií, ktoré sú potom používané v komponentoch pomocou funkcie *import*. Takéto funkcie sa používajú aj ako náhrada funkcionality pipe v Angulare, prípadne filter vo frameworku Vue.

**Auth.js** – tento súbor v sebe deklaruje funkcie, ktoré sa starajú o funkciu prihlasovania a registrácie. Tento obsahujú jednotlivé funkcie pre prihlásenie a pre registráciu

**ActionFormatService.js** – Tento súbor deklaruje funkciu, ktorá napodobňuje funkcionalitu *ActionTypePipe* v Angulare a *actionFilter* vo Vue. Jedna sa teda o funkciu, ktorá dynamicky mení číselné vyjadrenie typu akcie na textové.

**Stores** – Obsahuje deklaráciu funkcie, ktoré obsluhujú ukladanie, prípadne zmazanie autorizačného tokenu pri prehlásení, respektíve pri odhlásení. Využíva sa aj pre zisťovanie či je užívateľ prihlásený.

## 8.2 Routing

Pre obsluhu routingu máme na výber z niekoľkých knižníc. Môže to byť oficiálna knižnica React Router Dom, alebo napríklad open source knižnica Reach Roter. V tejto aplikácií sa používa oficiálny React Router Dom.

Jednotlivé cesty sa definujú priamo v komponente *BrowserRouter*, ako jednotlivé komponenty *Route*, kde sa zadefinuje cesta ktorú majú obsluhovať, a komponent, ktorý sa má načítať. Tento router bohužiaľ neponúka natívnu cestu vytvorenie chránenej cesty, preto existuje viacero spôsobov, ako dosiahnuť túto funkcionalitu. Jednou z možností je vytvoriť si nový komponent. Tento komponent bude obsahovať funkcionalitu, ktorá skontroluje či existuje autorizačný token. Ak tento token existuje, povolí aktivovať cestu, ak token neexistuje, odkáže užívateľa späť na prihlasovací formulár.

```
const ProtectedRoute = ({ component: Component, ...rest }) => (
<Route {...rest} render={(props) => (getToken() ?
<Component {...props} /> : <Redirect to={{ pathname: '/login',
state: { from: props.location }}} />
  )} />
);
<BrowserRouter>
  <Route path="/login" component={Login}/>
  <ProtectedRoute path="/home" component={Home}/>
</BrowserRouter>
```

Pri takomto spôsobe vytvorenia chránených ciest potom už len pre jednotlivé cesty použiť jednotlivé komponenty na základe toho, či majú byť prístupné každému, alebo len prihlásenému užívateľovi.

## 8.3 Autentifikácia

Základ autentifikácie tvorí súbor *Auth.js*, ktorý obsahuje funkcie zodpovedné sa správu prihlásenia, sú to funkcie pre registráciu a prihlásenie. Tieto funkcie vyžiadajú request na API, a spracujú výsledok. Po prihlásení využije store, ktorý má na starosti uloženie tokenu do pamäte prehliadača.

## 8.4 Dashboard

Celý dashboard obsluhuje komponent s názvom *Home*. Tento komponent sa stará o zobrazenie podriadených komponentov ako kariet. V rámci neho sú riešené aj jednotlivé requesty na API, pre všetky karty. Tieto údaje sú následne posielané do komponent pomocou props. Vďaka tomu, že komponenty reagujú na zmenu týchto dát, stačí tieto dáta zmeniť len v nadradenom komponente. Táto zmena vyvolá aj prerenderovanie podriadených komponentov.

Vďaka tomu, ak užívateľ zmení svoj stav, napríklad zaznamená príchod do práce, sa tento stav zmení priamo v nadriadenom komponente, čo spôsobí aktualizáciu jednotlivých kariet. Odpadá nutnosť riešiť tieto aktualizácie pomocou eventov.

```
class Home extends Component {
  state = {
    lastAction: {},
    actions: [],
    userActions: []
  }
  componentDidMount() {
    this.getActions()
    this.getUserActions()
  }
  getUserActions() {
    axios.get(`/users/actions`)
    .then(response => {
      this.setState({
        userActions: response.data.slice(0, 6)
      })
    })
  }
  setAction = (actionTypeID) => {
    axios.post(`/actions`,
    {
      actionTypeID,
      date: new Date(),
      userID: getUserID()
    }).then(response => {
          this.getActions()
          this.getUserActions();
      })
  }
  render() {
    return (
```

```
    <div className="dashboard">
      <div className="widget">
        <DashboardLastActions actions={this.state.actions}/>
      </div>
      <div className="widget">
        <DashboardActions lastAction={this.state.lastAction}
setAction={this.setAction.bind(this)}/>
      </div>
      <div className="widget">
        <DashboardAttendance
userActions={this.state.userActions} />
      </div>
      <div className="widget">
        <DashboardStats actions={this.state.actions}/>
      </div>
    </div>
    );
  }
}

export default Home;
```

V kóde vyššie je zobrazená časť komponentu, ktorý sa stará o domovskú stránku. Predstavuje príklad toho ako vyzerá bežný komponent v React aplikácií. Obsahuje vlastný stav, zadefinované vlastné funkcie a JSX kód, ktorý sa nachádza vo funkcií *render()*, v ktorom sa posielajú dáta do podriadených komponentov pomocou props.

## 8.5 Prítomný a Záznamy

Obe tabuľky sú riešené pomocou Material UI tabuľky. Funkcionalita je veľmi podobná tej vo frameworku Vue, s jedným podstatnejším rozdielom. Jazyk JSX, v ktorom sa píšu šablóny v Reacte, nepodporuje žiadnu špecialnú funkcionalitu postavenú na jazyku HTML, tak ako je to pri Vue a Angular. Vzhľadom na to, že sa jedná o jazyk, ktorý je vo výsledku preložený do JavaScriptu, môže v ňom používať priamo všetky JavaScript funkcie. Pre výpis riadkov tabuľky sa teda nevyužíva žiadna špeciálna direktíva, ale funkcia *map()*, ktorá prechádza všetkými prvkami listu a postupne vráti jednotlivé elementy z neho.

```
<Table>
  <TableHead>
    <TableRow>
      <TableCell>Akcia</TableCell>
      <TableCell>Čas</TableCell>
```

```
      <TableCell>Meno</TableCell>
    </TableRow>
  </TableHead>
  <TableBody>
    {this.state.actions.map(item => (
      <TableRow key={item.id}>
        <TableCell scope="row">
        <div className="action-cell">
          <div
          className={`action-marker ${actionFor-
mat(item.actionTypeID, false)}`}></div>
            {actionFormat(item.actionTypeID)}
        </div>
        </TableCell>
        <TableCell>{new Da-
te(item.date).toLocaleDateString()}</TableCell>
        <TableCell>{item.name}</TableCell>
      </TableRow>
    ))}
  </TableBody>
</Table>
```

V kóde je možnosť vidieť použite funkcie *map()* pre výpis riadkov tabuľky v JSX a použitie funkcií pre formátovanie dátového výstupu.

## 8.6 Kalendár

Pre kalendár je použitá rovnaká knižnica ako je to pri frameworku Vue. Jej implementácia je však o čosi náročnejšia. Každý typ pohľadu je nutné implementovať zvlášť, to isté platí o výberovníku, medzi týmito pohľadmi. Výsledný pohľad je však v podstate totožný.

# 9 POROVNANIE FRAMEWORKOV

Aplikácie boli navrhnuté tak, aby mali rovnaký cieľ. Vďaka tomu je možné názorne vidieť, ako sa rovnaký typ problémov rieši v jednotlivých technológiách. To umožňuje porovnať objektívne vlastnosti, ako je napríklad rýchlosť načítania jednotlivých aplikácií, prípadne ich výsledná veľkosť, ktorú budú zaberať na serveri.

## 9.1 Porovnanie veľkosti produkčných buildov aplikácií

Pre vystavenie produkčnej verzie aplikácie do ostrej prevádzky je potrebné vytvoriť produkčný build aplikácie. Vývoj prebieha vo vývojárskom móde. V tomto móde sú dostupné niektoré funkcionality, ktoré zbytočne zväčšujú výslednú veľkosť aplikácie, prípadne môžu predstavovať bezpečnostne riziká. Jedná napríklad o funkcionalitu tzv. source map. Tieto súbory mapujú výsledný vyrenderovaný JavaScript kód na zdrojové súbory aplikácie. Vďaka tomu je možný debugging aplikácie priamo v prehliadači. Ďalšou funkciou je automatická aktualizácia zobrazenia v prehliadači, po tom ako uložíme zmenu v kóde. Vďaka tomu nie je nutné po každej zmene vykonať manuálnu aktualizáciu stránky po každej zmene.

Produkčný build túto funkcionalitu odstraňuje. Vďaka odstráneniu source máp, je v prehliadači dostupný len vyrenderovaný minifikovaný JavaScript a CSS kód. Hlavnou úlohou je zmenšenie výsledných súborov, a zvýšenie jeho výkonnosti. Majú však svoje opodstatnenie, ak veľkosť aplikácie nie je problém. Stávajú sa situácie, kedy niektorá časť aplikácie nefunguje len na určitom prostredí, napríklad na produkcií. Bez nich je debuggovanie takmer nemožné. Preto sa ukladanie source máp musí vyhodnotiť pre každú situáciu zvlášť.

### 9.1.1 Angular

Pre vytvorenie produkčného buildu aplikácie je nutné pomocou terminálu v zložke projektu spustiť príkaz *ng build --prod*. Tento build okrem vytvorenia produkčného balíčku, skontroluje aplikáciu do väčšieho detailu, ako je to pri vývojovom režime. Existujú chyby v aplikácií, ktoré vývajarským buildom prejdú, produkčným však nie. Jedna sa napríklad o volanie neexistujúcej funkcie, prípadne volanie funkcie s nesprávnym počtom parametrov, zo šablóny komponentu. Dôvodom vynechania tejto hĺbkovej kontroly vo vývojárskom režime, je zrýchlenie jednotlivých buildov. Produkčné buildy trvajú rádovo dlhšie, a veľmi by spomaľovali proces vývoja aplikácie.

Po úspešnom dokončení buildu je vytvorený nový priečinok v zložke projektu, s názvom *dist*. V ňom sa nachádza výsledný projekt pripravený ísť na server, alebo hosting. Ak chceme do produkčného buildu zahrnúť aj source mapy, je nutné túto možnosť povoliť v súbore *angular.json*. Výsledná veľkosť aplikácie vytvorenej v rámci tejto práce bez source máp je 1.2MB. S povolenými source mapami veľkosť stúpne na 7.63MB.

### 9.1.2 Vue

Spustenie produkčného buildu v technológií Vue je vykonané spustením príkazu *npm run build*. Tento príkaz vytvorí priečinok *dist*, ktorého obsah je pripravený na vystavenie na hosting. Vo základnom nastavení však tento build automaticky pridáva do buildu aj source mapy. Preto je potreba ich manuálne vypnúť. Pre to je nutné vytvoriť nový súbor v koreňovom adresári projektu, s názvom *vue.config.js.* V ňom sa dá, okrem iného, vypnúť generovanie týchto máp.

```
module.exports = {
  productionSourceMap: false
};
```

Produkčný balíček má výslednú veľkosť 5.75MB. Po vynechaní generovania source máp veľkosť klesla na 1.21MB.

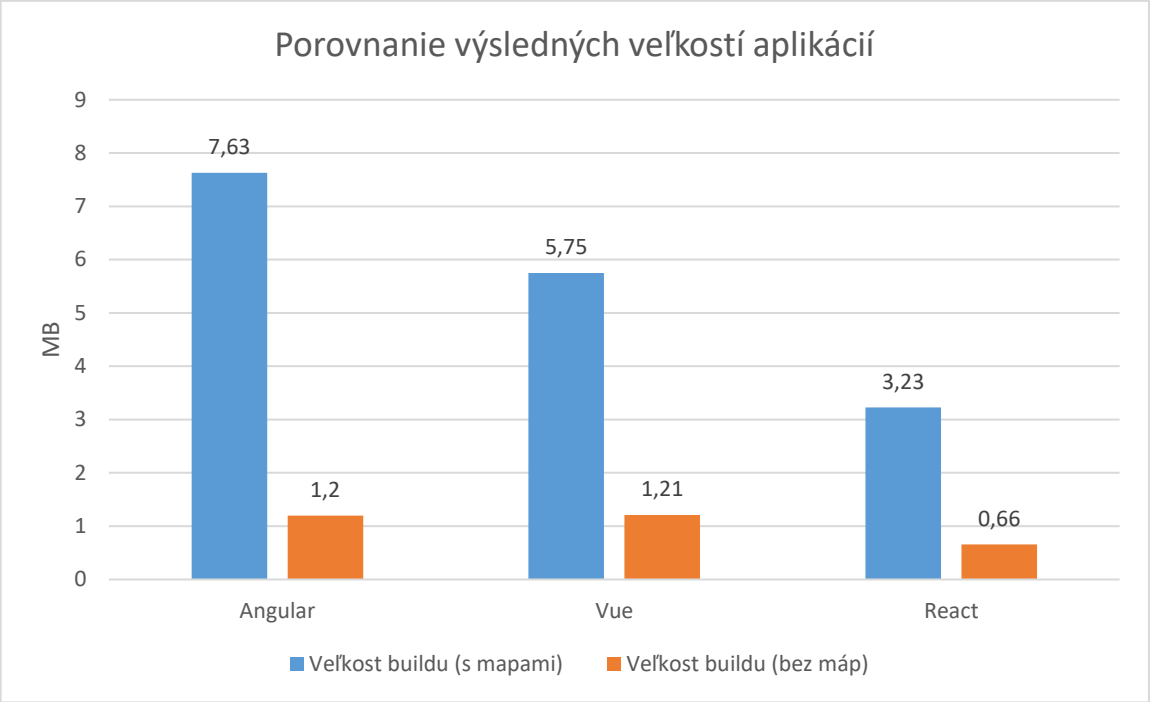
### 9.1.3 React

React má identický proces vytvorenia produkčného balíčku. Rovnako aj platí, že React automaticky vytvára source mapy, a nemá ani oficiálnu cestu ako ich z buildy odstárniť. Najjednoduchšou, a aj developermi odporúčanou možnosť je po builde tieto mapy jednoducho vymazať.

Balíček so source mapami má veľkosť 3.23MB, bez source máp veľkosť klesne na 656KB

### 9.1.4 Porovnanie výsledkov

Pri porovnaní výsledkov vidíme, že Angular v tomto porovnaní vyšiel najhoršie. Pri balíčkoch bez máp, je na tom spolu s Vue rovnako, vytvára však omnoho väčšie source mapy. Najmenšie aplikácia vytvára framework React v oboch prípadoch, s mapami, aj bez nich.




Obrázok 10. Výsledné veľkosti jednotlivých aplikácií zobrazene v grafe

## 9.2 Porovnanie rýchlostí aplikácií

Pre porovnanie rýchlosti použijem nástroje vývojárskej konzoly, ktorú ponúka prehliadač Google Chrome. Záložka Performance umožňuje nahrať priebeh načítania aplikácie, a rozpíše, koľko ktorý krok trval. Všetky testovania prebiehali lokálne na tom istom počítači, a aplikácie boli pripojený na rovnaký lokálny server. Týmito krokmi sa docielilo minimalizovanie externých vplyvov, ktoré by mohli ovplyvniť výsledky.

Vzhľadom na to, že sa jedná o single page aplikácie, načítavajú sa celé hneď pri prvotnom príchode na stránku. Táto funkcionalita sa dá obísť, a znížiť prvotnú veľkosť stránku funkcionalitou s názvom „lazy loading“. Pri použití tejto techniky sa načítavajú jednotlivé stránky aplikácie až potom, ako sa na danú časť užívateľ dostane. Deje sa to však na pozadí, a nedochádza k obnoveniu stránky v prehliadači.

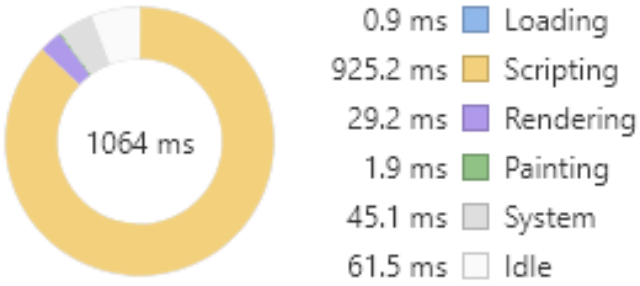
Ideálnym miesto pre zistenie rýchlosti aplikácie je načítanie domovskej stránky aplikácie po obnovení prehliadača. Týmto sa otestuje rýchlosť načítania, vyrenderovania, a rýchlosť základných funkcií, ako napríklad vytvorenie requestu na API a spracovanie týchto dát. Obsahuje najväčšie množstvo podriadených komponentov. Táto stránka takisto neobsahuje takmer žiadne časti z externých knižníc, takže výsledky nebudú ovplyvnené výkonnosťou týchto doplnkov.

Záložka Performace obsahuje možnosť začať nahrávanie výkonu aplikácie a automaticky obnoviť stránku. Je dôležité počas tohto testu vypnúť cachovanie, ktoré by mohlo skresliť výsledky. Treba však dodať že sa jedná len o prvotné načítanie aplikácie, ktoré užívateľ pri svojej návšteve zažije len raz, prípadne pri dodatočnom obnovení stránky.

Ako ďalší test môžeme použiť záložku Audit, ktorá obsahuje nástroj Lighthouse, ktorý vykoná komplexnú analýzu stránky z viacerých uhlov pohľadu, napríklad aj z pohľadu výkonnosti. Tento test je automatizovaný, a teda odstraňuje ľudský zásah, vďaka čomu je presnejší.

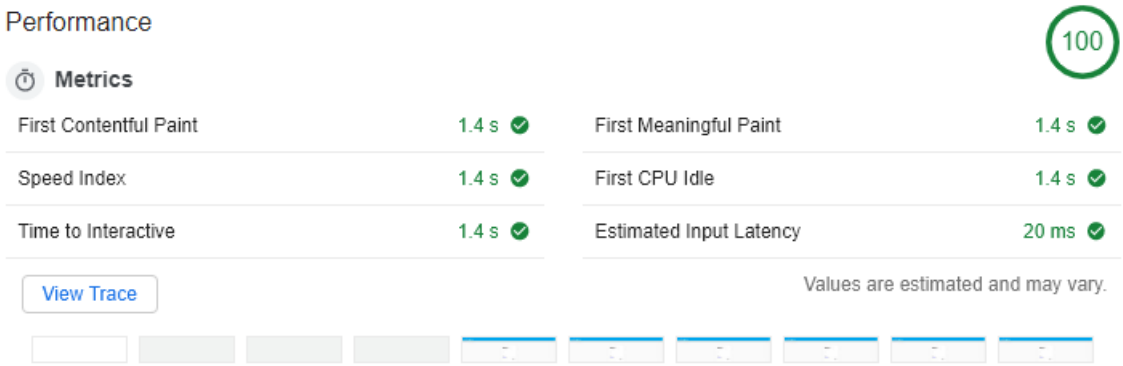
### 9.2.1 Angular

Ako prvá bola otestovaná aplikácie napísaná v Angulare. Pomocou dostupných nástrojov dokážeme rozsah merania dodatočne orezať tak, aby zbytočne neobsahoval čas kedy sa pri meraní nič nedialo. Jedná sa hlavne o čas, medzi dokončením načítania stránky, a manuálnym vypnutím nahrávania výkonu.




Obrázok 11. Výsledky výkonového testu aplikácie v technológií Angular

Kompletné načítanie stránky trvalo takmer presne jednu sekundu. Najdlhší čas zabrala položka Scripting. Táto položka obsahuje hlavne spracovávanie súboru main.js prehliadačom. Tento súbor obsahuje minifikovaný kód celej aplikácie.

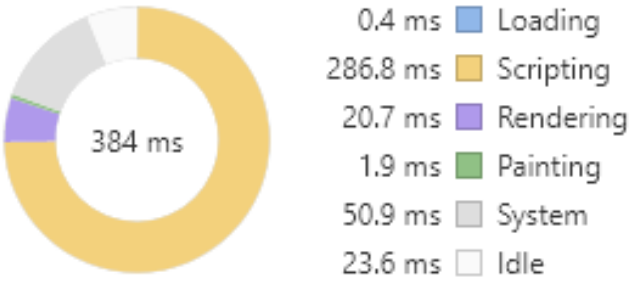



Obrázok 12. Výsledok auditu Angular aplikácie

Podľa nástroja Lighthouse dôjde po obnovení stránky ku vykresleniu a možnosti ovládať aplikáciu po 1,4 sekundách.

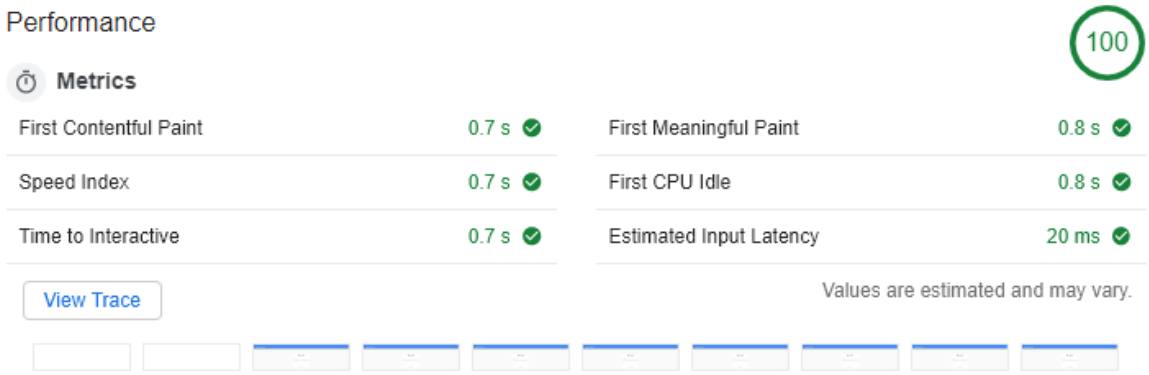
### 9.2.2 Vue

Ako ďalšiu aplikáciu som podrobil testu aplikáciu vo frameworku Vue.




Obrázok 13. Výsledky výkonového testu aplikácie v technológií Vue

Načítanie aplikácie vo frameworku trvalo necelých 400 ms. To je podstate rýchlejšie, ako je to pri predošlom prípade. Scripting v tomto prípade trval 287 ms

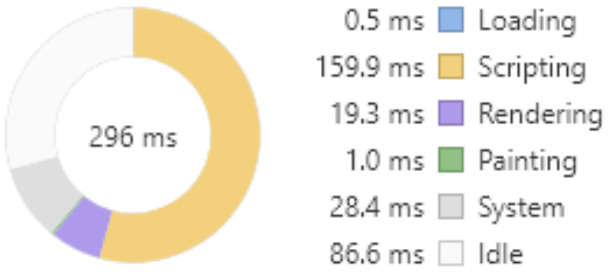



Obrázok 14. Výsledok auditu Vue aplikácie

Predošlé meranie potvrdzuje aj audit, ktorý nameral prvé vykreslenie a použiteľnosť aplikácie po 0,7 sekundách.

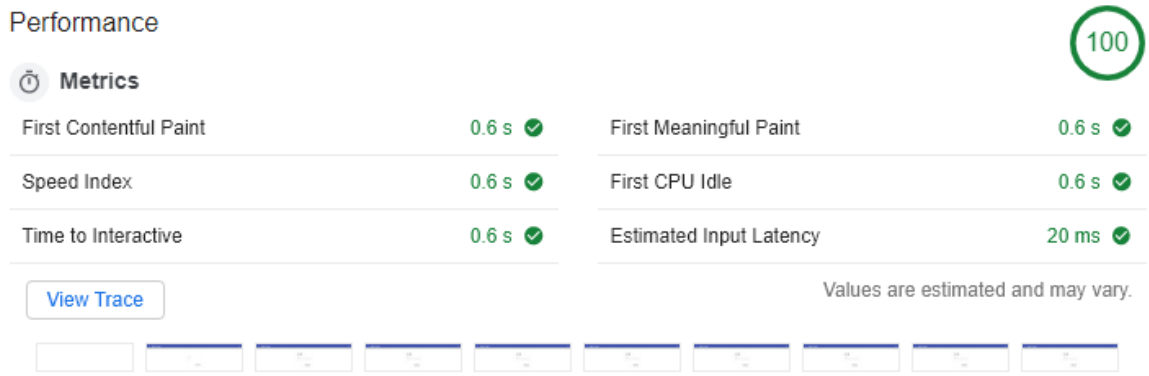
### 9.2.3 React

Poslednou testovanou aplikáciou bola aplikácia v Reacte.




Obrázok 15. Výsledky výkonového testu aplikácie v technológií React

Najrýchlejšie načítanie zvláda technológia React. Scripting trval iba necelých 160 ms, čo je menej ako tretina pri porovnaní s Angularom.




Obrázok 16 Výsledok auditu React aplikácie

Podľa auditu trvá prvotné načítanie a získanie možnosti ovládať aplikáciu 0,6 sekundy. Tento výsledok je konzistentný s predchádzajúcim meraním, kde manuálne meranie pomocou Performance bolo približne o 100 ms rýchlejšie ako to bolo pri Vue aplikácií.

## 9.3 Porovnanie použiteľnosti a náročnosti na vývoj

Frameworky nijakým spôsobom neobmedzujú na použiteľnosti. Hranice toho, čo je možné vytvoriť nám tieto technológie nijak neurčujú, a pri vývoji aplikácií som sa nestretol s obmedzeniami, ktoré som musel prekonať kvôli niektorému z frameworkov.

Čo sa týka objektívnych metrík, ako je napríklad veľkosť a rýchlosť aplikácie, vyšiel z nich najúspešnejšie framework React. Konzistentne vyhral vo všetkých testoch, ktoré boli v rámci tejto práce vykonané.

Náročnosť vývoja je z veľkej časti subjektívna vec. Veľmi záleží na preferenciách a predchádzajúcich skúsenostiach vývojára. Za technológiu najľahšiu na naučenie rozhodne považujem framework Vue. Vývoj v rámci tejto technológie je intuitívny, používa zaužívané technológie, s ktorými je veľa vývojárov už oboznámených. Tento framework sa dá považovať za prienik Reactu a Angularu, z oboch technológií si zapožičiava určitú funkčnosť, pri zachovaní malej celkovej veľkosti a vysokej rýchlosti. Je vhodný na nasadenie do existujúcich projektov, kde potrebujeme pridať určitý level interakcie a na vytvorenie menších single page aplikácií.

Framework Angular predstavuje najkompletnejšie a najrobustnejšie riešenie. Hneď po inštalácií je pripravený pre vytvorenie aplikácie bez nutnosti doinštalovať ďalšie pluginy a knižnice. Je takisto najstriktnejší v dodržovaní architektúry, a vývojár nemá také veľké množstvo možností ako riešiť určitý typ problému, tak ako je to hlavne pri Reacte. Tento systém má svoje výhody a nevýhody. Hlavnou nevýhodou je, že riešenie, ktoré uprednostňuje Angular, nemusí byť vždy najoptimálnejšie. Je to zrejmé aj z testov vykonaných v tejto práci. Význam nevýhody dlhšieho počiatočného načítania sa však s veľkosťou projektu zmenšuje. Veľkou výhodou je však to, že toto riešenie je konzistentné, a to aj naprieč projektami. Ak teda príde vývojár do zabehnutého Angular projektu, tento projekt si dokáže rýchlo osvojiť. Preto, ako najvhodnejší typ projektu pre túto technológiu vidím veľké a zložité projekty, na ktorých pracujú veľké tímy vývojárov s rôznymi skúsenosťami.

React poskytuje vývojárovi voľnú ruku pri riešení všetkých problémov. Framework je modulárny, jednotlivé projekty sa dajú poskladať a navrhnúť veľmi efektívne, tak aby boli čo

najvýkonnejšie. Táto voľnosť a modulárnosť však znamená, že je technológia veľmi závislá na počiatočnom návrhu a vybratí správnych modulov aplikácie, pre začiatočníka je pomerne náročná na naučenie, a teoreticky každý projekt napísaný v Reacte môže vyzerať diametrálne odlišne. Toto môže spôsobiť problémy pri prípadnom začleňovaní nových členov do vývojového tímu. React je vhodný ako technológia pre projekty, na ktorom pracujú skúsení vývojári, a primárnym cieľom je dosiahnuť čo najväčší výkon a efektivitu.

# 10 VÝUKOVÉ MATERIÁLY PRE ŠTUDENTOV

V rámci praktickej časti som vypracoval prezentácie, ktoré môžu byť použité na pomoc pri výuke webových technológií a tvorbe single page aplikácií. Prezentácie sa postupne venujú vysvetleniu základných pojmov a dôvodu použitia single page aplikácií, ich silnými a slabými stránkami. V stručnosti sa prezentácie venujú technológiám a nástrojom potrebným k vytváraniu SPA.

V ďalších častiach sú jednotlivo popísané všetky tri frameworky, požiadavky na vývojové prostredie, ich základný popis, charakteristické vlastnosti a funkcie, a použitie týchto funkcií. Sú postavené tak, aby podľa nich študent dokázal každý framework nainštalovať na lokálne vývojové prostredie, a vyskúšal si prácu s jednotlivými funkciami frameworkov.

Prezentácie sú priložené ako digitálna kópia tejto práce.

# ZÁVĚR

V rámci teoretickej práce boli popísané single page aplikácie a technológie, ktoré sú potrebné pre tvorbu tohto typu aplikácií. V ďalších kapitolách sú podrobnejšie popísané momentálne tri najpopulárnejšie frameworky pre tvorbu single page aplikácií, a to Angular, React a Vue. Sú popísané ich charakteristické vlastnosti a funkcionalita.

V rámci praktickej časti tejto diplomovej práce boli vytvorené tri single page aplikácie vytvorené v troch rôznych frameworkoch, ktoré boli pomocou REST API napojené na serverovú časť, obsluhujúcu spojenie s databázou. Je podrobne popísaná architektúra jednotlivých aplikácií. Aplikácie sú napísané vo frameworkoch Angular, React a Vue, a serverová aplikácia je napísaná v technológií .NET Core. Aplikácie simulujú dochádzkový systém pre firmu, ktorá umožňuje spravovať dochádzku jednotlivých zamestnancov. Do systému je možné sa zaregistrovať a prihlásiť. Užívatelia majú po prihlásení možnosť spravovať jednotlivé akcie, ako napríklad príchod, alebo odchod z práce. V aplikáciách sú poskytnuté informácie a štatistiky o jednotlivých akciách pomocou rôznych zobrazení. Aplikácie sú navrhnuté tak, aby boli modulárne, a jednoducho škálovateľné. Aplikácie majú rovnaký grafický návrh aj funkčnosť, preto je na nich vidno, ako sa jednotlivé technológie vysporiadavajú s tými istými sadami problémov.

Práca sa venuje porovnaniu jednotlivých technológií na základe vytvorených aplikácií. Aplikácie sú porovnané objektívnymi parametrami ako je napríklad výsledná veľkosť a rýchlosť aplikácie. V tomto objektívnom porovnaní jednoznačne vyhral framework React, vyvažuje to však o čosi náročnejším vývojom. Tento framework je aj najotvorenejší a najmodulárnejší, a necháva vývojárom voľnú ruku pri riešení problémov.

Angular vyšiel z merania ako najpomalší, svoje výhody však má v pomernej jednoduchosti vývoja, funkčnosti bez nutnosti doinštalovať ďalšie moduly, a jednoznačne určenej architektúre, ktorá uľahčuje spravovať veľké projekty.

Framework Vue je ako prienik predošlých dvoch technológií , z ktorých si vyberá jednotlivé funkcie. Platí pri ňom veľká možnosť modularity, tak ako je to pri Reacte, používa však jasnejšie určenú architektúru, a tradičné technológie, s ktorými sa veľa vývojárov stretlo už v minulosti. To z neho robí najintuitívnejšiu technológiu, najľahšiu na naučenie.

Súčasťou práce sú prezentácie popisujúce jednotlivé technológie a ich vlastnosti. Prezentácie sú navrhnuté tak, aby podľa nich dokázal študent framework nainštalovať na lokálne

vývojové prostredie, a dokázal vytvoriť základnú aplikáciu, na ktorej sa dá ďalej stavať. Boli vytvorené štyri prezentácie, jedna sa venuje single page aplikáciám celkovo, ich princípu, výhodám a nevýhodám. Ďalšie tri prezentácie sa venujú jednotlivých frameworkom. Každý framework je popísaný vo vlastnej prezentácií.

# SEZNAM POUŽITÉ LITERATURY

[1] SCOTT, Emmit A. SPA design and architecture: understanding single-page web applications. Shelter Island, NY: Manning, 2016. ISBN 978-1617292439.

[2] ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET [online]. [cit. 2019-04-30]. Dostupné z: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[3] BROWN, Tiffany B., Kerry BUTTERS a Sandeep PANDA. HTML5 okamžitě: [ovládněte HTML5 za víkend]. Brno: Computer Press, 2014. ISBN 9788025142967

[4] HTML 5.2 [online]. [cit. 2019-05-02]. Dostupné z: https://www.w3.org/TR/2017/REC-html52-20171214/

[5] GASSTON, Peter. CSS3. Přeložil Ondřej BAŠE. Brno: Computer Press, 2016. ISBN 9788025146415.

[6] CSS preprocessor [online]. [cit. 2019-05-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor

[7] Javascript [online]. [cit. 2019-05-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[8] The 10 most popular programming languages, according to the 'Facebook for programmers' [online]. [cit. 2019-05-02]. Dostupné z: https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10#1-javascript-10

[9] Architecture overview [online]. [cit. 2019-04-30]. Dostupné z: https://angular.io/guide/architecture

[10] JANSEN Remo. Learning TypeScript. Birmingham, Spojené království: Packt Publishing, 2015. ISBN 1783985550.

[11] FAIN, Yakov a Anton MOISEEV. Angular 2 development with TypeScript. Shelter Island, NY: Manning Publications Co., 2017. ISBN 9781617293122.

[12] Introduction to modules [online]. [cit. 2019-04-30]. Dostupné z: https://angular.io/guide/architecture-modules

[13] FREEMAN, Adam. Pro Angular 6. New York, NY: Springer Science Business Media, 2018. ISBN 978-1484236482.

[14] Introduction to components [online]. [cit. 2019-04-30]. Dostupné z: https://angular.io/guide/architecture-components

[15] Lifecycle sequence [online]. [cit. 2019-05-02]. Dostupné z: https://angular.io/guide/lifecycle-hooks

[16] A quick intro to Dependency Injection: what it is, and when to use it [online]. [cit. 2019-04-30]. Dostupné z: https://medium.freecodecamp.org/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f

[17] Routing & Navigation [online]. [cit. 2019-04-30]. Dostupné z: https://angular.io/guide/router

[18] Binding syntax: An overview [online]. [cit. 2019-04-30]. Dostupné z: https://angular.io/guide/template-syntax

[19] Angular NgFor: Everything you need to know [online]. [cit. 2019-05-02]. Dostupné z: https://malcoded.com/posts/angular-ngfor/

[20] DatePipe [online]. [cit. 2019-05-02]. Dostupné z: https://angular.io/api/common/DatePipe

[21] Angular 4 - Pipes [online]. [cit. 2019-05-02]. Dostupné z: https://www.tutorialspoint.com/angular4/angular4_pipes.htm

[22] Hello World, <angular/> is here [online]. [cit. 2019-05-02]. Dostupné z: http://misko.hevery.com/2009/09/28/hello-world-angular-is-here/

[23] A sneak peek at the radically new Angular 2.0 [online]. [cit. 2019-05-02]. Dostupné z: https://jaxenter.com/angular-2-0-112094.html

[24] Angular 2.0 announcement backfires [online]. [cit. 2019-05-02]. Dostupné z: https://jaxenter.com/angular-2-0-announcement-backfires-112127.html

[25] Angular - One framework. Mobile & desktop. [online]. [cit. 2019-05-02]. Dostupné z: https://github.com/angular/angular

[26] WIERUCH, Robin. The Road to learn React: Your journey to master plain yet pragmatic React.js. CreateSpace Independent Publishing Platform, 2018. ISBN 978-1986338820.

[27] Tutorial: Intro to React [online]. [cit. 2019-05-02]. Dostupné z: https://reactjs.org/tutorial/tutorial.html#what-is-react

[28] Components and Props [online]. [cit. 2019-05-02]. Dostupné z: https://reactjs.org/docs/components-and-props.html

[29] React.Component [online]. [cit. 2019-05-02]. Dostupné z: https://reactjs.org/docs/react-component.html

[30] React Lifecycle Methods – A Deep Dive [online]. [cit. 2019-05-02]. Dostupné z: https://programmingwithmosh.com/javascript/react-lifecycle-methods/

[31] Draft: JSX Specification [online]. [cit. 2019-05-02]. Dostupné z: https://facebook.github.io/jsx/

[32] Fedosejev, Artemij. React.js essentials: a fast-paced guide to designing and building scalable and maintainable web apps with React.js. First published. Birmingham: Packt Publishing, 2015. x, 183 stran. Open source community experience distilled. ISBN 978-1-78355-162-0.

[33] A Beginners Guide to Redux [online]. [cit. 2019-05-02]. Dostupné z: https://www.gistia.com/beginners-guide-redux/

[34] The History of React.js on a Timeline [online]. [cit. 2019-05-02]. Dostupné z: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

[35] React: A declarative, efficient, and flexible JavaScript library for building user interfaces [online]. [cit. 2019-05-02]. Dostupné z: https://github.com/facebook/react

[36] CHAU, Guillaume. Vue.js 2 Web Development Projects: Learn Vue.js by building 6 web apps. Packt Publishing, 2017. ISBN 978-1787127463.

[37] Components Basics [online]. [cit. 2019-05-09]. Dostupné z: https://vuejs.org/v2/guide/components.html

[38] Understanding Vue.js Lifecycle Hooks [online]. [cit. 2019-05-09]. Dostupné z: https://alligator.io/vuejs/component-lifecycle/

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| API | Aplication Programming Interface. |
| CLI | Command Line Interface |
| CSS | Cascading Style Sheets |
| DOM | Document Object Model |
| HTML | Hyper Text Markup Language |
| HTTP | Hyper Text Transfer Protokol |
| JS | JavaScript |
| JSON | JavaScript Object Notation |
| JSX | JavaScript Extension |
| MB | Megabyte |
| MVC | Model-View-Controller |
| MVVM | Model-View-VieModel |
| REST | Representational State Transfer |
| SASS | Syntactically Awesome Style Sheets |
| SPA | Single Page Application |
| XML | eXtensible Markup Language |

# SEZNAM OBRÁZKŮ