TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

JavaScriptové frameworky pro vývoj webových aplikací

Diplomová práce

Studijní program:	N6209 Systémové inženýrství a informatika
Studijní obor:	Manažerská informatika
Autor práce:	**Bc. Antonín Doležal**
Vedoucí práce:	Mgr. Tomáš Žižka, Ph.D. Katedra informatiky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

Zadání diplomové práce

JavaScriptové frameworky pro vývoj webových aplikací

Jméno a příjmení: **Bc. Antonín Doležal**
Osobní číslo: E17000265
Studijní program: N6209 Systémové inženýrství a informatika
Studijní obor: Manažerská informatika
Zadávající katedra: Katedra informatiky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Analýza a porovnání moderních javascriptových frameworků
2. Základní syntaxe jednotlivých frameworků
3. Charakteristika a implementace vybraného javascriptového frameworku
4. Implementace frameworku na vybraném projektu
5. Zhodnocení navržených řešení

Rozsah grafických prací:
Rozsah pracovní zprávy: 65 normostran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

- AMBLER, Tim a Nicholas CLOUD. 2015. *JavaScript frameworks for modern web dev.* New York: Apress, Expert's voice in Web development. ISBN 978-1-4842-0663-8.
- BLOKDYK, Gerardus. 2018. *Comparison of Javascript Frameworks.* California, USA: CreateSpace Independent Publishing Platform, ISBN 978-1963828225.
- FREEMAN, Adam.2018. *Pro Vue.JS 2.* New York, NY: Springer Science Business Media, ISBN 978-1-4842-3804-2.
- ŽÁRA, Ondřej. 2015. *JavaScript: programátorské techniky a webové technologie.* Brno: Computer Press, ISBN 978-80-251-4573-9.
- PROQUEST. 2019 *Databáze článků ProQuest* [online]. Ann Arbor, MI, USA: ProQuest. [cit. 2019-09-26]. Dostupné z: http://knihovna.tul.cz

Konzultant: Mgr. Dita Plchová

Vedoucí práce: Mgr. Tomáš Žižka, Ph.D.
Katedra informatiky

Datum zadání práce: 31. října 2019
Předpokládaný termín odevzdání: 31. srpna 2021

L.S.

prof. Ing. Miroslav Žižka, Ph.D.
děkan

doc. Ing. Klára Antlová, Ph.D.
vedoucí katedry

V Liberci dne 31. října 2019

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

30. července 2020 Bc. Antonín Doležal

Anotace

Diplomová práce se zabývá JavaScriptovými frameworky pro vývoj moderních webových aplikací. Úkolem práce je nastínit možný postup implementace vybraného JavaScriptového frameworku na projektu z reálného prostředí. Dalším úkolem je pomoc při rozhodování mezi konkrétními JavaScriptovými frameworky. V první části práce jsou vysvětleny základní pojmy vztahující se ke zmíněné problematice. Je provedena analýza a porovnání zvolených frameworků, které v současné době dominují jak ve své popularitě, tak v počtu svých uživatelů. V další části práce jsou ukázány odlišnosti v základní syntaxi mezi danými frameworky na vybraných příkladech a postup vytvoření nového projektu pro každý zvolený framework. Poté se práce věnuje představení vybrané webové aplikace, definuje důvody a cíle přechodu na novější technologie, popisuje výběr nejvhodnějšího frameworku, a nakonec ukazuje samotný postup implementace.

Klíčová slova

JavaScriptové frameworky, JavaScriptové knihovny, webové aplikace, Angular, React, Vue, jednostránkové webové aplikace, metody životního cyklu komponenty, frontend

Annotation

JavaScript Frameworks for Web Application Development

This thesis focuses on JavaScript frameworks with the purpose of developing modern web-based applications. The main objective is to outline a possible procedure for the implementation of the selected JavaScript framework on a project from a real environment. Secondary objective is to provide aid in the framework selection process. The first part of the thesis explains basic concepts related to the thesis' subject. A comparative analysis of the selected frameworks, which all currently dominate the market in both popularity and the number of users, is performed. In the next part of the thesis, differences in the basic syntax between the frameworks on selected examples are shown as well as the process of creating a new project for each framework. The thesis then focuses on the introduction of the chosen web application, defines the reasons and goals behind the transition to the newer technologies, describes the selection of the most suitable framework, and finally demonstrates the implementation process itself.

Key Words

JavaScript frameworks, JavaScript libraries, web applications, Angular, React, Vue, single page applications, component lifecycle methods, frontend

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CLI	Command Line Interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
JS	JavaScript
JSX	JavaScript XML
MVC	Model View Controller
MVVM	Model View ViewModel
NPM	Node Package Manager
SASS	Syntactically Awesome Style Sheets
SPA	Single Page Application
UI	User Interface
URL	Uniform Resource Language
XML	Extensible Markup Language

Úvod

Diplomová práce se zabývá JavaScriptovými frameworky pro vývoj moderních webových aplikací. JavaScriptové frameworky se v posledních letech značně rozrostly, a to jak z hlediska využívanosti, tak z hlediska jejich množství, a staly se určitým standardem při vývoji moderních webových aplikací.

Hlavním cílem diplomové práce je nastínit možný postup použití konkrétního JavaScriptového frameworku na reálném projektu. Sekundárním cílem je pomoc při rozhodování mezi vybranými JavaScriptovými frameworky. Dalšími dílčími cíli práce jsou vymezení základní terminologie spojené s problematikou, charakterizace daných JavaScriptových frameworků, jejich analýza, porovnání a ukázková implementace včetně příkladů možné syntaxe.

Diplomová práce je rozdělena do dvou částí. První, teoretická část práce, je věnována uvedení do problematiky vývoje moderních webových aplikací, vysvětlení základních pojmů a terminologie, jako například rozdíl mezi JavaScriptovými frameworky a knihovnami nebo princip jednostránkových webových aplikací.

Dále jsou v práci popsány tři konkrétní JavaScriptové frameworky, které v současnosti dominují jak ve své popularitě, tak v počtu svých uživatelů. Tyto frameworky jsou poté porovnány podle různých kritérií, jako například složitost učení či budoucí vývoj. Dále v práci následuje obsáhlá kapitola, kde jsou ukázány možné varianty základní syntaxe, které se snaží řídit osvědčenými způsoby a konvencemi daného JS frameworku.

Druhá část diplomové práce je věnována použití vybraného JavaScriptového frameworku na konkrétním projektu. Nejdříve je představen samotný projekt, jeho funkcionalita, uživatelské rozhraní, využívané technologie. Je zde také popsán výběr nejvhodnějšího frameworku pro daný projekt. Dále je v práci demonstrováno použití vybraného JavaScriptového frameworku a možný postup při vývoji s několika ukázkami. Na závěr je uvedeno zhodnocení přechodu na novější technologie.

1 Základní terminologie

K tomu, aby mohla být provedena určitá analýza a porovnání JavaScriptových frameworků, je nejprve nutné specifikovat základní pojmy vztahující se k této problematice. V této kapitole jsou tedy představeny některé základní technologie a pojmy tykající se JavaScriptových frameworků a tvorby moderních webových aplikací.

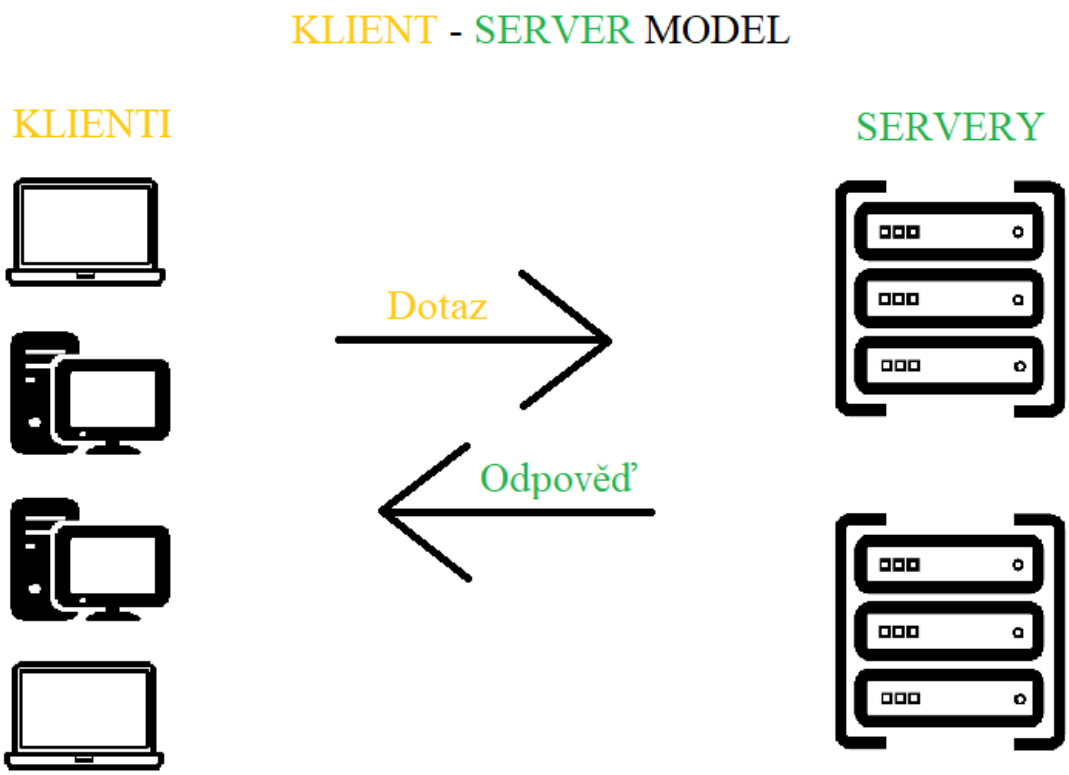
1.1 Webové aplikace

Webová aplikace je software nebo program, který je přístupný pomocí webového prohlížeče. To je obrovskou výhodou oproti desktopovým aplikacím, protože nejen, že tyto aplikace není potřeba stahovat a instalovat, lze je také použít na jakékoliv platformě, která podporuje moderní webové prohlížeče (Guru99, 2020).

Webové aplikace fungují na klient – server modelu. Klient může být stroj nebo program. Klientským strojem se rozumí zařízení, pomocí kterého může uživatel přistupovat k webu. Může to být například laptop, stolní počítač, chytrý telefon, tablet atd. Klientským programem je program, který uživateli umožňuje vytvářet dotazy skrze web, například webový prohlížeč. Klient, ať už je to stroj nebo program, tedy představuje prostředek a způsob, jak učinit dotaz skrze web.

Server je program, který funguje na vysoce výkonných počítačích či jiných zařízeních. Servery poskytují funkcionalitu a slouží (serve) ostatním programům, které se označují jako klienti. Většina serverů běží nepřetržitě a jeden server může obsluhovat tisíce klientů najednou. Server je tedy program, který nepřetržitě poslouchá dotazy klientů a jakmile nějaký dotaz obdrží, tak pošle odpověď.

Když tedy navážeme na předešlé odstavce, můžeme říci, že klient – server model je centralizovaná webová architektura, která rozděluje počítače do dvou sekcí, tazatele (klienty) a poskytovatele odpovědí (servery), kteří mezi sebou sdílejí a vyměňují data, viz obrázek 1 Klient - Server Model (Christensson, 2016).



Obrázek 1: Klient - Server Model
Zdroj: vlastní zpracování s využitím Online Web Fonts, 2018

Programovací jazyky pro vývoj webových aplikací se dělí na klientské a serverové podle toho, kde je daný jazyk vykonáván. Typickými klientskými jazyky jsou HTML, CSS a JavaScript. Tyto jazyky se starají o prezenční vrstvu webové aplikace neboli frontend. HTML a CSS nejsou kompilovány, protože to jsou značkovací jazyky. Serverové jazyky se starají o vrstvu pracující s daty neboli backend. Jsou to například C#, Java, PHP, JavaScript, Python a mnoho dalších (Singhal, 2019).

1.2 Frontend a backend webových aplikací

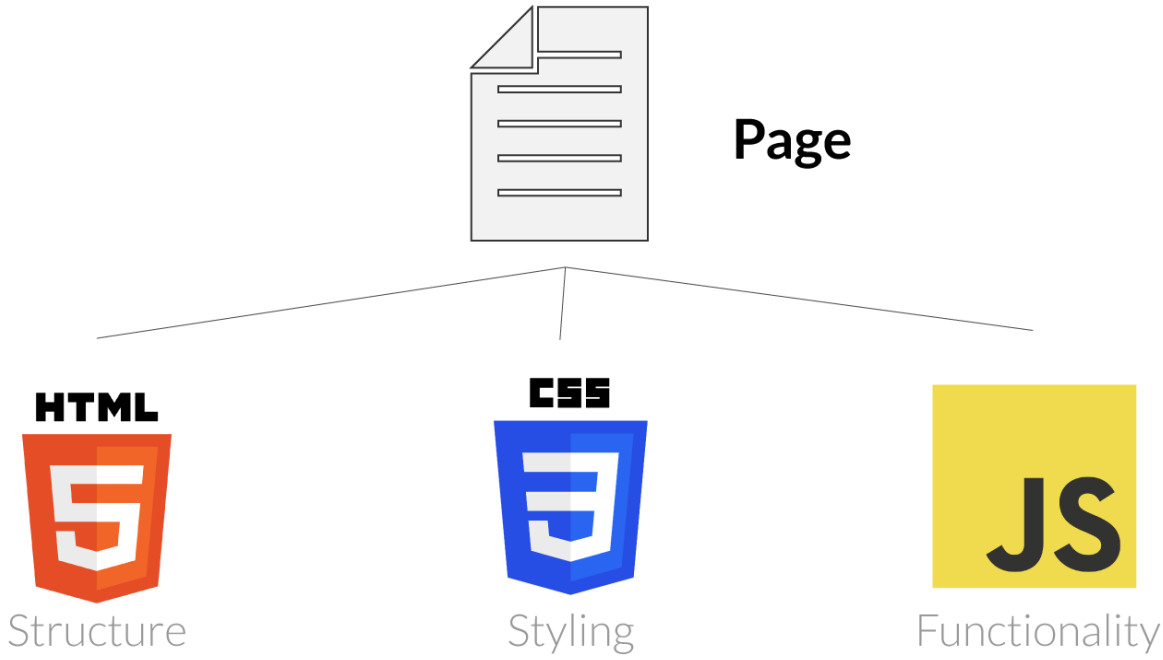
Frontend, z anglického front – přední a end – konec, je prezenční vrstva aplikace, tedy ta část aplikace, kterou může uživatel přímo vidět a interagovat s ní, například grafické uživatelské rozhraní. Frontend zahrnuje vše, co uživatelé přímo vidí: barvy, styl textu, obrázky, tlačítka, navigační menu a mnoho dalšího. Všechny tyto prvky webové stránky jsou tvořeny pomocí HTML, CSS a JavaScriptu, a proto se označují jako frontend jazyky.

Backend, z anglického back – zadní a end – konec, je vrstva pracující s daty, tedy ta část aplikace, která není pro uživatele viditelná. Backend pracuje na serveru, stará se o logiku, ukládá data a zajišťuje vše k tomu, aby aplikace na straně klienta fungovala správně. Uživatel k této vrstvě nepřímo přistupuje pomocí frontendu, který slouží jako prostředník. Typickými programovacími jazyky pro vývoj backendu jsou serverové jazyky viz podkapitola 1.1 Webové aplikace.

Jelikož může být JavaScript použit jak na vývoj frontendu, tak backendu, dělí se podle této logiky i JavaScriptové frameworky, tedy na frontendové a backendové. Frontendovými frameworky a knihovnami jsou například Angular, React, jQuery, Vue, SASS a mnoho dalších. Backendovými frameworky jsou například Express, Django, Meteor, Next a další. Diplomová práce je zaměřena na frontendové frameworky, především pro vývoj jednostránkových aplikací (viz 1.12 SPA) (Singhal, 2019).

1.3 Vývoj frontend technologií

Pro vývoj frontend části webových aplikací je zpravidla využíváno tří technologií, HTML, CSS a JavaScript. Tyto technologie spolu dohromady vytváří základní balík pro vytváření dynamických webových stránek a aplikací viz obrázek 2.



Obrázek 2: Základní frontend jazyky
Zdroj: Kononenko, 2018

Hypertext Markup Language, zkráceně HTML, je značkovací jazyk, který vytváří kostru dané webové stránky neboli strukturuje webový obsah jako například definování odstavců, nadpisů a tabulek či vkládání obrázků a videí.

Cascading Style Sheets, zkráceně CSS, je jazyk, který se stará o design, určuje tedy jak budou jednotlivé HTML prvky zobrazeny, jako například jakou barvou bude pozadí stránky, jak velké bude písmo nebo zda bude obsah rozdělen do více sloupců či nikoliv.

JavaScript je skriptovací jazyk, který zajišťuje funkcionalitu, určuje dynamické a interaktivní prvky dané webové stránky a říká, co se stane, když na tyto prvky uživatel například klikne nebo po nich přejede myší (Kononenko, 2018).

1.3.1 **Historie**

HTML vytvořil Tim Berners-Lee na konci roku 1991 (oficiálně vydáno až v roce 1995 jako HTML verze 2.0) podporovalo pouze text a skládalo se z pouhých 18 značek. O dva roky později bylo vydáno HTML verze 1.0, které umožňovalo vytváření strukturovaných webových stránek, ale z hlediska designu šlo dělat jen velmi málo. To vedlo ke vzniku Kaskádových stylů, zkratka CSS, které navrhl Håkon Wium Lie v roce 1994 (Wanyokie, 2018).

První polovina 90. let byla pro internet velmi důležitým okamžikem. Klíčoví hráči, Netscape a Microsoft, byli uprostřed války prohlížečů, Netscape Navigator a Microsoft Internet Explorer soupeřili o dominanci. Netscape se proto spojuje se společností Sun, která stojí za programovacím jazykem Java. V roce 1995 vytvořil Brendan Eich, vývojář Netscape, za pouhých 10 dní nový skriptovací jazyk s názvem Mocha. Ten byl následně přejmenován na LiveScript a o několik měsíců později byl přejmenován znovu na JavaScript, jak je znám dodnes (DeGroat, 2019 a Aston, 2015).

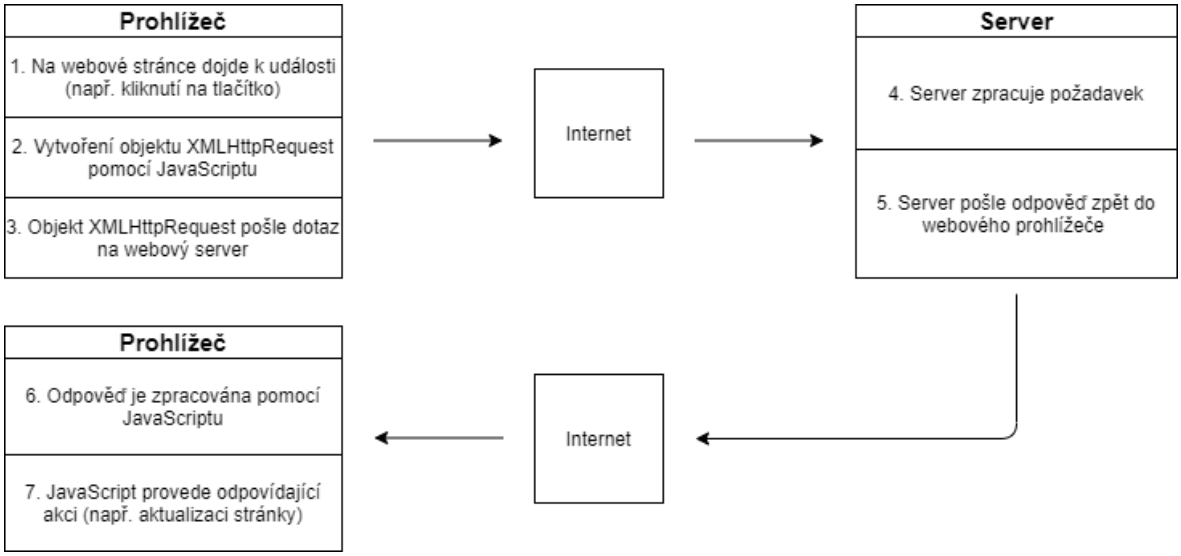
JavaScript neměl nejlepší začátek, neměl dobré výsledky a byl považován spíše za „lepidlo uživatelského rozhraní“, které používají designéři a jiní neinženýři. Realita je však taková, že tento jazyk umožnil internetu velký rozvoj. Programátoři mohli lépe reagovat na události

a vytvářet interaktivní webové stránky. Díky tomu se JavaScript rychle rozšířil a stal se nedílnou součástí moderních webů (DeGroat, 2019).

Okolo roku 2000 byl JavaScript převážně v plenkách a většinou se používal k provádění základních změn na stránce. Mezi hlavní body patřily přechody myší, rozbalovací nabídky a posouvající se text (dnes považováno za samozřejmost s CSS). V té době dominoval webovým prohlížečům Internet Explorer, který obsahoval wrapper (obal) kolem knihovny, kterou Microsoft vyvinul pro svůj vlastní e-mailový produkt, Outlook. Tento objekt XMLHttpRequest se nakonec stal standardizovaným napříč prohlížeči a byl bránou k tomu, jak dnes přistupujeme ke většině webů (Barker, 2018).

Je velmi důležité, že tento XMLHttpRequest umožnil odeslání žádosti na server a zpracování odpovědi bez toho, aniž by bylo nutné aktualizovat okno prohlížeče. Vývojáři v této funkcionalitě viděli potenciál vytvářet skutečné webové aplikace, které mohou nahradit desktopové. Na základě této technologie vznikl tedy nový přístup k vyměňovaní dat mezi klientem a serverem, který byl pojmenován jako AJAX, tedy Asynchronous JavaScript and XML (Barker, 2018).

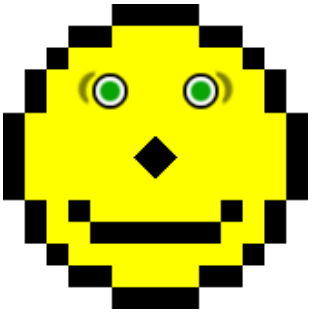
AJAX je způsob, kterým je možné provádět asynchronní aktualizaci webových stránek pomocí vyměňování dat se serverem na pozadí viz obrázek 3.



Obrázek 3: Příklad AJAX komunikace
Zdroj: vlastní podle w3schools, 2020

Během prvních let většina prohlížečů nesplňovala standarty specifikace CSS. Tím pádem doházelo k problémům s kompatibilitou mezi prohlížeči. Web, který vypadal dobře v jednom prohlížeči, vypadal hrozně v jiném. Podpora více prohlížečů byla tedy noční můrou. Bylo vytvořeno mnoho různých CSS hacků, které opravovaly nejrůznější problémy s konkrétními prohlížeči (Wanyoike, 2018).

Skupina známá jako Web Standards Project vytvořila řadu CSS testů nazvaných ACID testy. Když je CSS v prohlížeči implementováno správně, měl by se zobrazit následující obrázek 4:



Obrázek 4: Splnění ACID testů
Zdroj: Wanyoike, 2018

Nicméně všechny hlavní prohlížeče tyto testy nějakým způsobem nesplnily. Díky úsilí Web Standards Project a online komunitám je nyní většina CSS problémů v hlavních prohlížečích vyřešena.

JavaScript, stejně jako CSS, musel projít těžkou cestou spojenou s kompatibilitou webových prohlížečů, kde bylo nutné ověřovat, zda daný JavaScriptový kód proběhne v pořádku ve všech prohlížečích. Jedna z prvních knihoven, která pomohla JavaScriptu s tímto problémem, je jQuery. Tato knihovna byla vydána v roce 2006 a rychle se stala velice populární. jQuery pomohlo vyřešit nejen mnoho problémů, které vznikaly různými implementacemi mezi prohlížeči, ale také obsahovalo spoustu užitečných funkcí, díky kterým bylo možné učinit jakýkoli web interaktivním (Wanyoike, 2018).

S potenciálem vytvářet webové aplikace, vývojáři vytvářeli větší a větší řešení, která testovali udržovatelnost jQuery. Produkty jako Gmail ukázaly, čeho je možné dosáhnout, ale

bylo zřejmé, že je zapotřebí lepších nástrojů, které by byly pro firmy vhodnější. To vedlo ke vzniku prvních JavaScriptových frameworků (Barker, 2018).

V roce 2010 byl vydán Backbone, první JavaScriptový framework, který se zaměřil na vytváření jednostránkových aplikací (SPA). Přibližně ve stejnou dobu byl vytvořen i AngularJS, který se následně dostal pod správu společností Google. AngularJS byl prvním frameworkem, který poskytoval kompletní strukturu pro vývoj prezenční vrstvy (frontend) aplikace. A za nimi následovalo mnoho dalších frameworků jako například Knockout, Ember, Meteor, React, Vue (Barker 2018).

S rostoucím množstvím těchto frameworků a knihoven vznikla potřeba pro správu závislostí. Závislostí se rozumí kód třetí strany, jako například zmíněné JavaScriptové knihovny, bez kterých celá aplikace nebo některé její časti nebudou fungovat. Pokud je v aplikaci pro funkci odesílání emailů použita například knihovna EmailJS, odesílání emailů bez této knihovny nebude fungovat. Vývojáři využívají závislostí, aby si ulehčili práci a nemuseli tak psát pro všechny problémy vlastní řešení.

Pod společností Twitter v roce 2012 vznikl Bower, správce balíčků pro frontendové závislosti. Bower byl používán k načtení a stažení všech závislostí, které byli umístěny na webu. V roce 2014 byl vytvořen centrální repositář pro frontendové balíčky s názvem npm registry (viz podkapitola 1.11 NPM a Node.js), který v současné době hostuje přes jeden milion balíčků (Wanyoike, 2018).

1.4 JavaScriptové knihovny

Obecně řečeno, JavaScriptové knihovny jsou kolekce připravených fragmentů kódu, které lze použít (a znovu použít) k provádění běžných JavaScriptových funkcí. Kód JS knihovny lze v případě potřeby použít připojením kódu dané knihovny ke zbytku kódu. Pokud bychom například chtěli použít jQuery knihovnu pro přidání funkce automatického doplňování pro vyhledávací panel, stačí do kódu vložit příslušný jQuery kód, který danou funkci načte z jQuery knihovny a ta poté poskytne požadovaný obsah (Morris, 2020).

1.5 JavaScriptové frameworky

Pokud se podíváme na doslovný překlad slova framework dojdeme ke slovům jako jsou kostra, struktura, skelet nebo systém. To JS frameworky přesně dělají, poskytují strukturu (kostru), na které je poté postaven celý projekt. Tato struktura je vytvořena prostřednictvím šablon stránek (vytvořených frameworkem), které obsahují specifické oblasti vyhrazené pro vkládání kódu.

Výhodou využití JS frameworků je celková efektivita a organizace, kterou sebou do projektu přináší. Kód bude úhledně strukturován díky šablonám a frameworky poskytují hotová řešení pro běžné problémy s kódováním. Na druhou stranu, celá tato struktura může být naopak i nevýhodou, protože jakýkoliv další kód, který je nad rámec daného frameworku, musí dodržovat pravidla a konvence specifické pro daný framework (Morris, 2020).

Nicméně moderní JavaScriptové frameworky, jako například Vue.js už řeší i tuto problematiku. JS frameworky je možné aplikovat na jakkoli velké části daného webu či aplikace a není tedy nutné, aby veškerý kód podléhal specifickým pravidlům daného frameworku.

1.6 JavaScriptové frameworky versus knihovny

Zatímco JavaScriptové knihovny jsou specializovaným nástrojem pro řešení individuálních problémů, frameworky jsou celou sadou nástrojů, které pomáhají formovat a organizovat danou webovou aplikaci. JavaScriptové frameworky jsou tedy kolekce JavaScriptových knihoven, které vývojářům poskytují předem napsaný JavaScriptový kód, který doslova slouží k tvorbě vlastní webové stránky či aplikace.

JS frameworky se od knihoven liší především tím, že celý proces je více komplexní. Framework nenabízí pouze individuální řešení problému jako je tomu u knihoven, ale nabízí celou strukturu, která organizuje ty části, kde je daný framework implementován (Morris, 2020).

1.7 Rozšíření jazyka JavaScript

Tato podkapitola popisuje dvě rozšíření jazyka JavaScript, TypeScript a JSX.

TypeScript, vyvinutý společností Microsoft, je open sourcový programovací jazyk, který slouží jako nadstavba JavaScriptu. Tento jazyk nebyl vytvořen jako náhrada JavaScriptu, jedná se spíše o rozšíření, které je navrženo tak, aby pomáhalo s některými problémy JavaScriptu. TypeScript přidává do JavaScriptu statické typování, třídy a rozhraní. TypeScript tedy z JavaScriptu dělá kompletní programovací jazyk, protože do něj přidává silně typovanou deklarativní strukturu. Díky tomu je kód méně náchylný na chyby a přispívá k lepší udržovatelnosti kódu. Toto je obzvláště důležité při práci s rozsáhlými aplikacemi (Afifi-Sabet, 2019 a Kumar, 2019).

JavaScript extension zkráceně JSX, je syntaxe využívaná Reactem, který rozšiřuje JavaScript takovým způsobem, že umožňuje psát HTML elementy uvnitř JavaScriptového kódu. JSX převede HTML tagy na elementy Reactu. Ukázka syntaxe JSX (Lindley, 2019):

const element = <h1>Tady je ukázka JSX!</h1>;

1.8 DOM

DOM neboli Document Object Model je programovací rozhraní pro HTML a XML dokumenty. Reprezentuje stránku tak, aby programy mohli měnit strukturu, styl a obsah daného dokumentu. DOM je objektově orientovaná reprezentace webové stránky v podobě uzlů a objektů. Programovací jazyky, jako například JavaScript, se mohou tímto způsobem k dané stránce připojit a upravovat ji (MDN, 2020).

1.9 Routing

Téměř každý web nebo webová aplikace používá routing (směrování). Přechod z jedné webové stránky na druhou pomocí změny URL adresy je velmi užitečná funkce, která se ve webovém prostředí stala již běžnou záležitostí. Routing je tedy mechanismus, pomocí kterého jsou webové requesty (požadavky) spojeny s určitým kódem. Kliknutím na odkaz

se URL adresa změní, což uživateli poskytne nová data či novou webovou stránku. Řešení routingu je již samozřejmostí většiny moderních frameworků, například u Angularu je již součástí frameworku, u Vue se jedná o oficiální knihovnu, kterou je možné do frameworku přidat (Schepenaar, 2017).

1.10 State Management

State management neboli správa stavu aplikace je proces či metoda udržování znalostí o stavu a vstupech aplikace. Stav jsou veškeré informace, které si daná aplikace ukládá, často s ohledem na předchozí události a interakce. Například DOM má vlastnost spravovat stav formulářů bez nutnosti dalších zásahů (tj. pamatuje si uživatelské vstupy).

Čím větší a komplexnější se aplikace stává, tím více může být správa stavu zmatenější a nepřehlednější, zejména když aplikace umožňuje mnoho uživatelských interakcí, které je třeba spravovat. Proto je důležité vybrat pro správu stavu správné nástroje, které s tímto problémem pomáhají, určují, kde se stav může změnit, jaké mohou být potenciální následky a usnadnit tak i budoucí ladění chyb. Tyto nástroje jsou obvykle ve formě knihoven, které do aplikace přináší povědomí o stavu. Například u Angularu je state management již součástí frameworku (Rouse, 2020).

1.11 NPM a Node.js

NPM neboli Node Package Manager zastupuje dvě věci. V první řadě se jedná o online úložiště pro publikování otevřených projektů Node.js, a za druhé se jedná o nástroj příkazového řádku pro interakci se zmíněným uložištěm. Tento nástroj pomáhá při instalaci balíčku, správě verzí a řízení závislostí. Stačí mít vybraný balíček, který je potřebné nainstalovat a pomocí npm je možné tento balíček nainstalovat pomocí zadání jediného příkazu do příkazového řádku. Na npm úložišti je publikováno velké množství Node.js knihoven a aplikací a každý den přibývá mnoho dalších. Tyto aplikace je možné vyhledat na webové adrese http://npmjs.org/. Npm je součástí Node.js, takže po nainstalování Node.js prostředí jsou oba tyto nástroje připraveny k použití (OpenJS Foundation, 2011).

Node.js je otevřené a multiplatformní prostředí, které umožňuje spouštět JavaScriptový kód mimo webový prohlížeč. Toto prostředí je postaveno na Chrome V8 JavaScript enginu a slouží k vývoji serverových části webových aplikací. Největšími výhodami Node.js je rychlost a schopnost obsloužit mnoho připojených klientů naráz. Z tohoto důvodu je často používáno pro tvorbu tzv. API serverů pro klientské SPA. Node.js mimo jiné poskytuje bohatou knihovnu různých JavaScriptových modulů, které do značné míry zjednodušují vývoj webových aplikací (Tutorials Point, 2020 a Máca, 2018).

1.12 SPA

Single page application, zkráceně SPA, je jednostránková aplikace, která běží ve webovém prohlížeči a během používání nevyžaduje opětovné načtení stránky. Díky tomu se mnohem více podobá desktopové aplikaci a vytváří tak „přirozenější“ prostředí. Aplikacemi, které využívají přístupu SPA jsou například Gmail, Google Maps, Facebook, Twitter a mnoho dalších. Hlavní výhodou těchto aplikací je rychlost. Většina potřebných zdrojů je načtena při prvním načtení aplikace a jediné co se mění, jsou data. Výsledkem je, že aplikace reaguje velmi rychle na dotazy uživatele bez nutnosti čekat na komunikaci mezi klientem a serverem.

SPA využívá HTML5 a Ajax k plynulým a dynamickým odpovědím na dotazy uživatelů, což umožňuje okamžitou aktualizaci obsahu v případě, že uživatel provede nějakou akci. Ve chvíli, kdy se stránka načte, interakce se serverem probíhá skrze Ajax volání, aby se stránka mohla aktualizovat bez nutnosti opětovného načtení (Rouse, 2016 a Skólski 2016).

2 Analýza a porovnání moderních JS frameworků

Jak již bylo zmíněno výše, práce je zaměřena na frontendové frameworky, především pro vývoj jednostránkových aplikací (viz 1.12 SPA). Existuje mnoho různých JavaScriptových frameworků a analýza všech těchto frameworků by byla velmi časově náročná. Pro tuto práci byly tedy vybrány pouze tři konkrétní frameworky, které v současné době dominují jak v popularitě, tak v počtu uživatelů. Těmito frameworky jsou Angular, React a Vue. Tato kapitola je tedy věnována nejdříve krátkému představení každého z uvedených frameworků a poté jejich porovnání podle různých kritérií.

2.1 Představení vybraných frameworků

Tato podkapitola je věnována krátkému představení jednotlivých JavaScriptových frameworků. Je zde uvedeno kdy daný framework vznikl, kdo je jeho autorem a příklady velkých společností, které daný framework využívají.

2.1.1 **Angular**

Angular, patřící společností Google, byl vytvořen v září 2010 pod jménem AngularJS, čímž je nejstarším frameworkem z vybraných frameworků. Nicméně v roce 2016, přichází nový Angular 2, který je označován pouze jako Angular. AngularJS a Angular bývají často zaměňovány, protože AngularJS je stále podporovaný a hojně využívaný. Tyto dva frameworky jsou však navzájem nekompatibilní. V dalších částech této práce je věnována pozornost novějšímu frameworku z této dvojice, tedy Angularu (Daityari, 2020).

Angular je frontendový framework využívající jazyk TypeScript k tvorbě webových aplikací. Díky tomu byl tento framework skvělý pro ty, kteří přišli z tradičních objektově orientovaných jazyků jako Java a C#, protože TypeScript se těmito jazyky inspiruje. Angular je využíván ve společnostech jako například Google, PayPal, Sony, Mastercard, Wix a mnoho dalších. Více informací je možné zjistit na oficiální webové adrese angular.io (Kopilevych, 2020).

2.1.2 **React**

React byl vyvinutý společností Facebook a na trh byl poprvé uveden v květnu 2013. Originální autorem byl Jordan Walke, inženýr ve společnosti Facebook. React se prezentuje spíše jako javascriptová knihovna pro tvorbu uživatelských rozhraní než framework, na rozdíl od frameworku jako například Angular. Problémy jako routing nebo state management byli ponechány třetím stranám, a to vedlo ke vzniku velkého a velmi aktivního ekosystému okolo Reactu (Daityari, 2020).

Mnoho velkých Reactových aplikací využívá pro state management knihovnu Redux a React Router pro routing, ale existuje mnoho dalších dobrých alternativ. React je nesmírně populární frontendový framework, který využívají nejrůznější společnosti jako například Facebook, Reddit, Netflix, Instagram, Droopbox, WhatsApp, Uber a mnoho dalších. Oficiální webové stránky jsou reactjs.org (Stencil, 2018).

2.1.3 **Vue**

Vue, také označování jako Vue.js, je nejmladším zástupcem z vybraných frameworků. Vue bylo vyvinuto, bývalým zaměstnancem společnosti Google, Evenem You a na trhu poprvé uvedeno v únoru 2014. V posledních třech letech se tento framework dočkal výrazné popularity, přestože za sebou nemá žádnou velkou společnost, která by ho přímo podporovala, na rozdíl od Angularu a Reactu (Daityari, 2020).

Vue je v mnoha aspektech podobné Reactu, ale také toho má hodně společného s AngularJS. Zatímco v Angularu je routing a state management součástí frameworku a v Reactu je řešení těchto problému naopak ponecháno knihovnám třetích stran, tým Vue vybírá třetí možnost a vytváří oficiální řešení těchto problémů, které jsou dobrovolné, ale přesto udržovaná v synchronizaci s hlavní knihovnou (Stencil, 2018).

Relativní jednoduchost, zkušenosti vývojářů, a dobrý výkon hodně přispěly k obrovskému nárůstu popularity tohoto frameworku. Další výhodou je to, že se jedná o „progresivní framework“ a díky tomu může být použit jak jako náhrada jQuery, tak pro velké SPA (Daityari, 2020).

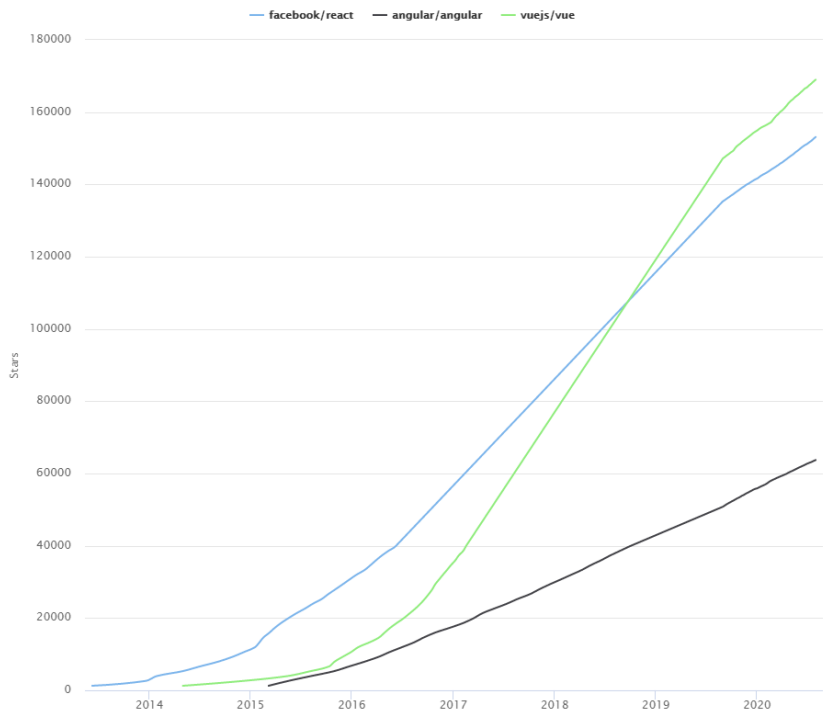
Vue je využíváno ve stále více společnostech, kterými jsou například Xiaomi, Alibaba, Grammarly, GitLab, Euronews a další. Jak již bylo zmíněno výše, za frameworkem Vue nestojí žádná velká společnost, a proto je podporován různými sponzory, které je možné nalézt na oficiální webové stránce vuejs.org (Stencil, 2018).

2.2 Porovnání

Následující podkapitola je věnována porovnání vybraných JavaScriptových frameworků. Frameworky jsou porovnávány podle různých hledisek jako je popularita, obtížnost učení, komunita či budoucí vývoj.

2.2.1 Popularita

Protože „angular“ a „react“ jsou běžně vyhledávaná slova, je obtížné získat komplexní pohled na popularitu vybraných frameworků pomocí Google Trends. A proto je v této práci pohlíženo na popularitu jednotlivých frameworků pomocí uložiště GitHub, kde jsou frameworky hodnoceny počtem hvězdiček.



Obrázek 5: Popularita vybraných frameworků v posledních letech
Zdroj: Daityari, 2020

Jak je vidět na obrázku 5, v druhé polovině roku 2016 došlo k náhlému zvýšení počtu hvězdiček frameworku Vue. V polovině roku 2018 pak framework Vue překonal i framework React a nyní patří Vue s Reactem mezi nejoblíbenější frameworky současnosti (Spec India, 2018 a Daityari, 2020).

2.2.2 **Komunita a vývoj**

Jelikož za Angularem a Reactem stojí velké společnosti, Google a Facebook, není pochyb o dalším růstu a vývoji obou těchto frameworků. Nicméně ani Vue nezaostává, například v tomto roce je plánované vydaní nového Vue 3, které bude založeno na Typescriptu. Pro všechny tři frameworky jsou pravidelně vydávány nové verze a aktualizace, což naznačuje, že na vývoji těchto frameworcích se neustále pracuje.

Pokud je na frameworky pohlíženo opět přes statistiky na uložišti Github viz tabulka 1, tak je vidět, že Vue má velký počet sledujících a hvězdiček, na druhou stranu má o hodně menší počet přispívajících. Čím více lidí přispívá a spolupracuje na funkcionalitách daného frameworku, tím lepším se stává jak kolekce knihoven okolo daného frameworku, tak samotný framework (Kopilevych, 2020 a Daityari, 2020).

Tabulka 1: Github statistiky

	Angular	React	Vue
Watchers	3.2 tis.	6.6 tis.	6.0 tis.
Stars	57 tis.	144 tis.	157 tis.
Forks	16.9 tis.	29.4 tis.	25.2 tis.
Contributors	1,089	1,361	289

Zdroj: vlastní zpracování podle Daityari, 2020

Možným vysvětlením, proč je Vue pozadu oproti svým kolegům v počtu přispívajících je, že celý tento framework je poháněn pouze open-sourcovou komunitou, zatímco Angular

a React mají velký podíl přispívajících v podobě zaměstnanců Googlu a Facebooku (Kopilevych, 2020 a Daityari, 2020).

2.2.3 **Složitost učení**

Angular, oficiální dokumentace https://angular.io/docs, je z vybraných frameworků nejvíce náročný na naučení, bereme-li v potaz celé řešení. Navíc zvládnutí frameworku jako takového souvisí i s pochopením konceptů jako TypeScript a MVC. Přestože naučení se toho frameworku trvá dlouho, tato investice do pochopení front-end vývoje se určitě vyplatí.

React má na svých stránkách https://reactjs.org/docs/getting-started.html průvodce, který by měl pomoci k seznámení se základy frameworku. Dokumentace je důkladná a s řešením běžných problémů. React není komplexní framework a pokročilé funkce vyžadují použití knihoven třetích stran. Naučení se základní struktury tedy není tak náročné jako u Angularu, ale je nutné brát zřetel také na dodatečné funkce.

Vue, oficiální průvodce https://vuejs.org/v2/guide/#Getting-Started, nabízí vyšší přizpůsobitelnost, a proto se snáze učí než oba předešlé frameworky. Vue má navíc společné rysy jak s Angularem, tak s Reactem. Z toho důvodu je přechod z obou těchto frameworků snadnou volbou. Nicméně jednoduchost a flexibilita Vue může být dvojsečným mečem, jelikož umožňuje tvorbu horšího kódu, který ztěžuje ladění a testování.

Vývoj v Angularu i v Reactu vyžaduje dobré znalosti JavaScriptu a rozhodování se okolo knihoven třetích stran. Angular a React mají specifické způsoby, jak dosáhnout určitých cílů, Vue je oproti tomu značně jednodušší. Stálé více společností tedy přechází na Vue, protože se s ním pracuje snadněji. U Vue může být však problémem ladění a testování (Daityari, 2020).

2.3 Shrnutí

Tato podkapitola je věnována celkovému shrnutí zjištěných údajů k daným frameworkům. Nejprve jsou jednotlivé frameworky stručně popsány a poté jsou přehledně seřazeny základní informace o vybraných frameworcích do tabulky.

Angular je plně vybavený framework, který poskytuje vše potřebné pro vývoj webové aplikace. Řešení problémů jako routing a state management je součástí frameworku. Hodí se především pro velké webové aplikace. Vývoj probíhá v jazyce TypeScript. Nabízí osvědčené postupy, jak psát kód a organizovat projekt. Protože vše potřebné pro vývoj je již součástí frameworku, nenabízí příliš velkou flexibilitu, ve výběru možných řešení. Angular je sice více obtížný na naučení než jeho kolegové, zato ale nabízí mnoho zajímavých funkcí a nástrojů. Stojí za ním společnost Google, díky čemuž má velký počet přispívajících uživatelů, popularita je však menší než u Reactu a Vue, to ale neznamená, že by tento framework nebyl populární.

React není framework, ale JavaScriptová knihovna pro tvorbu UI. Řešení problémů jako routing a state management není součástí, je potřeba využít knihoven třetích stran. Hodí se jak pro velké, tak malé webové aplikace. Vývoj probíhá v jazyce JavaScript potažmo v JSX, je však možné použít i TypeScript. Díky tomu, že je React závislý na knihovnách třetích stran, vzniklo velké množství možností, které nabízejí velkou flexibilitu ve výběru řešení. To však může být i nevýhodou, záleží na pohledu jednotlivce. Kvůli jeho funkcionálnímu přístupu k programování a využití JSX místo jednoduchého HTML, není příliš jednoduchý na naučení. Za Reactem stojí společnost Facebook, je velice populární a má velký počet přispívajících uživatelů.

Vue je střední cestou mezi plně vybaveným Angularem a strohým Reactem. Řešení problémů jako routing a state management sice není součástí frameworku, zato Vue nabízí oficiální knihovny pro jejich řešení. Tyto knihovny jsou spravovány týmem Vue, čímž je zaručena jejich dobrá spolupráce s frameworkem. Vue se hodí především pro menší a středně velké webové aplikace. Vývoj probíhá v jazyce JavaScript, je však možné použít i TypeScript (Vue 3 bude nově používat TypeSciprt namísto JavaScriptu). Výhodou Vue je také flexibilita, protože je možné použít jak oficiální knihovny, tak knihovny třetích stran. Vue je jednodušší na učení ve srovnání s Angularem a Reactem. I přes to, že za tímto frameworkem nestojí žádná velká společnost, tak je velice populární, má však malý počet přispívajících uživatelů.

Součástí shrnutí je i tabulka porovnávající základní informace o vybraných frameworcích viz tabulka 2.

Tabulka 2: Shrnutí základních informací

	Angular	React	Vue
Typ	Framework	JS knihovna pro tvorbu UI	Framework
Zakladatel	Google	Facebook (Jordan Walke)	Evan You
Datum vydání	Září 2010	Květen 2013	Únor 2014
Napsaný v jazyce	TypeScript	JavaScript	JavaScript
Vývoj v jazyce	TypeScript	JSX	JavaScript
Podpora Typescriptu	Ano	Ano	Ano
Routing	Ano	Ne – neoficiální knihovny	Ne – oficiální knihovny
State Management	Ano	Ne – neoficiální knihovny	Ne – oficiální knihovny
Učení	Těžké	Středně těžké	Lehké
Společnosti využívající daný framework	Google, PayPal, Sony, Mastercard, Wix a další	Facebook, Reddit, Netflix, Instagram, Droopbox, WhatsApp, Uber a další	Xiaomi, Alibaba, Grammarly, GitLab, Euronews a další
Popularita (v hvězdičkách)	57 tis.	144 tis.	157 tis.
Počet přispívajících uživatelů	1,089	1,361	289

Zdroj: vlastní zpracování podle Daityari, 2020

3 Základní syntaxe jednotlivých frameworků

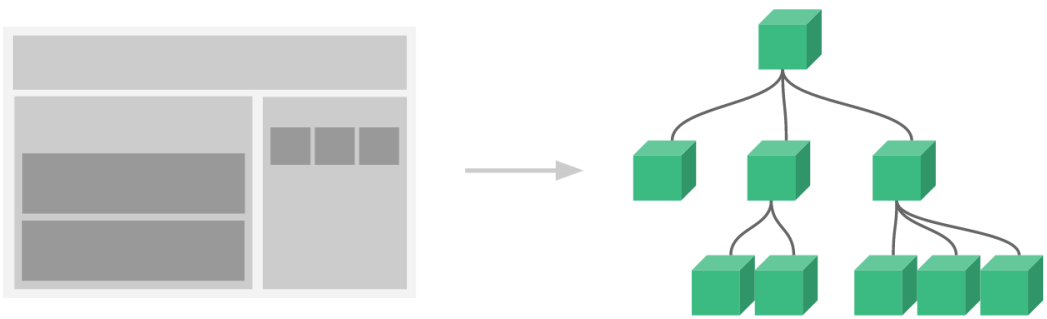
Tato kapitola má za úkol ukázat možné varianty základní syntaxe vybraných JavaScriptových frameworků, tedy Angularu, Reactu a Vue, a případně i poukázat na jejich odlišnosti. Jednotlivé podkapitoly jsou vždy rozděleny do tří sekcí, každá tato sekce bude věnována jednomu z vybraných frameworků.

Všechny následující příklady syntaxe se snaží řídit osvědčenými způsoby a konvencemi vybraných frameworků. Nejedná se vždy o jediný možný zápis. K vybrané ukázce syntaxe je ve většině případů ponechán i okolní zdrojový kód, aby mohlo dojít k pochopení dané problematiky jako celku.

Ukázky zdrojového kódu v jednotlivých podkapitolách se vždy snaží o dosažení stejného výsledku ve vybraných frameworcích, aby bylo možné co nejlépe identifikovat odlišnosti mezi jednotlivými syntaxemi.

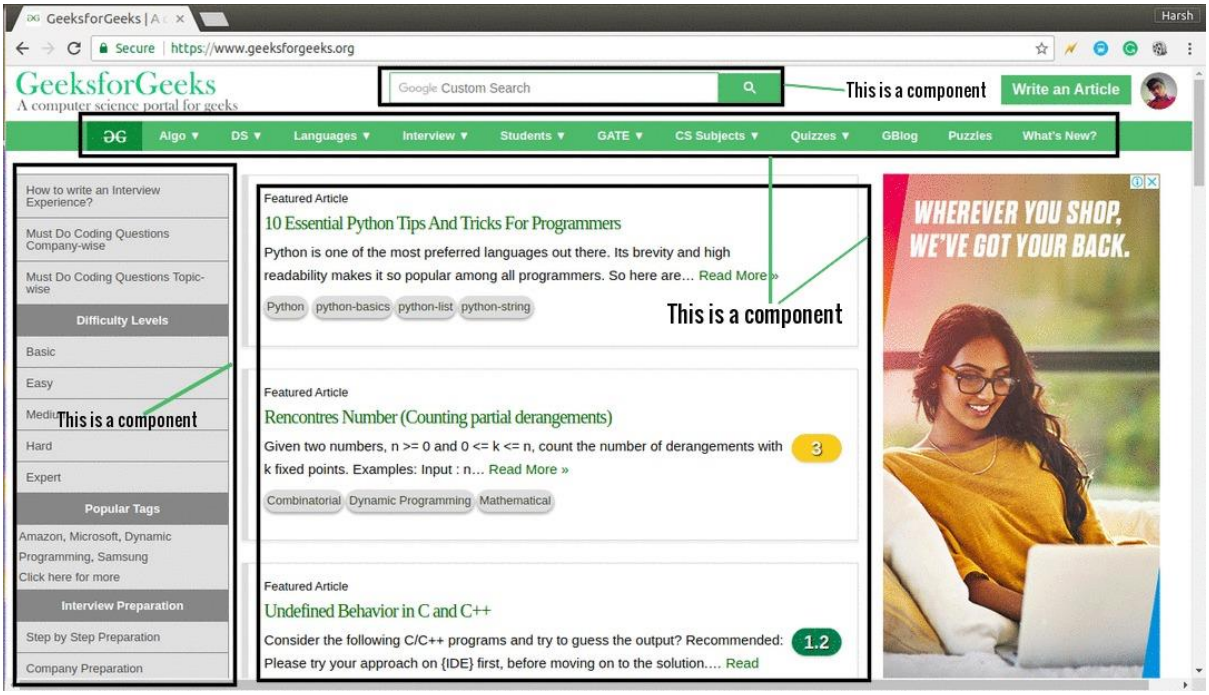
3.1 Komponenty

Komponenty jsou základními stavebními jednotkami Angularu, Reactu i Vue. Pro aplikace využívající tyto frameworky je tedy běžné, že jsou organizovány jako strom komponent viz obrázek 6 (Vue.js, 2020).



Obrázek 6: Strom komponent
Zdroj: Vue.js, 2020

Komponentou může být například záhlaví, zápatí, postranní panel nebo menu, kde každá tato komponenta obvykle obsahuje další komponenty, jako jsou navigační odkazy, články a mnoho dalších viz obrázek 7.



Obrázek 7: Rozdělení stránky na komponenty
Zdroj: Agarwal, 2019

V této podkapitole je tedy dále věnována pozornost tomu, jak definovat jednoduchou komponentu ve vybraných JavaScriptových frameworcích.

3.1.1 **Angular**

V Angularu je k definování komponenty nejdříve zapotřebí přidat tzv. dekorátor. Dekorátor umožňuje uživateli přidat do existujícího objektu nové funkce bez nutnosti jeho změny. V podstatě jde o obal, který slouží k rozšíření daného objektu. Dekorátor pro komponentu se v Angularu označuje klíčovým slovem *Component* a přidává se z balíku *@angular/core* ve kterém je definována základní funkcionalita Angularu, nízko úrovňové služby a různé utility (Tutorial Point, 2019 a Angular, 2020).

Nejprve je tedy dekorátor přidán následujícím způsobem: *import { Component } from '@angular/core'*. A poté je možné se na tento dekorátor v kódu odkázat pomocí *@Component*.

V Angularu se tedy komponenta definuje přidáním výše zmíněného dekorátoru a přidáním třídy, která poté již obsahuje funkcionalitu dané komponenty. Definice komponenty v Angularu tedy může vypadat například jako v následující ukázce.

```
import { Component } from '@angular/core';
@Component({
...
})
export class WelcomeComponent {
}
```

Ukázka kódu 1: Komponenta v Angularu

Neměla by být opomíjena ani skutečnost, že k tomu, aby tato komponenta mohla být dostupná pro jiné komponenty nebo aplikace je nutné, aby byla deklarována uvnitř Angular modelu. Kam je nejprve potřeba danou komponentu přidat a potom ji zmínit uvnitř *@NgModule* dekorátoru, za klíčovým slovem *declarations* viz ukázka 1 (Angular, 2020).

```
import { NgModule } from '@angular/core';
import { WelcomeComponent } from './welcome.component';
@NgModule({
  declarations: [WelcomeComponent]
})
export class ... {}
```

Ukázka kódu 2: Přidaní komponenty do Angular modulu

3.1.2 **React**

React umožňuje definovat komponenty jako třídy nebo jako funkce. Komponenta definovaná jako funkce, která je pojmenována *"Welcome"*, neobsahuje žádné parametry a pouze vrátí nadpis *"Hello"* vypadá jako v následující ukázce.

```
const Welcome = () => <h1>Hello</h1>;
```

Ukázka kódu 3: Komponenta jako funkce v Reactu

Komponenty definované jako třídy v současné době poskytují více možností. K definování React komponenty jako třídy, je použito klíčové slovo *class*, které označuje, že se jedná o třídu a poté následuje název dané komponenty. K definování React komponenty je ale zapotřebí tuto třídu ještě rozšířit o základní třídu *React.Component*. Toho je docíleno pomocí klíčového slova *extends,* viz ukázka kódu 4.

```
class NazevKomponenty extends React.Component {}
```

Ukázka kódu 4: Komponenta jako třída v Reactu

Důležitou skutečností je také to, že nově definovaná komponenta musí vždy obsahovat metodu render(). V ukázce 5 je demonstrován příklad komponenty "Welcome", obsahující povinnou metodu render(), která vrátí nadpis "Hello" (React, 2020 a Kagga, 2018).

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello</h1>;
  }
}
```

Ukázka kódu 5: Komponenta v Reactu

3.1.3 **Vue**

Ve Vue je nejprve nutné do souboru přidat Vue samotné, k tomu dojde pomocí následujícího řádku:

```
import Vue from 'vue';
```

Ukázka kódu 6: Import Vue

Poté je už možné k Vue přistupovat pomocí tečkové notace. K definici nové komponenty je potřeba napsat Vue.component a kulaté závorky. Uvnitř těchto závorek se pak v apostrofech

nachází název dané komponenty, za kterým následuje funkcionalita dané komponenty ve složených závorkách viz ukázka 7.

```
Vue.component('nazev-komponenty', { kód komponenty })
```

Ukázka kódu 7: Definice komponenty ve Vue

V ukázce 8 je předvedeno, jak ve Vue definovat podobnou komponentu jako v předcházející podkapitole, tedy komponentu, která se bude jmenovat “welcome” a vrátí nadpis “Hello” (Vue.js, 2020).

```
import Vue from 'vue'

Vue.component('welcome', {
  data(): function() {
    return {
       <h1>Hello</h1>
    }
  }
})
```

Ukázka kódu 8: Komponenta ve Vue

3.2 Vkládání závislostí

Vkládání závislostí (dependency injection) je důležitou součástí návrhu každé aplikace. Závislosti jsou různé služby nebo objekty, které jsou potřebné k výkonu dalších funkcí. Dependency injection je tedy kód ve kterém se komponenta ptá, požaduje tyto závislosti od externích zdrojů namísto jejich vytváření (Angular, 2020).

Pro všechny tři vybrané frameworky platí, že:

- vkládání závislostí se zpravidla provádí na začátku souboru
- využívají klíčová slova *import* a *from*
- po slově *import* je uveden název dané závislosti
- po slově *from* je uvedena cesta, kde se závislost nachází

3.2.1 **Angular**

Syntaxe vkládání závislostí v Angularu vypadá tedy následovně: *import { Název } from 'cesta k souboru'*.

Při vkládání služby (service) je v Angularu, na rozdíl od Reactu a Vue, ještě nutné uvést danou závislost definovat v konstruktoru viz ukázka 9 (Angular, 2020).

```
import { Component } from '@angular/core';
import { Notification } from 'services/notification'; //vložení
služby

@Component({
  ...
})
export class ... {
  constructor(
    private notification: Notification, // definice závislosti v
konstuktoru
  ) {}
}
```

Ukázka kódu 9: Vkládání závislostí v Angularu

3.2.2 **React**

Vkládání závislostí v Reactu nemá žádný speciální mechanismus, vypadá následovně: *import Název from 'cesta k souboru'*. Jediným rozdílem v této syntaxi oproti Angularu je v tom, že název dané závislosti není obalen složenými závorkami (Grzybek, 2018).

```
import React, { Component } from 'react';
import Notification from 'utils/notification'; //vložení služby

export class ... extends Component {
...
}
```

Ukázka kódu 10: Vkládání závislostí v Reactu

3.2.3 **Vue**

Ve Vue je syntaxe vkládání závislostí stejná jako u Reactu: *import Název from 'cesta k souboru'*. Všechny závislosti jsou jednoduše vyjmenovány na začátku souboru (Grzybek, 2018).

```
import Vue from 'vue';
import Notification from 'utils/notification'; //vložení služby

Vue.component(..., {
...
});
```

Ukázka kódu 11: Vkládání závislostí ve Vue

3.3 Šablony (templates)

Šablonami neboli templates se v tomto kontextu rozumí HTML kód, který je součástí každé webové aplikace. Každý z vybraných frameworků, řeší umístění šablony jiným způsobem.

V následujících podkapitolách je tedy stručně popsán přístup jednotlivých frameworků a syntaxe předávání možných atributů a vlastností do těchto šablon.

3.3.1 **Angular**

V Angularu má každá template svůj vlastní soubor, který je následně vkládán do příslušné komponenty pomocí klíčového slova *tepmlateUrl* spolu s cestou a názvem dané template, uzavřené v apostrofech.

```
templateUrl: './zmen-heslo.component.html'
```

Ukázka kódu 12: Template Angular

Template a komponenta v Angularu zpravidla tvoří pár a dodržují konvenci stejných pojmenování. Například komponenta pro změnu hesla je pojmenována *zmen-heslo.component.ts*, tudíž template k této komponentě by měla být pojmenována *zmen-heslo.component.html*.

Následující ukázka kódu demonstruje vložení template do komponenty s názvem *zmen-heslo.component.ts* (Angular, 2020).

```
//  zmen-heslo.component.ts
...
@Component({
  templateUrl: './zmen-heslo.component.html' // vložení template
})
export class ZmenHesloComponent {
  ...
}
```

Ukázka kódu 13: Vložení template v Angularu

Angular má tři druhy atributů, které umí předat do těchto šablon:

- bindování textu jako například *size="string"*
- bindování vlastností, např. *[disabled]="value"*
- bindování událostí (eventů), např. *(click) = "eventHandler ()"*

Příklad použití je vidět v následující ukázce:

```
<primary-button
  size="big"
  [disabled]="true"
  (click)="ulozContent()"
>
  Ulož
</primary-button>
```

Ukázka kódu 14: Atributy Angular

3.3.2 **React**

Šablona neboli template je v Reactu součástí JavaScriptového souboru, kde je psána pomocí jazyka JSX (viz podkapitola 1.7 Rozšíření jazyka JavaScript). Tímto způsobem se template přímo prolíná s JavaScripotvým kódem. JSX proto používá konvenci velkých a malých písmen k rozlišení mezi uživatelem definovanými komponentami a prvky DOM. V ukázce 15 je tlačítko *PrimaryButton* využívající právě zmíněnou konvenci malých a velkých písmen, na rozdíl od Angularu a Vue, kde je stejný prvek nazván *primary-button* (Grzybek, 2018).

React stejně jako Angular umí předat do šablony různé atributy:

- bindování textu jako například *size="string"*
- bindování vlastností, např. *disabled*
- bindování událostí (eventů), např. *onClick = { this.eventHandler }*

Příklad použití demonstruje následující ukázka:

```
<PrimaryButton //konvence velkých a malých písmen
  size="big"
  disabled
  onClick={ this.ulozContent }
>
  Ulož
</PrimaryButton>;
```

Ukázka kódu 15: Atributy React

3.3.3 **Vue**

Ve Vue jsou šablony také součástí JavaScriptového souboru, ale na rozdíl od Reactu mají svůj vlastní oddíl, čímž nedochází k míchání JavaScriptového kódu spolu s HTML kódem. Soubor je rozčleněn do tří částí: script, template a style. Část script obsahuje JavaScriptový kód, template zahrnuje HTML kód a ve style je CSS kód. Soubor vypadá nějak takto (Vue, 2020):

```
<script>
       // Tady je JavaScript kód!
</script>

<template>
       // Tady je HTML kód!
</template>

<style>
       // Tady je CSS kód!
</style>
```

Ukázka kódu 16: Rozdělení Vue souboru

Vlastnosti komponenty mohou být do šablony předávány:

- Přímo (jako řetezec), například *size="big"*
- Dynamicky, použitím v-bind (zkráceně znakem dvojtečky :),
    - např. *v-bind:disabled="true"*
- Pro eventy, použitím v-on (zkráceně znakem zavináče @) v kombinaci s názvem eventu, a názvem metody jako hodnoty, např. *v-on:click="ulozContent"*.

Příklad použití je vidět v následující ukázce:

```
<primary-button
  size="big"
  v-bind:disabled="true"    // :disabled="true"
  v-on:click="ulozContent" // @click="ulozContent"
>
    Ulož
</primary-button>
```

Ukázka kódu 17: Atributy Vue

Při porovnání ukázek 14, 15 a 17 je možné vidět několik rozdílů v syntaxi mezi všemi frameworky, například:

- Angular při bindování vlastností využívá hranatých závorek
- React při bindování událostí využívá složených závorek
- Vue vytváří vlastní jmenný prostor (namespace), *v-bind*, *v-on*

3.4 Interpolace

Nejzákladnější formou bindování dat je textová interpolace. Textová interpolace je proces vyhodnocování řetězců, které obsahují jeden nebo více zástupných symbolů. Výsledkem procesu je nahrazení těchto zástupných symbolů odpovídajícími hodnotami. Interpolace tedy umožňuje vložení výrazů a jejich následné přeložení uvnitř HTML elementů (Angular, 2020).

3.4.1 **Angular**

V Angularu interpolace používá jako oddělovač dvojité složené závorky. Příkladem interpolace v následující ukázce je *{{ image.alt }}* (Angular, 2020).

```
<img [src]="image.url" alt="{{ image.alt }}" />
```

Ukázka kódu 18: Interpolace Angular

3.4.2 **React**

React pro interpolaci používá také složené závorky, ale již ne dvojité *{ this.props.image.alt }* (Grzybek, 2018).

```
<img src={ this.props.image.url } alt={ this.props.image.alt } />;
```

Ukázka kódu 19: Interpolace React

3.4.3 **Vue**

Vue pro interpolaci využívá dvojité složené závorky, stejně jako tomu je u Angularu, tedy *{{ image.alt }}.*

```
<img :src="image.url" alt="{{ image.alt }}" />
```

Ukázka kódu 20: Interpolace Vue

Ve Vue je také možné vytvářet jednorázové interpolace, které se při změně dat neaktualizují. K této jednorázové interpolaci Vue používá atribut *v-once* (Vue.js, 2020).

```
<span v-once>Hello {{ username }}!</span>
```

Ukázka kódu 21: Jednorázová interpolace Vue

3.5 Základní hodnoty vlastností

V podkapitole 3.1 Komponenty je zmíněno vytvoření jednoduché komponenty. Tyto komponenty ale samy o sobě nebudou příliš užitečné, proto je potřeba je více rozvinout, tedy předávat do nich data. K tomuto účelu složí takzvané *props*, neboli vlastnosti komponenty. V této podkapitole je tedy ukázáno, jak je možné tyto vlastnosti definovat a přiřadit jim základní hodnoty (Grzybek, 2018).

V každé z následujících podkapitol je vytvořena vlastnost komponenty:

- název: *zobrazKnihy*
- datový typ: *boolean*
- základní hodnota: *true*

3.5.1 **Angular**

V Angularu se vlastnost komponenty definuje jedním řádkem (Angular, 2020):

```
název: datový typ = základní hodnota
```

Ukázka kódu 22: Vlastnost komponenty Angular

Příklad syntaxe je vidět v následující ukázce:

```
...
export class ... {
  zobrazKnihy: boolean = true; // přiřazení typu a základní hodnoty
}
```

Ukázka kódu 23: Základní hodnoty vlastností Angular

3.5.2 **React**

Syntaxe v Reactu je o něco komplikovanější než v Angularu, nejprve je nutné přiřadit datový typ.

```
název: datový typ
```

Ukázka kódu 24: Přiřazení datového typu React

Poté provést přiřazení základní hodnoty.

```
název: základní hodnota
```

Ukázka kódu 25: Přiřazení základní hodnoty React

Ukázka syntaxe pro React:

```
...
export class ... extends ... {
  static propTypes = {
    zobrazKnihy: PropTypes.bool // přiřazení typu
  };

  static defaultProps = {
    zobrazKnihy: true // přiřazení základní hodnoty
  };
}
export class ... {
  zobrazKnihy: boolean = true; // přiřazení typu a základní hodnoty
}
```

Ukázka kódu 26: Základní hodnoty vlastností React

3.5.3 **Vue**

Syntaxe ve Vue je obdobná syntaxi v Reactu, jen dle mého názoru je o něco jednodušší. Nejprve je specifikován název vlastnosti a poté následuje dvojtečka se složenými závorkami, které obsahují datový typ a základní hodnotu dané vlastnosti:

```
název: {type: datový typ, default: základní hodnota}
```

Ukázka kódu 27: Vlastnost komponenty Vue

Demonstrace syntaxe pro framework Vue:

```
...
Vue.component('...', {
  props: {
    zobrazKnihy: {
      type: Boolean, // přiřazení typu
      default: true // přiřazení základní hodnoty
    }
  }
});
```

Ukázka kódu 28: Základní hodnoty vlastností Vue

3.6 Podmínky a podmíněné vykreslování

Podmíněné vykreslování je založeno na vyhodnocování podmínek, tedy zda se výraz vyhodnotí jako pravda a dojde k vykreslení určitého bloku, nebo se výraz vyhodnotí jako nepravdivý a k vykreslení daného bloku nedojde.

3.6.1 **Angular**

V Angularu jsou pro podmínky využívány direktivy *ngIf*, *ngIf;else* nebo *ngIf;then;else*. Direktiva *NgIf* je na oficiálních stránkách Angularu popsána jako strukturální směrnice obsahující šablonu, která je zobrazena na základě vyhodnocení *boolean* výrazu. Když se

výraz vyhodnotí jako true (pravda), Angular vykreslí šablonu uvedenou v klauzuli *then*, a pokud false nebo null, Angular vykreslí šablonu uvedenou v klauzuli *else*. Výchozí šablona klauzule *else* je prázdná (Angular, 2020).

Pro podmínky je možné použít rozšířenou syntaxi, při niž je direktiva *ngIf* obalena hranatými závorkami a umístěna uvnitř elementu *ng-template* viz ukázka 29.

```
<ng-template [ngIf]="podmínka">
  <div>Obsah k vykreslení pokud je podmínka splněna.</div>
</ng-template>
```

Ukázka kódu 29: Podmínka ngIf Angular

Obecně je ale více používaná zkrácená forma syntaxe, **ngIf= "podmínka"*, která je uvedena jako atribut daného elementu viz ukázka 30.

```
<div *ngIf="podmínka">
  Obsah k vykreslení pokud je podmínka splněna.
</div>
```

Ukázka kódu 30: Zkrácená podmínka Angular

Ukázka zkrácené syntaxe pro *ngIf* s *else* blokem.

```
<div *ngIf="podmínka; else elseBlok">
  Obsah k vykreslení pokud je podmínka splněna.
</div>
<ng-template #elseBlok>
  Obsah k vykreslení pokud není podmínka splněna.
</ng-template>
```

Ukázka kódu 31: Podmínka s else blokem Angular

3.6.2 **React**

Podmíněné vykreslování v Reactu funguje stejným způsobem jako v JavaScriptu. To tedy znamená, že React nemá žádnou speciální syntaxi a používá JavaScriptové operátory jako *if* nebo ternární operátor. Běžnějším přístupem k podmíněnému vykreslování v Reactu je ternární operátor, ale každému může vyhovovat jiný přístup, a proto jsou v této práci zmíněné obě varianty (React, 2020).

V následující ukázce je v metodě *render* vytvořena proměnná *obsah*, do které je poté na základně vyhodnocení podmínky přiřazena hodnota. Tuto proměnnou je poté možné například vykreslit, nebo jí dále využívat, to již, ale není předmětem této podkapitoly.

```
render() {
  let obsah; // definice proměnné
  if (podmínka) {
    obsah = Obsah k vykreslení, pokud je podmínka splněna.
  } else {
    obsah = Obsah k vykreslení, pokud není podmínka splněna.
  }
}
```

Ukázka kódu 32: Podmínky React

Ternární operátor se typicky skládá ze tří částí:

- podmínka
- co se má stát, pokud bude podmínka splněna
- co se má stát, pokud nebude podmínka splněna

Tyto částí jsou odděleny klíčovými znaky, tedy: *podmínka ? true : false*.

V ukázce 33 je demonstrováno použití ternárního operátoru na elementu *div*.

```
<div>
  { podmínka
    ? Obsah k vykreslení, pokud je podmínka splněna.
    : Obsah k vykreslení, pokud není podmínka splněna.
  }
</div>
```

Ukázka kódu 33: Ternární operátor React

3.6.3 **Vue**

Vue pro podmínky a podmíněné vykreslování používá tři vlastní atributy *v-if*, *v-else* a *v-else-if*. Použití těchto atributů je obdobné jako v Angularu. Atribut *v-if* se používá k podmíněnému vykreslení bloku, který je vykreslen pouze v případě, že výraz tohoto atributu vrátí pravdivou hodnotu. Ukázka 34 demonstruje jednoduché použití atributu *v-if* na elementu *div*.

```
<div v-if="podmínka">
  Obsah k vykreslení, pokud je podmínka splněna.
</div>
```

Ukázka kódu 34: Podmínka v-if Vue

V-else atribut se používá k označení *else* bloku, který má být proveden, pokud nedojde ke splnění podmínky ve *v-if* elementu. Je důležité zmínit, že *v-else* element musí okamžitě následovat za elementem *v-if* nebo *v-else-if* elementem jinak nebude rozpoznán.

Atribut *v-else-if*, jak již jméno napovídá, slouží jako *else if* blok, tedy pokud nedojde ke splnění první podmínky, *v-else-if* element obsahuje druhou podmínku atd. Tento element, stejně jako *v-else* musí okamžitě následovat za elementem *v-if* nebo jiným *v-else-if* elementem, jinak nedojde k jeho rozpoznání (Vue.js, 2020).

```
<div v-if="1.podmínka">
  Obsah k vykreslení, pokud je 1.podmínka splněna.
</div>
<div v-else-if="2.podmínka">
  Obsah k vykreslení pokud,
1.podmínka není splněna a 2.podmínka je splněna.
</div>
<div v-else>
 Obsah k vykreslení, pokud není ani jedna z podmínek splněna.
</div>
```

Ukázka kódu 35: Podmínky Vue

3.7 Formuláře

Zpracovávání uživatelských vstupů pomocí formulářů je základním kamenem mnoha běžných aplikací. Aplikace tyto formuláře využívají k tomu, aby se uživatelé mohli přihlásit, aktualizovat svůj profil, zadávat citlivé informace, a k mnoha dalším úkonům, při kterých je potřebné zadávání dat. Tato podkapitola demonstruje možné způsoby vytvoření jednoduchých formulářů pro jednotlivé frameworky (Angular, 2020).

3.7.1 **Angular**

Angular nabízí dva různé přístupy k manipulaci uživatelského vstupu prostřednictvím formulářů, reaktivní a řízené šablonou. Tyto přístupy se liší ve zpracování a správě dat z formulářů. Obecně lze říci, že reaktivní formuláře jsou robustnější: rozsáhlejší, znovu použitelné a testovatelné. Pokud jsou tedy formuláře klíčovou součástí dané aplikace, je lepší využít tento přístup.

Formuláře řízené šablonou jsou užitečné pro přidání jednoduchého formuláře. Lze je snadno přidat do aplikace, ale nemají takový rozsah jako reaktivní formuláře. Pokud jsou kladeny na formulář pouze základní požadavky a logika, která lze spravovat pouze v šabloně, je vhodné využít tento přístup. Pro porovnání těchto dvou přístupů je v následující části této podkapitoly vytvořen stejný formulář s jedním vstupním polem oběma metodami.

V následující ukázce je komponenta se vstupním polem implementována pomocí reaktivního formuláře. Nejdříve jsou definovány závislosti a poté následuje komponenta s template, uvnitř které se nachází vstupní pole. Model formuláře je instancí *FormControl* (Angular, 2020).

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';
@Component({
  selector: 'app-reactive-favorite-color',
  template: `
    Favorite Color:
       <input type="text" [formControl]="favoriteColorControl">
  `
})
export class FavoriteColorComponent {
  favoriteColorControl = new FormControl('');
}
```

Ukázka kódu 36: Reaktivní formulář Angular

V nadcházející ukázce je stejná komponenta jako v předchozí ukázce se vstupním polem, ale s využitím šablonou řízeného formuláře. Nejdříve je opět definován dekorátor pro komponentu, poté komponenta samotná a template obsahující vstupní pole.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-template-favorite-color',
  template: `
    Favorite Color:
       <input type="text" [(ngModel)]="favoriteColor">
  `
})export class FavoriteColorComponent {
  favoriteColor = '';
}
```

Ukázka kódu 37: Řízený formulář Angular

3.7.2 **React**

V Reactu existují také dva přístupy, kontrolované komponenty a nekontrolované komponenty. V kontrolované komponentě jsou data z formuláře zpracovávána komponentou Reactu, která udržuje vstupní hodnoty a poté je aktualizuje pomocí funkce *setState()*. Alternativou jsou nekontrolované komponenty, kde jsou data z formuláře zpracovávána pomocí samotného DOM. Ve většině případů je doporučováno implementovat formuláře pomocí kontrolovaných komponent, a proto je tato podkapitola věnována pouze tomuto přístupu.

V HTML prvky formuláře jako <*input*>, <*textarea*> a <*select*> obvykle udržují svůj vlastní stav a aktualizují jej na základě vstupu uživatele. V Reactu je tento proměnný stav obvykle udržován ve vlastnosti *"state"* dané komponenty a aktualizován pouze pomocí funkce *setState()*. Tyto dva přístupy lze zkombinovat tím, že React stav bude jediným takzvaným "zdrojem pravdy". Poté komponenta reactu vykreslující formulář také řídí to, co se ve formuláři stane při následujícím uživatelském vstupu. Vstupní prvek formuláře, jehož hodnota je tímto způsobem řízena pomocí Reactu, se nazývá "kontrolovaná komponenta" (React, 2020).

V následující ukázce je možné vidět kontrolovanou komponentu, která stejně jako v Angularu vykreslí vstupní pole. V ukázce je také možné vidět funkci *setState()*, která je zavolána v případě změny ve vstupním poli formuláře.

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  render() {
    return (
      <form>
        <label>
          Favorite Color:
          <input type="text" value={this.state.value}
          onChange={this.handleChange} />
        </label>
      </form>
    );
  }
}
```

Ukázka kódu 38: Formulář React

Z ukázky je vidět, že hodnota atributu je nastavena na element formuláře. To znamená, že zobrazená hodnota bude vždy "*this.state.value*", čímž učiní z Reactového stavu "zdroj pravdy". A protože metoda "*handleChange*" je spuštěna při každém stisknutí klávesy pro aktualizaci React stavu, zobrazená hodnota je aktualizovaná zároveň s tím, jak uživatel píše.

3.7.3 **Vue**

Ve Vue se pro formuláře využívá atributu *v-model* k vytvoření obousměrného bindování dat. *V-model* automaticky vybírá správný způsob aktualizace prvku na základě typu vstupu, stačí ho tedy jenom přidat ke vstupnímu poli, jako tomu je v následující ukázce (Vue, 2020).

```
<input v-model="favoriteColor">
```

Ukázka kódu 39: Atribut v-model Vue

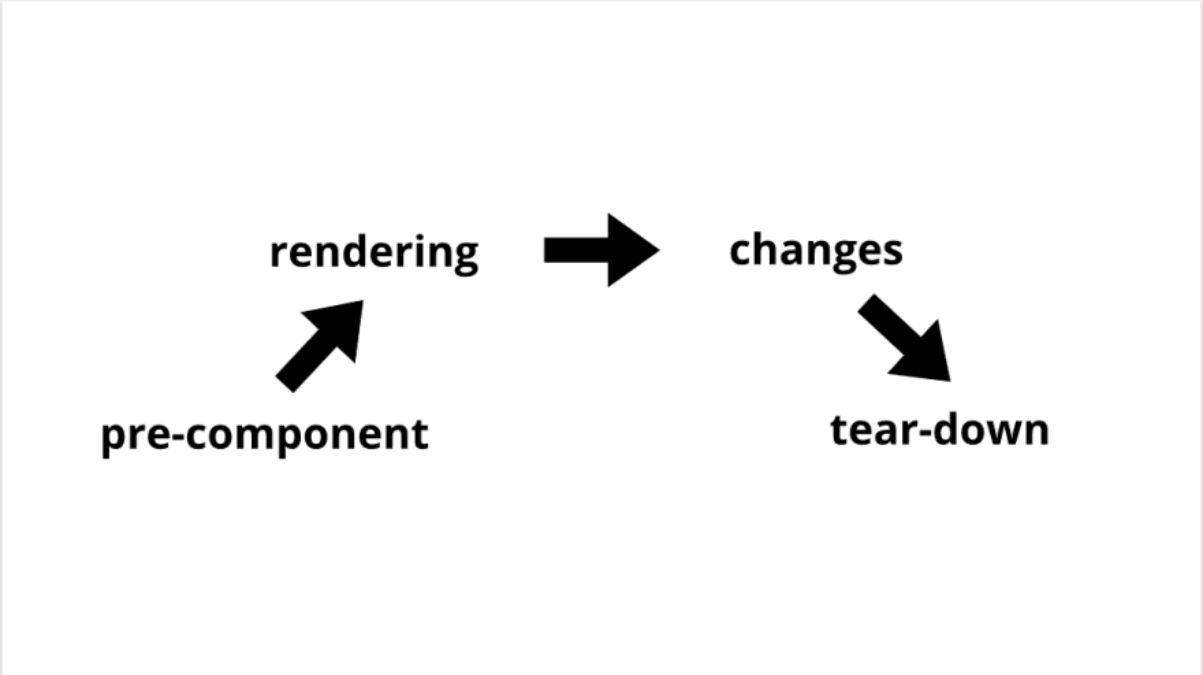
V ukázce 40 je demonstrováno zobrazení vstupu z předchozí ukázky.

```
<p>Favorite color is: {{ favoriteColor }}</p>
```

Ukázka kódu 40: Zobrazení vstupu Vue

3.8 Metody životního cyklu komponenty

Životní cykly komponenty jsou různé fáze, kterými daná komponenta prochází od jejího stvoření až po zničení (odstranění). Tento životní cyklus je spravovaný frameworkem, který dané komponenty vytváří, přidává a odebírá je z DOM, a aktualizuje DOM, kdykoliv se změní stav komponenty. Životní cykly komponenty můžeme rozdělit do čtyř hlavních fází viz obrázek 8.



Obrázek 8: Životní cyklus komponenty
Zdroj: Barth, 2019

První možnou fází cyklu je, že komponenta ještě neexistuje. Tedy to znamená, že komponenta byla zavolána, ale na obrazovce se ještě nic neobjevilo. Je to tedy taková fáze cyklu, která provede něco ještě předtím, než dojde k vykreslování komponenty. Příkladem může být kontrola aktuálního uživatele a jeho práv, než se pokusíme zobrazit nějaké informace.

Další fází cyklu je, že komponenta již existuje a je v procesu vykreslování. Toto je jedna z nejčastěji využívaných částí cyklu, během které chceme spustit určitý kód. V této fázi dochází ke spuštění různých operací a volání jako například asynchronní API volání.

Po vykreslení komponenty se dostáváme do další fáze cyklu, kde je možné reagovat na různé změny a interakce. Je důležité zmínit, že to může, ale také nemusí mít za následek překreslení komponenty. Kódem, který chceme v této fázi spustit může být například ověření formuláře.

A nakonec fáze zahození komponenty, kam se můžeme dostat například opuštěním stránky nebo zavřením modálního okna atd. V této fázi cyklu existuje také řada věcí, které můžeme chtít provést, jako například odhlášení se od datových streamů, nebo provést výzvu o neuložených změnách.

Do teď jsme mluvili pouze obecných fázích a zdaleka to nejsou jediné fáze cyklu komponenty, frameworky tento cyklus rozdělily do tzv. metod životního cyklu. Přičemž každý framework má své vlastní metody, které se volají v určitý moment životního cyklu komponenty. Metoda životního cyklu je tedy JavaScriptová metoda, která je spuštěna v určité fázi životního cyklu komponenty.

V tabulce Tabulka 3 jsou příklady metod vybraných frameworků, které zhruba odpovídají daným fázím obecného životního cyklu. Nelze je však přímo porovnávat, protože každý framework řeší vytváření a vykreslování komponent trochu jiným způsobem. Ale pokud se na metody z tabulky 3 podíváme obecně, jsou si velmi podobné (Barth, 2019).

Tabulka 3: Příklady metod životního cyklu komponenty

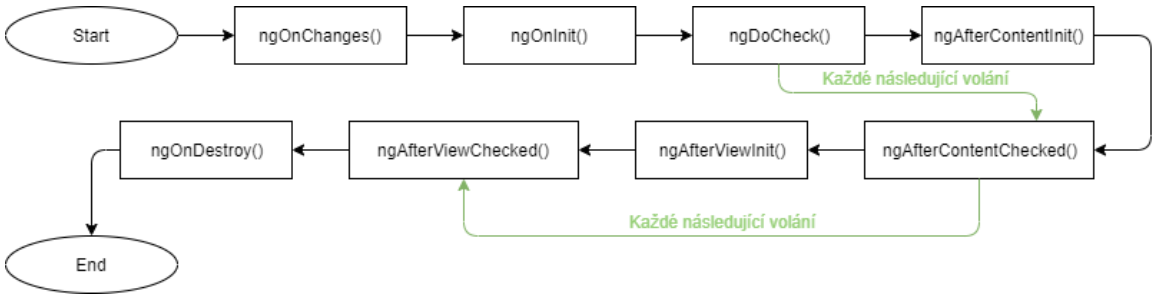
	Pre-component	Rendering	Changes	Tear-down
Angular	-	ngOnInit	ngOnChanges	ngOnDestroy
React	constructor	componentDidMount	componentDidUpdate	componentWillUnmount
Vue	created	mounted	updated	destroyed

Zdroj: vlastní zpracování podle Barth, 2019

V následujících podkapitolách se tedy blíže podíváme na jednotlivé metody vybraných frameworků a jejich posloupnost.

3.8.1 **Angular**

Metody životního cyklu mohou být v Angularu využity na instance direktiv a komponent. Po vytvoření komponenty či direktivy pomocí konstruktoru, Angular volá metody životního cyklu v následujícím pořadí a specifických okamžicích viz obrázek 9 (Angular, 2020).



Obrázek 9: Metody životního cyklu Angular
Zdroj: vlastní zpracování podle Angular, 2020

ngOnChanges()

Odpoví v případě, když Angular nastaví či přenastaví datově-vázané vstupní vlastnosti. Metoda přijímá objekt *SimpleChanges* s aktuálními a předchozími hodnotami vlastností. Tato metoda se volá před metodou *ngOnInit()* a kdykoliv dojde ke změně jedné nebo více datově-vázaných vstupních vlastností.

ngOnInit()

Inicializuje komponentu či direktivu poté, co Angular poprvé zobrazí datově-vázané vlastnosti a nastaví vstupní vlastnosti dané komponenty či direktivy. Metoda se volá jednou a to po prvním zavolání metody *ngOnChanges()*.

ngDoCheck()

Detekuje a jedná podle změn, které Angular nedokáže nebo nezjistí sám. Metoda se volá při každém spuštění detekce změn, okamžitě po metodách *ngOnChanges()* a *ngOnInit()* (Angular, 2020).

ngAfterContentInit()

Odpoví poté, co Angular promítne externí obsah do zobrazení komponenty nebo do zobrazení, ve kterém je direktiva. Tato metoda se volá po prvním zavolání metody *ngDoCheck()*.

ngAfterContentChecked()

Odpoví potom, co Angular zkontroluje obsah promítnutý do komponenty či direktivy. Metoda se volá po metodě *ngAfterContentInit()* a po každém následujícím zavolání metody *ngDoCheck()*.

ngAfterViewInit()

Odpoví poté, co Angular inicializuje zobrazení komponenty a podřízené zobrazení či zobrazení, ve kterém je direktiva. Tato metoda je se volá jednou po prvním zavolání metody *ngAfterContentChecked()*.

ngAfterViewChecked()

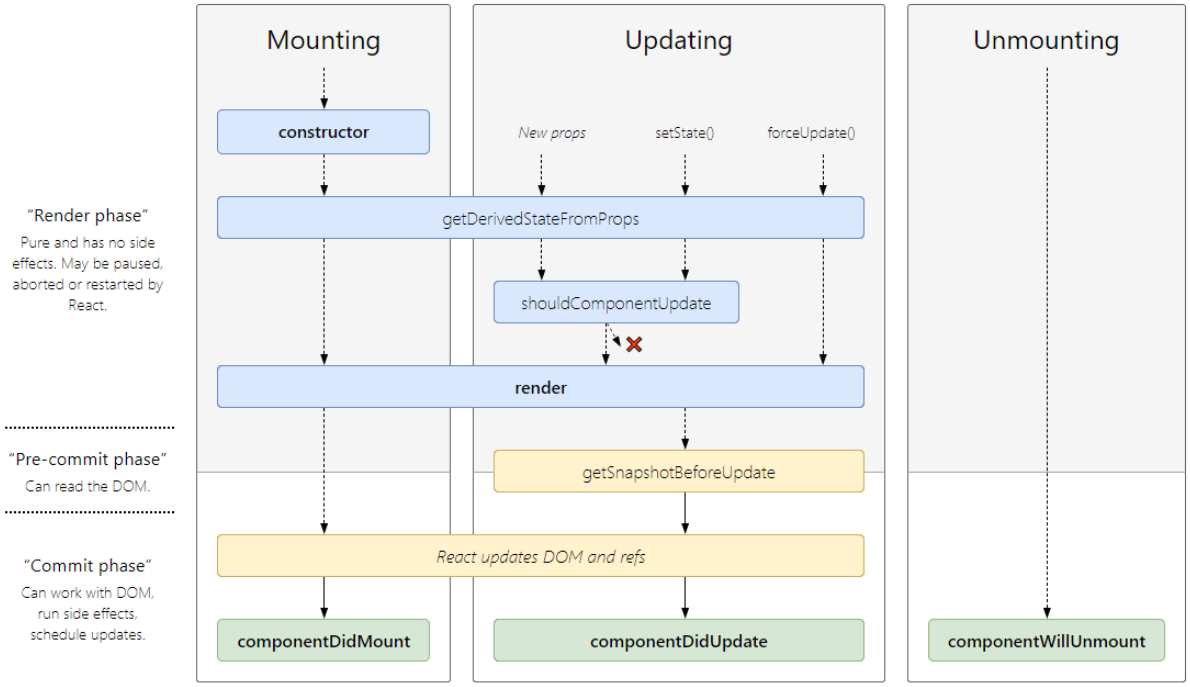
Odpovídá potom, kdy Angular zkontroluje zobrazení komponenty a podřízené zobrazení či zobrazení, ve kterém se direktiva nachází. Metoda se volá po metodě *ngAfterViewInit()* a po každém následujícím volání metody *ngAfterContentChecked()*.

ngOnDestroy()

Provede čištění před tím, než Angular odstraní danou komponentu či direktivu. Tato metoda je zavolána těsně před tím, než Angular zničí danou komponentu či direktivu (Angular, 2020).

3.8.2 **React**

Metody životní cyklus jsou v Reactu rozděleny do několika částí, mounting, updating a unmounting viz obrázek 10.



Obrázek 10: Metody životního cyklu React
Zdroj: Maj, 2020

Mouting

K volání těchto metod dochází při vytváření a vkládání instance komponenty do DOM v následujícím pořadí.

- constructor()
- static getDerivedStateFromProps()
- render()
- componentDidMount()

Aktualizování (Updating)

Aktualizace komponenty může být způsobeno změnou vlastností nebo stavu. V takovém případě je komponenta překreslena a metody jsou volány v následujícím pořadí.

- static getDerivedStateFromProps()
- shouldComponentUpdate
- render()

- getSnapshotBeforeUpdate()
- componentDidUpdate()

Unmounting

Tato metoda je volána, když je komponenta odstraňována z DOM.

- componentWillUnmount()

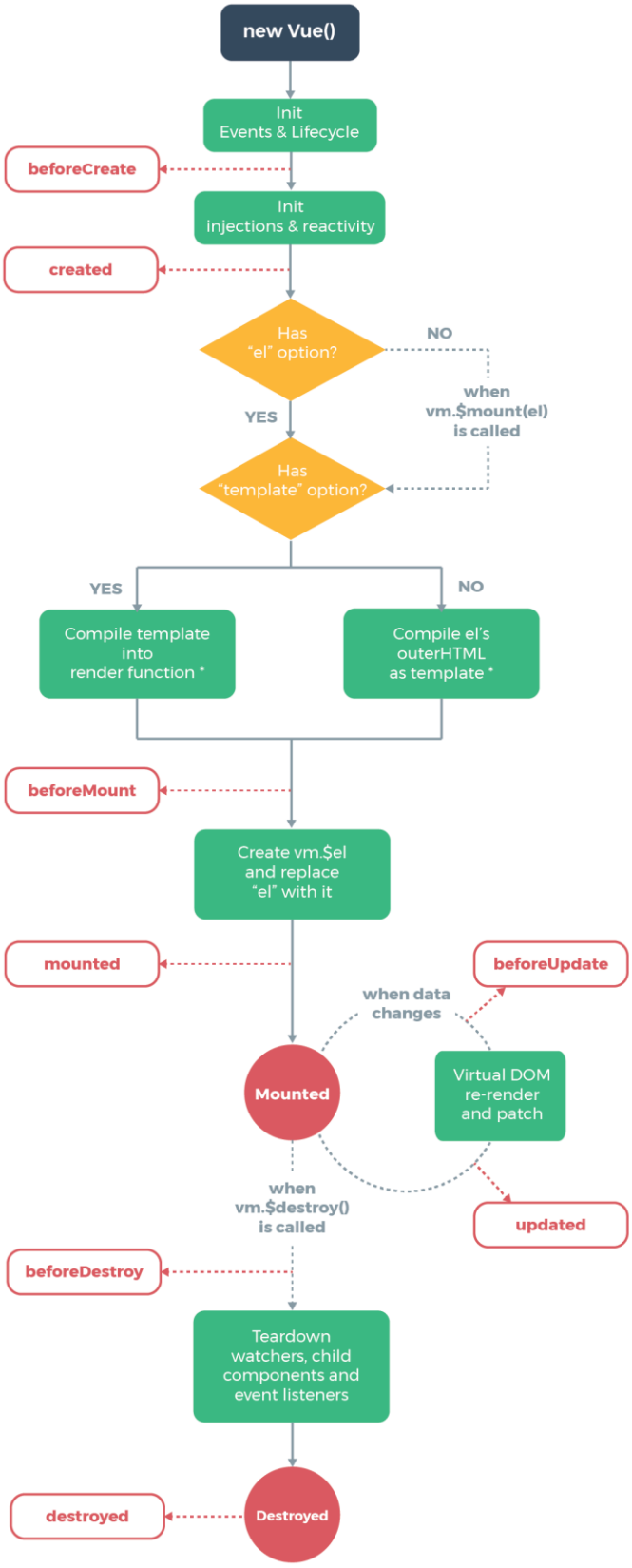
Vypořádání se s chybou (Error Handling)

Tyto metody jsou volány tehdy, když dojde k chybě během vykreslování komponenty, v nějaké metodě životního cyklu, nebo v konstruktoru jakékoliv podřízené komponenty (React, 2020).

- static getDerivedStateFromError()
- componentDidCatch()

3.8.3 **Vue**

Každá Vue instance prochází řadou inicializačních kroků viz obrázek 11.



Obrázek 11: Metody životního cyklu Vue
Zdroj: Vue.js, 2020

beforeCreate

Volá se synchronně ihned po inicializaci instance. Jednoduchý příklad syntaxe.

```
new Vue({
  beforeCreate: function () {
    console.log('Tato metoda se volá před tím, než se vytvoří instance!')
  }
})
```

Ukázka kódu 41: Metoda beforeCreate Vue

created

Volá se po vytvoření instance. Jednoduchý příklad syntaxe.

```
new Vue({
  created: function () {
    console.log('Tato metoda se volá poté, co došlo k vytvoření instance!')
  }
})
```

Ukázka kódu 42: Metoda created Vue

beforeMount

Volá se těsně před tím, než dojde k mountingu: funkce vykreslení bude poprvé zavolána. Jednoduchý příklad syntaxe.

```
new Vue({
  beforeMount: function () {
    console.log('Tato metoda se volá před tím, než dojde k moutingu
instance!')
  }
})
```

Ukázka kódu 43: Metoda beforeMount Vue

mounted

Volá se potom, co dojde k moutingu instance. Tato metoda nezaručuje, že došlo také k moutingu všech podřízených komponent. K počkání na vykreslení celého zobrazení lze použit direktivu *vm.$nextTick* uvnitř metody *mounted*.

```
new Vue({
  mounted: function () {
    this.$nextTick(function () {
      // Kód bude spuštěn až potom, co se vykreslí celé zobrazení
    })
  }
})
```

Ukázka kódu 44: Metoda mounted Vue

beforeUpdate

Volá se vždy, když se změní data, před opětovným vykreslením dat.

updated

Volá se po změně dat, které způsobí opětovné překreslení dat. Tato metoda nezaručuje, že byly překresleny také všechny podřízené komponenty. K počkání na vykreslení celého zobrazení lze použit direktivu *vm.$nextTick* uvnitř metody *updated*.

```
updated: function () {
  this.$nextTick(function () {
    // Kód bude spuštěn až potom,
    // co se překreslí celé zobrazení
  })
}
```

Ukázka kódu 45: Metoda updated Vue

activated

Volá se tehdy, pokud je daná komponenta aktivována.

deactivated

Volá se, pokud je daná komponenta deaktivována.

beforeDestroy

Volá se těsně před zničením instance Vue. V této fázi je instance stále plně funkční.

destroyed

Volá se po zničení Vue instance. Pokud se zavolá tato metoda dojde k rozvázání všech direktiv dané instance.

errorCaptured

Volá se, když je zachycena chyba z jakékoliv podřízené komponenty (Grzybek, 2018).

3.9 Design

Součástí každé webové aplikace či stránky jsou také CSS styly, které se starají o design dané webové stránky. Styly je možné jednotlivým prvkům stránky přiřadit buď staticky nebo dynamicky. V této podkapitole je zobrazeno pouze jednoduché přidaní statických stylů.

3.9.1 **Angular**

V Angularu jsou pro CSS styly vytvořeny vlastní soubory, které jsou zpravidla pojmenovávány stejně jako template, ke které dané styly přísluší. Cesty k těmto souborům jsou pak definovány v komponentách za direktivou "*styleUrls*". V následující ukázce je tedy nejdříve přidána template a potom soubor s CSS styly.

```
@Component({
  templateUrl: './zahlavi.html'
  styleUrls: ['./zahlavi.scss']
```

Ukázka kódu 46: Přidání template a CSS Angular

V následující ukázce je zobrazen soubor *zahlavi.html*, uvedený jako template v komponentě. V tomto souboru je definován element *h1*, který obsahuje *class "header"* (Angular, 2020).

```
<h1 class="header">
    Ahoj z Angularu
</h1>
```

Ukázka kódu 47: Template zahlavi.html Angular

A nyní zbývá jen ukázka 48 se souborem *zahlavi.scss*, ve kterém je definována *class "header"* z minulé ukázky.

```
.header {
    color: red;
}
```

Ukázka kódu 48: CSS soubor zahlavi.scss Angular

3.9.2 **React**

V Reactu existuje mnoho přístupů ke stylizaci aplikace od tradičních preprocesorů (jako v Angularu) až po tzv. CSS v JS. Mezi nejoblíbenější patří css-moduly, stylizované komponenty a aphrodite.

V následující ukázce je nejprve vytvořena proměnná *"header"*, do které je přiřazen daný styl. Na tento styl je poté odkazováno v elementu *h1* (React, 2020).

```
const header = { color: blue };

 render() {
   return (
      <h1 style={header}>
        Ahoj z Reactu
      </h1>
  );
}
```

Ukázka kódu 49: Design React

3.9.3 **Vue**

Komponenty ve Vue lze jednoduše upravit v oddílu style. Pokud má daná značka atribut *scoped*, CSS styly budou aplikovány pouze na prvky aktuální komponenty. V ukázce 50 je tedy nejdříve v oddílu template vytvořen element *h1* obsahující *class "header"*. Poté je do této *class* v oddílu style, přiřazen daný styl (Vue, 2020).

```
<template>
  <h1 class="header">
     Ahoj z Vue
  </h1>
</template>

<script>
  ...
</script>

<style scoped>
  .header {
    color: green
  }
</style>
```

Ukázka kódu 50: Design Vue

3.10 Vložení HTML template

V této podkapitole jsou ukázky, jak je možné vložit HTML template (označováno také jako innerHTML) v jednotlivých frameworcích.

3.10.1 **Angular**

Vložení HTML template v Angularu.

```
<p [innerHTML]="article.content"></p>
```

Ukázka kódu 51: Vložení HTML template Angular

3.10.2 **React**

Vložení HTML template v Reactu.

```
<p dangerouslySetInnerHTML={ { __html: article.content } } />;
```

Ukázka kódu 52: Vložení HTML template v Reactu

3.10.3 **Vue**

Vložení HTML template ve Vue (Grzybek, 2018).

```
<div v-html="article.content"></div>
```

Ukázka kódu 53: Vložení HTML template Vue

4 Implementace vybraných JavaScriptových frameworků

Tato kapitola se věnuje jednotlivým postupům instalace vybraných JavaScriptových frameworků. Z tohoto důvodu je tedy nutné nejdříve specifikovat základní předpoklady, které jsou potřebné k dalšímu postupu.

- Základní znalost HTML a CSS
- Základní znalost jazyků TypeScript a JavaScript
- Základní znalost anglického jazyka
- Základní znalosti tvorby webových aplikací

Všechny projekty v této kapitole využívají npm a TypeScript pro zachování co největší podobnosti jednotlivých implementací.

4.1 Instalace nutných nástrojů

Pro instalaci vybraných frameworků je nutné nejprve nainstalovat Node.js verze minimálně 8.9, doporučováno je však 8.11.0 a vyšší. Na webové stránce nodejs.org lze najít dvě aktuální verze Node.js. Současnou verzi se všemi nejnovějšími funkcemi, a verzi LTS, doporučovanou pro většinu uživatelů, tuto verzi je doporučeno stáhnout a nainstalovat.

Spolu s Node.js je stažen a nainstalován npm, který je také potřebný k dalším krokům vytvoření nového projektu. Zda instalace obou těchto závislostí proběhla v pořádku, lze jednoduše ověřit pomocí zadání příkazů, *node -v* pro Node.js a *npm -v* pro npm, do příkazového řádku. Po zadání příkazu by se v příkazovém řádku měla vypsat aktuální verze daného balíčku viz obrázek 12.

Obrázek 12: Ověření Node.js a npm
Zdroj: vlastní

Dále je vhodné nainstalovat nějaký editor kódu. V této práci je využito editoru Visual Studio Code, který je možné stáhnout na webové adrese https://code.visualstudio.com. Je však možné použít jakýkoliv jiný.

4.2 Angular

Pro založení a následnou správu projektu v Angularu je potřeba nainstalovat Angular CLI. Angular CLI je nástroj využívaný v příkazovém řádku pro inicializaci, vývoj, zakládání a spravování Angular aplikací. Před instalací tohoto nástroje je nejprve nutné nainstalovat Node.js a npm viz podkapitola 4.1 Instalace nutných nástrojů.

4.2.1 Instalace Angular CLI

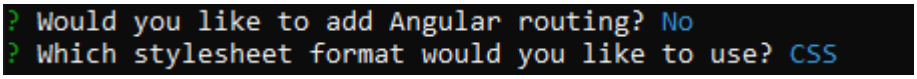
Zadáním příkazu *npm install – g @angular/cli* do příkazové řádky je nainstalován Angluar CLI. Před spuštěním příkazu musí být příkazový řádek otevřen v administrátorském režimu, aby byla dostupná všechna potřebná práva. Ověření instalace je možné provést zadáním příkazu *ng*, který vypíše informační zprávu se všemi dostupným příkazy, pokud instalace proběhla v pořádku (Angular, 2020).

4.2.2 Založení projektu

K novému projektu je vhodné vytvořit novou složku. V tomto postupu je složka pojmenována ProjektAngular a je vytvořena v C:\Uzivatele\Jmeno_Uzivatele příkazem: *mkdir C:\Uzivatele\Jmeno_Uzivatele\ProjektAngular*. Do nové složky je nutné následně přejít, *cd C:\Uzivatele\Jmeno_Uzivatele\ProjektAngular*. V této složce je nyní potřeba, pro založení nového projektu, zadat příkaz: *ng new nejlepsi-angular-projekt*.

Příkaz *ng new,* má jako parametr jméno projektu, v tomto případě nejlepsi-angular-projekt. Jako oddělovač u víceslovných názvů je vhodné použít například pomlčky. Pokud je jméno projektu odděleno mezerami, tak příkaz *ng new* vezme jako název projektu pouze část k první mezeře a zbytek bude ignorovat. Další podmínkou je, že název projektu nesmí obsahovat diakritiku, jinak uvedený příkaz zahlásí chybu.

Po zadání příkazu *ng new nejlepsi-angular-projekt* jsou postupně zobrazeny dvě otázky viz obrázek 13, stačí je obě potvrdit klávesou Enter, čímž budou ponechány ve výchozí variantě (Angular, 2020).



Obrázek 13: Instalace Angular
Zdroj: vlastní

Nyní je nutné přejít do složky, která se vytvořila spolu se založením projektu, *cd nejlepsi-angular-projekt (cd jmeno-projektu)*

Celá cesta v tomto ilustračním příkladu:
C:\Uzivatele\Jmeno_Uzivatele\ProjektAngular\nejlepsi-angular-projekt.

Při zakládání projektu byly vytvořeny nezbytné složky a soubory, které je možné zobrazit zadáním příkazu:

dir /b

- e2e – složka obsahující automatické testy
- node_moodules – složka s nainstalovanými moduly, které ukládá npm
- src – složka se zdrojovými kódy aplikace, podrobnější informace jsou uvedeny níže
- package.json a package-lock.json – konfigurační soubory pro npm
- ostatní soubory – další konfigurační soubory projektu, především pro Angular a Typescript

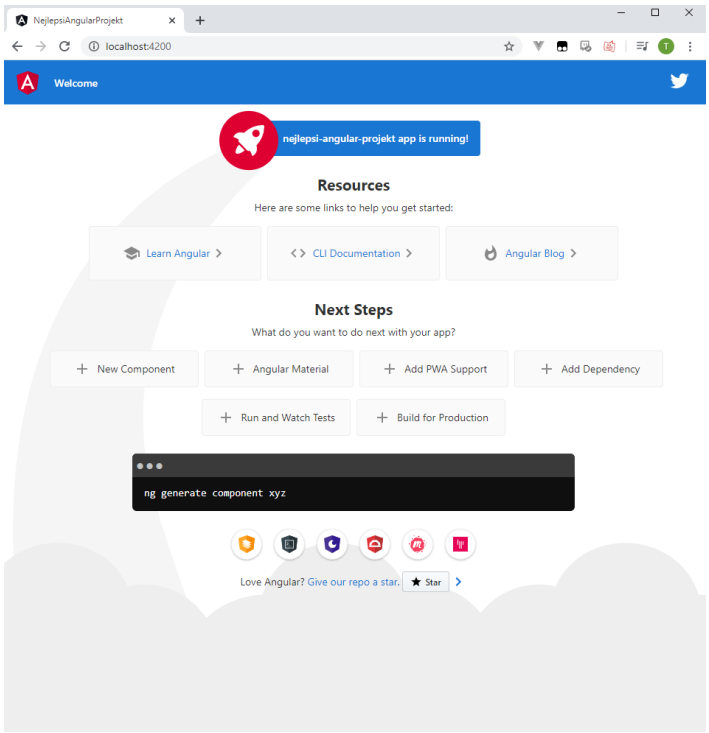
4.2.3 **První spuštění**

Zadáním příkazu: *npm run start* do příkazové řádky dojde po několika vteřinách k nastartování projektu viz obrázek 14.

```
C:\Uzivatele\Jmeno_Uzivatele\ProjektAngular>cd nejlepsi-angular-projekt

C:\Uzivatele\Jmeno_Uzivatele\ProjektAngular\nejlepsi-angular-projekt>npm run start

> nejlepsi-angular-projekt@0.0.0 start C:\Uzivatele\Jmeno_Uzivatele\ProjektAngular\nejlepsi-angular-projekt
> ng serve

0% compiling
Compiling @angular/core : es2015 as esm2015

Compiling @angular/common : es2015 as esm2015

Compiling @angular/platform-browser : es2015 as esm2015

Compiling @angular/platform-browser-dynamic : es2015 as esm2015

chunk {main} main.js, main.js.map (main) 57.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 9.74 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.7 MB [initial] [rendered]
Date: 2020-03-28T13:55:50.909Z - Hash: 51f50880cb7b3cc8056c - Time: 12090ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.

Date: 2020-03-28T13:55:51.625Z - Hash: 51f50880cb7b3cc8056c
5 unchanged chunks

Time: 358ms
: Compiled successfully.
```

Obrázek 14: Start projektu Angular
Zdroj: vlastní

Po zadání webové adresy http://localhost:4200/ do webového prohlížeče je zobrazena úvodní stránka nově vytvořeného projektu viz obrázek 15 (Angular, 2020).



Obrázek 15: Úvodní stránka nového projektu Angular
Zdroj: vlastní zpracování podle Angular, 2020

4.2.4 **Úprava projektu**

Projekt je nyní potřebné otevřít v editoru kódu, například Visual Studio Code viz podkapitola 4.1 Instalace nutných nástrojů.

Úvodní stránka nového projektu Angular

Po otevření projektu v editoru je vidět struktura projektu, jak byla uvedena v podkapitole 4.2.2 Založení projektu. Nyní je tedy potřebné přejít do složky src, která obsahuje zdrojové kódy projektu. Ve složce src se nachází:

- app – hlavní složka celé aplikace, obsahující komponenty
- assets – obsahuje statické soubory, například obrázky, dokumenty
- environments – obsahuje různá prostředí pro vývoj
- index.html – vstupní bod aplikace
- main.ts – inicializuje novou Angular instanci
- styles.css – obsahuje globální CSS styly, které nesouvisí s jednotlivými komponentami
- polyfills.ts, test.ts – ostatní soubory, určené k další konfiguraci

Ze složky src je nyní potřeba přejít do složky app a zde otevřít soubor *app.component.ts*, který by měl vypadat nějak takto:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'nejlepsi-angular-projekt';
}
```

Ukázka kódu 54: Soubor app.components.ts

Soubor je upraven následujícím způsobem. Proměnná *title* je smazána a je vytvořená nová proměnná *zprava*, do které je přiřazen řetězec „Toto je upravená komponenta v Angularu!“. Soubor po úpravách vypadá takto:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  zprava = 'Toto je upravená komponenta v Angularu!';
}
```

Ukázka kódu 55: Upravený soubor app.components.ts

Nyní je potřeba otevřít soubor *app.component.html*, smazat veškerý obsah tohoto souboru a následně přidat jednoduchý *div*, který zobrazí proměnnou *zprava*.

Poslední změna je provedena v souboru *app.component.css*, přidáním následujících řádků.

```
.stylZpravy {
    color: orange;
    font-size: 25px;
    text-align: center;
}
```

Ukázka kódu 56: Soubor app.component.css

Tímto je textu nastavena barva, velikost a zarovnání. Po uložení výše zmíněných souboru by webová aplikace měla vypadat jako na obrázku 16.

Obrázek 16: Upravená komponenta Angular
Zdroj: vlastní

4.3 React

Jedním z největším rozdílu při implementaci Reactu oproti Angularu (i Vue), je v tom, že v Reactu není nutné před samotným založením projektů instalovat speciální CLI pro práci s daným frameworkem. Před založením projektu je ale nutné nainstalovat Node.js a npm jako tomu je pro všechny vybrané frameworky viz podkapitola 4.1 Instalace nutných nástrojů.

4.3.1 **Založení projektu**

Pro založení projektu je nejprve nutné otevřít terminál a do něj zadat tento příkaz: *mkdir C:\Uzivatele\Jmeno_Uzivatele\ProjektReact*

pomocí kterého je v C:\Uzivatele\Jmeno_Uzivatele vytvořena nová složka s názvem ProjektReact. Do této složky bude nový projekt vytvořen, proto je potřeba do této složky nejprve přejít pomocí příkazu:

cd C:\Uzivatele\Jmeno_Uzivatele\ProjektReact

V této složce je nyní potřeba, pro založení nového projektu, do terminálu zadat příkaz:

npx create-react-app nejlepsi-react-projekt --template typescript --use-npm

K příkazu *npx create-react-app* jsou v tomto postupu přidány tři argumenty. Prvním argumentem je jméno projektu, *nejlepsi-react-projekt*. Druhým argumentem je *--template typecsript*, tento argument způsobí, že nově vytvořený projekt bude v Typescriptu a posledním argumentem je *--use-npm*, pomocí kterého je možné nově vytvořený projekt spouštět přes npm příkazy (React, 2020).

Po vytvoření projektu, je nutné přejít do složky s projektem:

cd nejlepsi-react-projekt

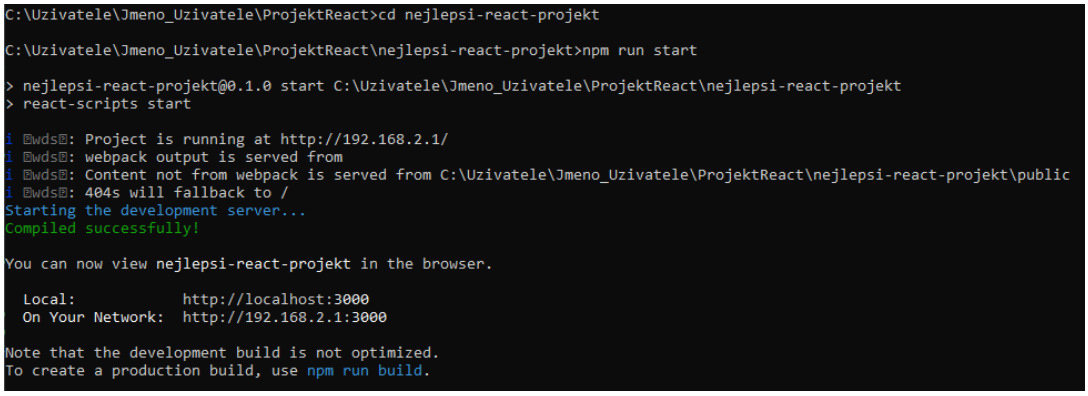
Při zakládání projektu byly vytvořeny nezbytné složky a soubory, které je možné zobrazit zadáním příkazu:

dir /b

- node_modules – složka, která obsahuje balíčky nainstalované pomocí npm
- public – složka se statickým obsahem jako například obrázky, obsahuje také index.html
- src – složka obsahující zdrojové kódy aplikace
- package.jsou a package-lock.json – konfigurační soubory k npm
- tsconfig.json – konfigurační soubor pro TypeScript

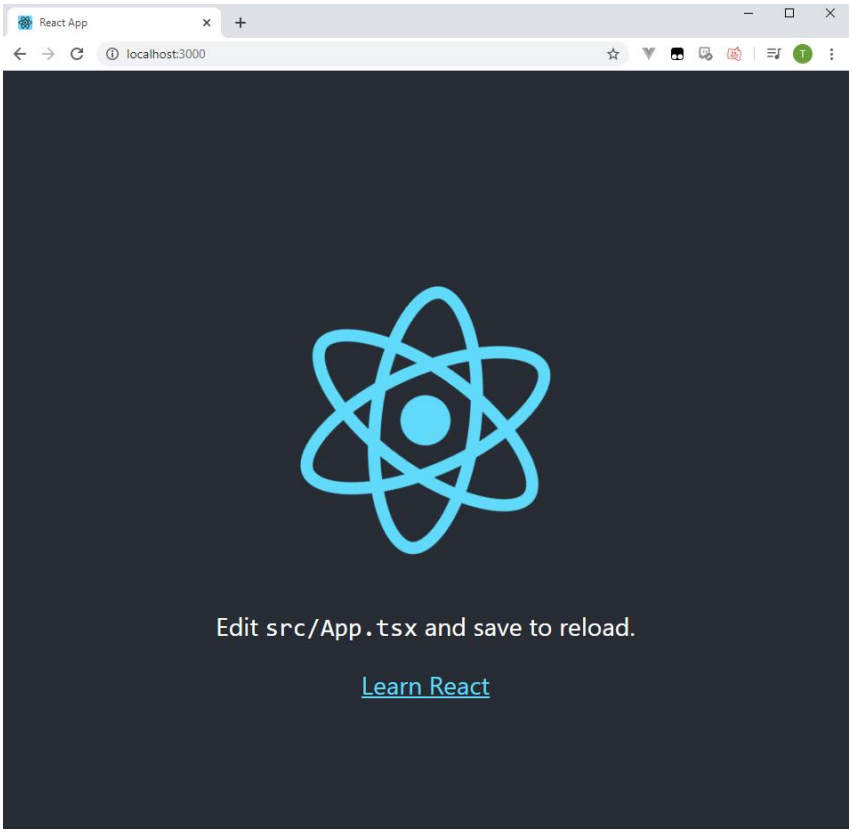
4.3.2 **První spuštění**

Zadáním příkazu *npm run start* do příkazové řádky dojde po několika vteřinách k nastartování projektu viz obrázek 17.



Obrázek 17: Start projektu React
Zdroj: vlastní

Nyní je potřeba přejít na webovou adresu http://localhost:3000/, kde je zobrazena úvodní stránka nově vytvořeného projektu viz obrázek 18.



Obrázek 18: Úvodní stránka nového projektu React
Zdroj: vlastní zpracování podle React, 2020

4.3.3 **Úprava projektu**

Projekt je nyní potřebné otevřít v editoru kódu, například Visual Studio Code viz podkapitola 4.1 Instalace nutných nástrojů. Po otevření projektu v editoru je vidět struktura projektu, jak byla uvedena v podkapitole 4.3.1 Založení projektu. Nyní je tedy potřebné přejít do složky src a zde otevřít soubor *App.tsx*, který by měl vypadat jako v následující ukázce.

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p> Edit <code>src/App.tsx</code> and save to reload. </p>
        <a className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer" >
          Learn React
        </a>
      </header>
    </div>
  );
}
export default App;
```

Ukázka kódu 57: Soubor App.tsx

Soubor je potřeba upravit následujícím způsobem. Do souboru je nejdříve přidána proměnná *zprava* do které je přiřazen řetězec „Toto je upravená komponenta v Reactu!“. Dále je vše ve funkci return smazáno a nahrazeno jednoduchým *divem*, který zobrazí proměnnou *zprava*. Soubor po úpravách:

```
import React from 'react';
import './App.css';

function App() {
  var zprava = 'Toto je upravená komponenta v Reactu!'
  return (
    <div className="stylZpravy">
      {zprava}
    </div>
  );
}
export default App;
```

Ukázka kódu 58: Upravený soubor App.tsx

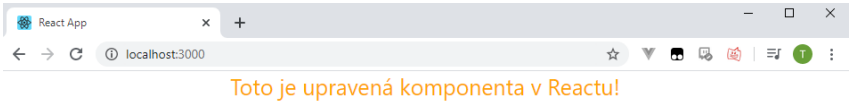
Nyní je ještě potřeba upravit styl dané zprávy, proto je nutné otevřít soubor *App.css*, smazat veškerý obsah tohoto souboru a nahradit ho následujícími řádky.

```
.stylZpravy {
  color: orange;
  font-size: 25px;
  text-align: center;
}
```

Ukázka kódu 59: Soubor App.css

Tímto opět dojde k nastavení barvy, velikosti a zarovnání textu stejně jako tomu bylo v Angularu.

Po uložení souborů je potřeba znovu přejít na webovou adresu http://localhost:3000. Pokud vše proběhlo v pořádku, měla by stránka nyní vypadat jako na obrázku 59 (React, 2020).



Obrázek 19: Upravená komponenta React
Zdroj: vlastní

4.4 Vue.js

Pro založení a následnou správu projektu ve Vue je potřeba nainstalovat Vue CLI. Vue CLI je komplexní nástroj pro rychlý vývoj ve Vue.js. Využívá se v příkazovém řádku a nabízí mnoho výhod, jako je interaktivní zakládání a spravování projektů, bohatou kolekci oficiálních pluginů, které integrují užitečné nástroje pro frontend vývoj, a také grafické uživatelské rozhraní k vytváření a spravování Vue.js projektů.

Vue CLI se skládá ze tří základních komponent. První komponentou je CLI neboli Command Line Interface, rozhraní příkazového řádku. CLI je globálně nainstalovaný balíček, který umožňuje používání vue příkazů v příkazovém řádku. Například *vue create* k rychlému vytvoření konstrukce nového projektu nebo *vue ui* ke správě projektu pomocí grafického uživatelského rozhraní.

Druhou komponentou je CLI service. Jedná se o npm balíček, který je nainstalován do každého projektu vytvořeného pomocí Vue CLI. CLI Service obsahuje:

- základní službu, která se stará o ostatní CLI Pluginy
- vnitřní webpack nastavení, které je optimalizováno pro většinu aplikací
- základní příkazy jako *serve*, *build* a *inspect*

A poslední komponentou jsou CLI Plugins. CLI Plugins jsou nejrůznější npm balíčky, které nabízejí volitelné funkce do Vue CLI projektů, jako například převod zdrojového kódu do Typescriptu (tzv. transpilace), formátování pomocí ESLint nebo testování. Pluginy jsou jednoduše rozpoznatelné pomocí toho, že jejich jména začínají buď *@vue/cli-plugin-* pro officiální pluginy nebo *vue-cli-plugin-* pro neoficiální tzv. komunitní pluginy.

Pluginy mohou být součástí projektu od samého začátku, tedy že jsou přidány již při instalaci projektu, nebo mohou být přidány i později. Seznam všech nainstalovaných pluginů je možné najít v souboru package.json (Vue.js, 2020).

4.4.1 **Instalace Vue CLI**

Po nainstalování všech nutných předpokladů, viz podkapitola 4.1 Instalace nutných nástrojů, přichází instalace Vue CLI balíčku pomocí zadání příkazu: *npm install -g @vue/cli* do příkazového řádku.

Před spuštěním příkazu musí být příkazový řádek otevřen v administrátorském režimu, aby byla dostupná všechna potřebná práva.

Po nainstalování Vue CLI balíčku je vhodné zadat do příkazového řádku příkaz *vue*, který by měl vypsat informační zprávu se všemi dostupným příkazy, pro ověření, že instalace proběhla v pořádku (Vue.js, 2020).

4.4.2 **Založení projektu**

Nejprve je vhodné vytvořit novou složku s názvem ProjektVue, do které bude následně nový projekt vytvořen. V tomto ilustrativním příkladu je složka vytvořena ve složce C:\Uzivatele\Jmeno_Uzivatele. Složku lze vytvořit zadáním příkazu:

mkdir C:\Uzivatele\Jmeno_Uzivatele\ProjektVue

do příkazové řádky. Po zadání tohoto příkazu, by měla být vytvořena nová složka, do které je následně nutné přejít, nejlépe pomocí příkazu:

cd C:\Uzivatele\Jmeno_Uzivatele\ProjektVue

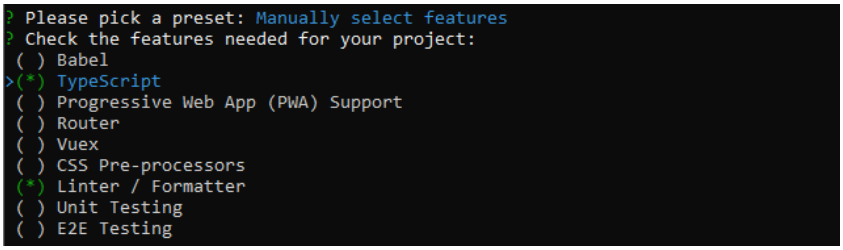
V této složce je nyní potřeba, pro založení nového projektu, zadat příkaz:

vue create nejlepsi-vue-projekt

Tento příkaz se skládá ze dvou částí. První částí jsou klíčová slova pro vytvoření nového projektu *vue create*. Druhá část, v tomto případě *nejlepsi-vue-projekt*, je povinným argumentem a označuje jméno projektu, v případě víceslovného názvu je vhodné jednotlivá slova oddělit pomlčkami, jako je tomu v tomto případě, případně lze použít například Velbloudí notaci. Mezery jsou jako oddělovač nevhodné, protože příkaz *vue create* vezme

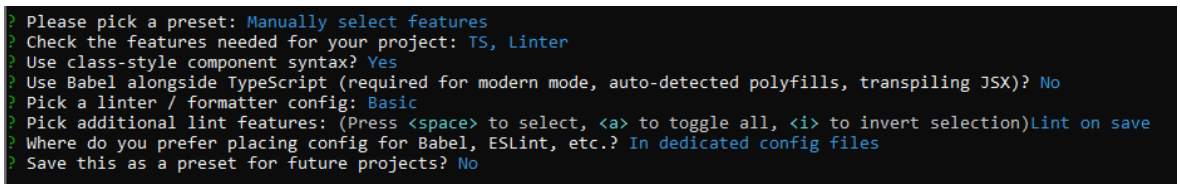
v potaz pouze první slovo jako název projektu a zbytek je ignorován. Další podmínkou je, že název projektu nesmí obsahovat diakritiku, jinak uvedený příkaz zahlásí chybu.

Po zadání příkazu *vue create nejlepsi-vue-projekt* se objeví možnosti nastavení projektu. První možností je základní, která využívá funkce babel a eslint. Druhá možnost nabízí vybrat funkce ručně. V tomto postupu je využito druhé možnosti, po vybrání je zobrazen seznam jednotlivých funkcí viz obrázek 20. Popis těchto funkcí je vypsán v podkapitole 4.4.5 Výběr funkcí Vue (Vue.js, 2020).



Obrázek 20: Výběr funkcí Vue
Zdroj: vlastní

Pro tento ukázkový projekt je zvolen TypeScript a Linter / Formatter. Ukázkový kód je pak přehlednější. Následuje sada otázek, u všech je ponechána výchozí varianta, která se zvolí pouze potvrzením dotazu klávesou Enter. Otázky jsou důležité pro pokročilé vývojáře, kteří vědí, co si zvolit. Nakonec by nastavení projektu mělo vypadat jako na obrázku 21.[1]



Obrázek 21: Výběr funkcí 2 Vue
Zdroj: vlastní

Po založení projektu, je nutné přejít do složky, která byla vytvořena, nejjednodušeji příkazem: *cd nejlepsi-vue-projekt (cd jmeno-projektu).*

[1] Založení nového projektu lze udělat také pomocí webového rozhraní, stačí do příkazového řádku zadat příkaz *vue ui*. Po zadaní tohoto příkazu se otevře okno webového prohlížeče, kde je možné pomocí průvodce založit nový projekt.

Celá cesta v tomto ilustračním příkladu: C:\Uzivatele\Jmeno_Uzivatele\ProjektVue\ nejlepsi-vue-projekt.

Při zakládání projektu byly vytvořeny nezbytné složky a soubory, které je možné zobrazit zadáním příkazu:

dir /b

- node_modules – je složka obsahující balíčky nainstalované přes npm
- public – obsahuje statický obsah včetně index.html, jako vstupní bod aplikace
- src – zdrojové kódy k aplikaci, podrobnější vysvětlení je uvedeno dále v tomto návodu
- package.json a package-lock.json - jsou konfigurační soubory k npm.
- .eslintrc, .browserslistrc, tsconfig.json – jsou jednotlivé konfigurace k nainstalovaným pluginům v rámci nového projektu, mohou být odlišné podle vybrané konfigurace během instalace projektu. Tyto soubory ponecháme ve výchozí variantě.

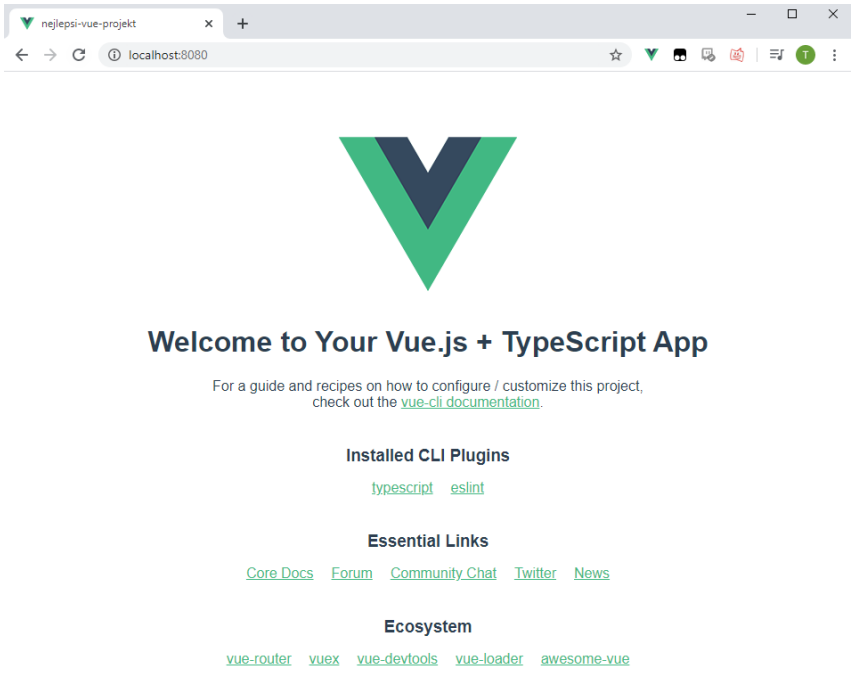
4.4.3 **První spuštění**

Zadáním příkazu *npm run serve*, do příkazové řádky dojde po několika vteřinách k nastartování projektu na webové adrese, která je také zobrazena v příkazovém řádku viz obrázek 22.

```
C:\Uzivatele\Jmeno_Uzivatele\ProjektVue>cd nejlepsi-vue-projekt

C:\Uzivatele\Jmeno_Uzivatele\ProjektVue\nejlepsi-vue-projekt>npm run serve

> nejlepsi-vue-projekt@0.1.0 serve C:\Uzivatele\Jmeno_Uzivatele\ProjektVue\nejlepsi-vue-projekt
> vue-cli-service serve

 INFO  Starting development server...
Starting type checking service...
Using 1 worker with 2048MB memory limit
98% after emitting CopyPlugin

 DONE  Compiled successfully in 2166ms

Type checking in progress...

  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.0.106:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.

No type errors found
Version: typescript 3.7.5
Time: 2238ms
```

Obrázek 22: Start projektu Vue
Zdroj: vlastní

Tuto webovou adresu, tedy http://localhost:8080/, je potřeba překopírovat či přepsat do webového prohlížeče, kde se zobrazí úvodní stránka nově vytvořeného projektu viz obrázek 23.



Obrázek 23: Úvodní stránka nového projektu
Zdroj: vlastní zpracování podle Vue, 2020

4.4.4 **Úprava projektu**

Projekt je nyní potřebné otevřít v editoru kódu, například Visual Studio Code viz podkapitola 4.1 Instalace nutných nástrojů.

Po otevření projektu v editoru je vidět struktura projektu, jak byla uvedena v podkapitole 4.4.2 Založení projektu. Nyní je tedy potřebné přejít do složky src, která obsahuje zdrojové kódy projektu. Ve složce src se nachází:

- assets - obsahuje statické soubory, například obrázky, dokumenty
    - logo Vue.js
- components – obsahuje zdrojové kódy pro Vue.js
    - nyní obsahuje pouze komponentu HelloWorld.vue
- App.vue – hlavní komponenta, obsahující další komponenty
- main.ts - inicializuje novou Vue instanci
- shims-vue.d.ts. – umožňuje vývojovému prostředí pochopit příponu souborů .vue
- shims-tsx.d.ts – umožňuje použití přípony .tsx

Zde je potřeba otevřít soubor *App.vue*, který by měl vypadat nějak takto:

```
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js + TypeScript App"/>
  </div>
</template>
<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';
import HelloWorld from './components/HelloWorld.vue';

@Component({
  components: {
    HelloWorld,
  },
})
export default class App extends Vue {}
</script>
<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

Ukázka kódu 60: Soubor App.vue

Tento soubor je potřeba upravit následujícím způsobem. Vše mezi tagy *template*, *script* a *style* je smazáno. V oddílu *template* je vytvořen jednoduchý *div*, který zobrazí vlastnost *zprava*. V oddílu *script* je definována jednoduchá vlastnost komponenty, které je přiřazený řetězec „Toto je upravená komponenta ve Vue!“. V oddílu *style* je nastavena barva, velikost a zarovnání textu. Takto by měl vypadat uvedený soubor po úpravách:

```
<template>
  <div class="stylZpravy">
    {{ zprava }}
  </div>
</template>
<script lang="ts">
export default {
  props: {
    zprava: {
      type: String,
      default: 'Toto je upravena komponenta ve Vue!',
    }
  },
};
</script>
<style>
.stylZpravy {
  color: orange;
  font-size: 25px;
  text-align: center;
}
</style>
```

Ukázka kódu 61: Upravený soubor App.vue

Nyní je potřeba soubor uložit a přejít na webovou adresu http://localhost:8080/. Pokud vše proběhlo v pořádku, měla by stránka nyní vypadat jako na obrázku 24.

Obrázek 24: Upravená komponenta Vue
Zdroj: vlastní

4.4.5 Výběr funkcí Vue

Tato podkapitola složí jako doplněk pro podkapitolu 4.4.2 Založení projektu.

Babel

Babel je sada nástrojů, která se používá především k převodu kódu ECMAScript 2015+ na zpětně kompatibilní verzi JavaScriptu pro současné a starší webové prohlížeče nebo prostředí (Babel, 2020).

Progressive Web App (PWA) Support

Progresivní webové aplikace jsou webové aplikace využívající moderní webová API k tvorbě webových aplikací napříč platformami. Tyto aplikace fungují prakticky všude a přináší sebou některé výhody nativních aplikací (MDN, 2020).

Vuex

Vuex slouží jako state management pattern (model správy stavu) a zároveň knihovna pro aplikace vytvořené pomocí Vue.js. Jedná se o centralizované uložiště pro všechny komponenty v aplikaci (Vue.js, 2020).

CSS Pre-processors

CSS preprocessor je program, který vygeneruje CSS (vysvětlit CSS) ze své unikátní syntaxe. Nejznámější jsou Sass, LESS (MDN, 2020).

Unit Testing

Unit testing je stupeň softwarového testování, kde jsou otestované nezávislé jednotky (units), komponenty daného softwaru. Unit je nejmenší testu schopná část, jakéhokoliv softwaru (ISTQB, 2020).

E2E Testing

End-To-End testing je metoda softwarového testování, která má za úkol otestovat aplikaci od začátku do konce. Cílem tohoto testování je simulace reálného scénáře proti dané aplikaci (Software Testing Help, 2020).

5 Použití frameworku na vybraném projektu

Tato kapitola je věnována použití vybraného frameworku na konkrétním projektu. Začátek kapitoly obsahuje nejdříve představení vybraného projektu, popis jeho funkcionality, uživatelského rozhraní a využívaných technologií. Dále je popsán výběr frameworku na základě několika kritérií. Poté následuje definování cílů a důvodů refaktoringu projektu a konec kapitoly se zaměřuje na zhodnocení.

5.1 Scripts & Forms

Dříve než dojde na samotnou implementaci vybraného frameworku na určitém projektu, je nutné nejdříve daný projekt představit. Projekt, v tomto případě webová aplikace, nese název Scripts & Forms a je zastřešena společností 2Ring, spol. s r.o. 2Ring působí v oblasti služeb pro kontaktní centra a IP telefonii od roku 2001. Tato firma je jednou z předních firem ve svém oboru a od roku 2011 je držitelem statusu „Cisco Preferred Solution Partner“.

Scripts & Forms provádí agenty kontaktních center skrze jejich rozhovory se zákazníky. Umožňují sběr dat a zaznamenávání strukturovaných informací o klientech a jejich potřebách. Agent prochází se zákazníkem formulář a odpovídá na hierarchicky strukturované otázky výběrem jedné z možností nebo pomocí volného textového pole. Tato aplikace, respektive řešení je součástí rozšířeného (a prémiového) balíčku 2Ring Gadgets pro Cisco Finesse.

V této práci je aplikace brána jako samostatná jednotka a záměrně zjednodušena, aby nedocházelo ke zbytečnému zveličování aplikace, neboť cílem této kapitoly je demonstrovat implementaci vybraného JavaScriptového frameworku na vybraném projektu. I z tohoto důvodu je všechna ostatní logika nesouvisející s danou implementací odstíněna, a to včetně závislostí, které vyplývají z balíčku 2Ring Gadgets.

5.1.1 **Funkcionalita**

Funkcionalita vybrané webové aplikace není nijak složitá. Pokud začneme se vstupy do této aplikace, zjistíme, že existují pouze dva. Prvním vstupem je URL adresa obsahující:

- jméno dotazníku
- identifikátor agenta
- identifikátor zákazníka
- identifikátor hovoru

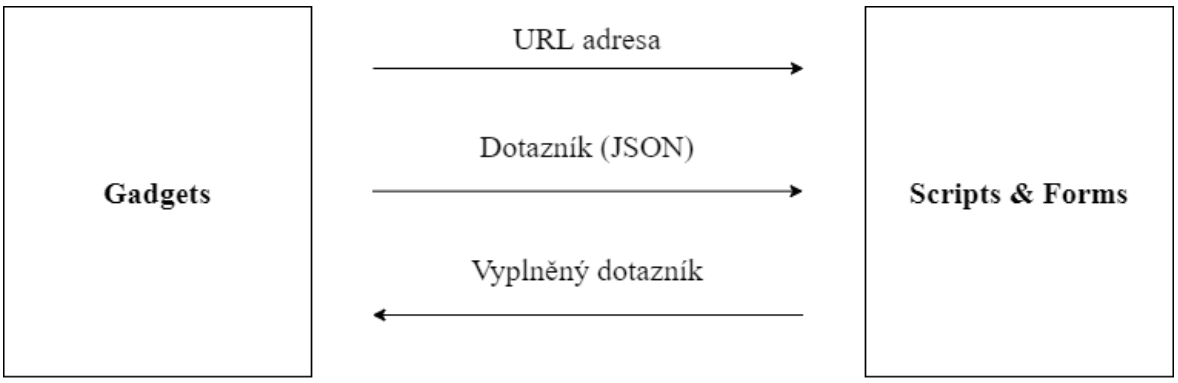
Druhým vstupem je samotný dotazník ve formátu JSON obsahující veškeré otázky a odpovědi.

Jedna otázka s příslušnými odpověďmi odpovídá zpravidla jedné kartě dotazníku, přičemž tyto karty mohou nabývat několika různých podob podle typu odpovědi na danou otázku. Pro lepší pochopení je uveden jednoduchý příklad. Pokud je otázka například „Máte domácího mazlíčka?“ s odpověďmi „Ano“ a „Ne“, daná karta bude typu „Single Choice“ neboli otázka s jednou možnou odpovědí. Pokud by však byla otázka „Jakého máte domácího mazlíčka?“ odpovědí již může být více a daná karta bude typu „Multi Choice“, otázka s více možnými odpověďmi.

Scripts & Forms nabízí několik typů karet:

- Single Choice – karta zobrazující odpovědi jako přepínače (radio button), může být vybrána pouze jedna odpověď
- Multi Choice – karta zobrazující odpovědi jako zaškrtávací pole (checkbox), může být vybráno více odpovědí
- Form – karta, obsahující jednu z následujících možností:
    - Drop Down List – karta zobrazující odpovědi jako rozbalovací seznam, může být vybrána pouze jedna odpověď
    - Text – karta zobrazující textové pole, do kterého uživatel zadává text
    - Date – karta zobrazující výběr data (date picker)
    - Number – karta zobrazující textové pole, které akceptuje pouze číslice

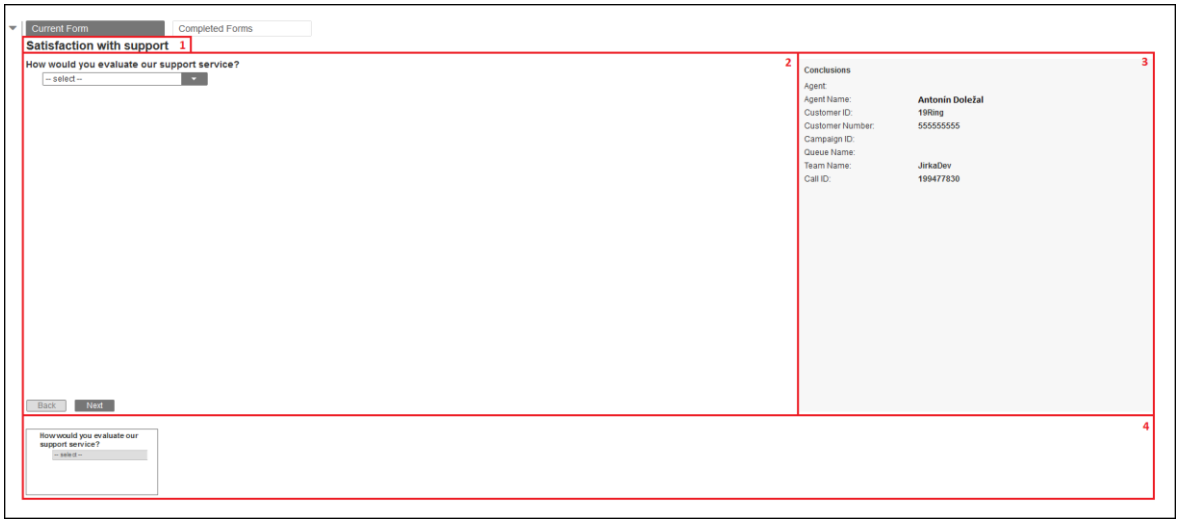
Na základě informací obdržených formou URL adresy dojde v aplikaci k načtení příslušného dotazníku odpovědnému agentovi ke konkrétnímu hovoru a zákazníkovi. Při vyplňování dotazníku jsou dle typu odpovědi zobrazovány různé typy karet. Spolu s vyplňováním dochází také k ukládání daného dotazníku do mezi paměti a teprve po vyplnění poslední karty dotazníku je vyplněný formulář uložen do databáze viz obrázek 25.



Obrázek 25: Vstupy a výstupy Scripts & Forms
Zdroj: vlastní

5.1.2 **Uživatelské rozhraní**

Tato podkapitola popisuje vybranou webovou aplikaci z uživatelského pohledu. V okamžiku, kdy aplikace obdrží potřebné vstupy, dojde k načtení aktuálního dotazníku. Aktuální karta obsahující první otázku a odpovědi je umístěna na levé straně. Vpravo se nachází informace s dotazníkem spojené a na spodu stránky je časová osa aktuálně vyplňovaného dotazníku viz obrázek 26.



Obrázek 26: Uživatelské rozhraní Scripts & Forms
Zdroj: Vlastní

Na obrázku 26 je vidět uživatelské rozhraní, které je rozděleno do několika segmentů:

1. Jméno aktuálního dotazníku
2. Aktuální karta dotazníku, tlačítka Back a Next
3. Informace k aktuálnímu dotazníku
4. Časová osa (timeline) aktuálního dotazníku

Tlačítka Back a Next slouží pro navigaci mezi jednotlivými kartami dotazníku. Uživateli je zbráněno přejít tlačítkem Next na další kartu v dotazníku, pokud se na aktuální kartě nacházejí validační chyby. Časová osa může být použita jako rychlá navigace mezi jednotlivými kartami aktuálního dotazníku, stačí pouze kliknout na náhled vybrané karty a aktuální karta bude změněna na vybranou kartu. Při přejití na poslední kartu aktuálního dotazníku je tlačítko Next nahrazeno tlačítkem Save, které po kliknutí způsobí ukončení dotazníku a jeho uložení do databáze.

5.1.3 **Koncept aplikace**

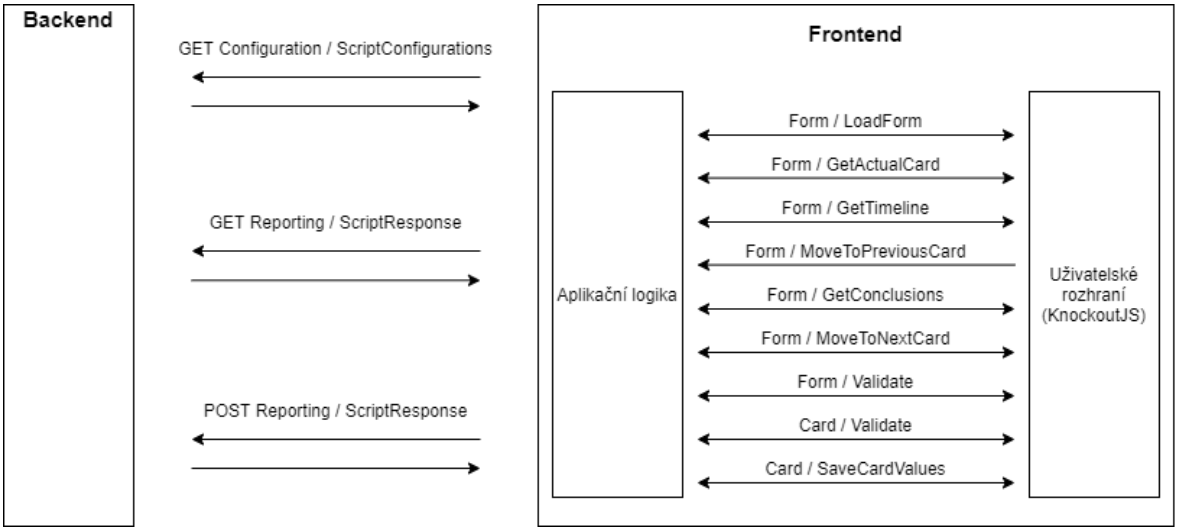
Scripts & Forms se stejně jako všechny jiné webové aplikace dělí na frontend a backend viz podkapitola 1.2 Frontend a backend webových aplikací. Řešení backendové strany aplikace není pro tuto diplomovou práci relevantní, a proto je věnována pozornost především

frontendové části aplikace. Nicméně je potřeba zmínit, že backend a frontend mezi sebou komunikují pomocí API. To tedy znamená, že frontend volá API backendu, aby získal potřebná data či funkcionalitu.

Frontendová část aplikace je rozdělena na další dva spolupracující celky:

- Aplikační logika
- Uživatelské rozhraní

Tyto dvě části spolu také komunikují pomocí API aplikační logiky viz obrázek 27.



Obrázek 27: Komunikace v aplikaci Scripts & Forms
Zdroj: vlastní

Komunikace mezi backendem a frontendem (Backend API)

GET Configuration / ScriptConfigurations / {name}

HTTP metoda, která na základě parametru name vrátí příslušný dotazník. Dotazník je zakódován pomocí Base64, rozkódování probíhá na straně klienta.

POST Reporting / ScriptResponse

HTTP metoda, která ukládá vyplněný dotazník do databáze a vrátí identifikátor uloženého dotazníku.

GET Reporting / ScriptResponse

HTTP metoda, která vrací na základě parametrů příslušné vyplněné dotazníky.

Komunikace mezi aplikační logikou a uživatelským rozhraním

Form / LoadForm (result: string)

Metoda, která se stará o načtení dotazníku.

Form / GetActualCard

Metoda, která vrátí aktuální kartu dotazníku.

Form / GetTimeline

Metoda, která vrátí časovou osu včetně příslušného seznamu karet.

Form / GetConclusions

Metoda, která vrací seznam odpovědí.

Form / MoveToPreviousCard

Metoda, která řeší přechod na předchozí kartu dotazníku.

Form / MoveToNextCard

Metoda, která se stará o přechod na další kartu v dotazníku. Tato metoda také řeší ukončení dotazníku, když se dotazník dostane na poslední kartu.

Form / Validate

Metoda, která zkontroluje daný dotazník, zda neobsahuje chyby.

Card / Validate (values: any)

Metoda, která zkontroluje, zda daná karta neobsahuje chyby.

Card / SaveCardValues(values: any)

Metoda, která uloží kartu.

5.2 Cíl a důvody refaktoringu

Aplikace Scripts & Forms byla vytvořena v letech 2011-2012 a pro vývoj prezenční vrstvy (frontend) aplikace byla zvolena v té době velmi nová technologie, JavaScriptová knihovna KnockoutJS. V průběhu let se aplikace rozrostla, technologie prohlížečů se masivně vyvíjely a technologie se stala limitující jak z hlediska údržby, tak z hlediska dalšího vývoje. Je tedy potřeba přeměnit (refaktorovat) původní kód na menší komponenty, moderní prezenční vrstvu a zlepšit tak i produktivitu.

5.2.1 **Důvody**

KnockoutJS je vynikající knihovna, ale pokud je aplikace dostatečně velká či komplexní, vykazuje určitá omezení. Prvním omezením je to, že se jedná pouze o knihovnu, nikoliv framework. Tato knihovna je navržena především ke spojování prvků uživatelského rozhraní s datovým modelem a samozřejmě k pár dalším funkcím, ale jinak zanechává úkol najít další vhodné knihovny či nástroje pro zvládnutí mnoha dalších věcí (metody životního cyklu komponent, state management atd.).

Dalším omezením je horší výkon. Moderní frameworky využívají nové funkce jazyka/prohlížeče, které v Knockoutu nejsou k dispozici. To se projevuje především v dlouhých či vnořených seznamech s komponentami, kde se Knouckout stává rychle velmi pomalým. A posledním velkým omezením je podpora komunity a vývoj. Knockout je stabilní knihovna, což je dobře, ale na druhou stranu hodně limitujících záležitostí určitě nebude rychle vyřešeno (jestli vůbec). Zájem komunity o tuto knihovnu také klesá, a proto je daleko jednoduší najít zdroje v podobě článků, knihoven, pluginů a nejrůznějších nástrojů právě pro nové moderní frameworky.

Knockout je dodnes využívaná knihovna, ještě zdaleka není odepsaná a ve skutečnosti se stále vylepšuje a vyvíjí, ale novější frameworky jsou v mnoha ohledech lepší a výkonnější.

5.2.2 **Cíl**

Cílů refaktoringu kódu a s tím spojených přechodů na novější technologie je hned několik. Hlavním cílem je celkové zjednodušení projektu (kódu) tak, aby bylo možné rychlejší a jednodušší přidávání nové funkcionality a správa aplikace jako taková. Dalším cílem je zlepšení optimalizace díky přechodu na novější technologie. Dalším důležitým cílem je také schopnost nastavit a dodržovat stejný design jako u ostatních firemních aplikací. A posledním cílem je lepší rozložení vybrané aplikace na komponenty a celkově lepší návrh těchto komponent.

Shrnutí cílů refaktoringu do několika bodů:

- Celkové zjednodušení projektu a standardizace
- Zjednodušení správy aplikace
- Optimalizace projektu pro současné prohlížeče
- Schopnost nastavit firemní design
- Lepší navržení komponent aplikace

5.3 Výběr frameworku

Výběr frameworku je jednou z klíčových částí této práce. V předešlých kapitolách jsou vybrané frameworky porovnávány dle nejrůznějších hledisek, a to včetně jejich odlišnosti v syntaxi. Každý z těchto frameworků má své výhody a nevýhody. A protože nelze jednoduše určit, který vybrat, je rozhodnutí učiněno na základě několika kritérií.

5.3.1 **Kritéria**

Prvním kritériem (faktorem) je firemní politika. Jak již bylo zmíněno v podkapitole 5.1 Scripts & Forms, vybraná webová aplikace je zastřešena společností 2Ring, spol. s r.o. Tato společnost nabízí, kromě Scripts & Forms, další spoustu produktů, které je potřeba při výběru také respektovat. V ostatních produktech se pro prezenční vrstvu aplikace ve většině případů využívá frameworku Vue.

Druhým kritériem je obtížnost a časová náročnost učení. Jelikož není možné za několik měsíců psaní této práce pochopit a naučit se „celý“ framework, tak je důležité naučit se co nejvíce v krátkém časovém horizontu. Toto kritérium je důležité také z dalšího důvodu. Projekt existuje a pravděpodobně bude existovat i v dalších letech, a proto je důležité, aby i jiní členové vývojového týmu nebo i nově příchozí zaměstnanci uměli s danou technologií zacházet. Tedy čím jednodušeji a rychleji se naučí s daným frameworkem zacházet, tím lépe.

Dalším kritériem je komunita a budoucí vývoj daného frameworku. Aktivní komunita je základní podmínkou pro zdravý a vyvíjející se ekosystém okolo frameworku a zároveň snižuje riziko nutnosti opětovné změny technologie v dalších letech.

A posledním kritériem je osobní názor, preference autora této práce, který měl možnost se s jednotlivými frameworky seznámit a prakticky vyzkoušet alespoň jejich základní principy. Autor práce nemá mnoho zkušeností s žádným JavaScriptovým frameworkem, pouze základní znalost HTML, CSS a JavaScriptu. Framework Vue se zdá být nejvíce intuitivním, méně obtížným na učení než jeho kolegové a celková práce s tímto frameworkem se jeví jako přirozená.

5.3.2 **Rozhodnutí**

Angular je plně vybavený framework, který si klade za cíl poskytnout vývojáři vše potřebné pro vývoj moderní webové aplikace. Díky svým daným postupům nabízí jasný způsob, jak psát kód a organizovat projekt. Angular je považován za komplikovaný, zato ale nabízí mnoho zajímavých funkcí a nástrojů. Nicméně pro projekt Scripts & Forms, který využívá i jiných knihoven, se příliš nehodí. Vše potřebné je již součástí frameworku a díky tomu by mohlo docházet k problémům s duplicitní funckionalitou a ke zbytečnému zvyšování složitosti daného projektu.

React je mnohem lehčí a méně robustní než Angular, ale jedná se o knihovnu, která je závislá na knihovnách třetích stran. To způsobuje, že existuje velké množství možností, ze kterých si vývojář může vybírat. Na jednu stranu je to dobře, na druhou stranu to vývojáře zatěžuje tím, že musí najít a integrovat tu „správnou“ knihovnu pro mnoho různých věcí. Je také

důležité zmínit, že kvůli jeho funkcionálnímu přístupu k programování a využití JSX místo jednoduchého HTML, není příliš jednoduchý na naučení.

Vue je střední cestou mezi plně vybaveným Angularem a Reactem. Vue má vlastní oficiální knihovny pro state management a routing, které jsou volitelné a nejsou tak součástí základu Vue. Tyto knihovny jsou spravovány týmem Vue, čímž je zaručena jejich dobrá spolupráce s frameworkem. Vue má také výhodu v tom, že je navrženo tak, aby mohlo být do projektu přidáváno postupně, tudíž není nutné, aby byl přepsán celý projekt najednou. Vue je také jednodušší na učení ve srovnání s Angularem a Reactem.

Jak již je možná zřejmé, vybraných frameworkem je Vue.js, protože jeho silné stránky nejlépe odpovídají zvoleným kritériím a nevýhody nejsou příliš relevantní. Jednoduché shrnutí do několika bodů:

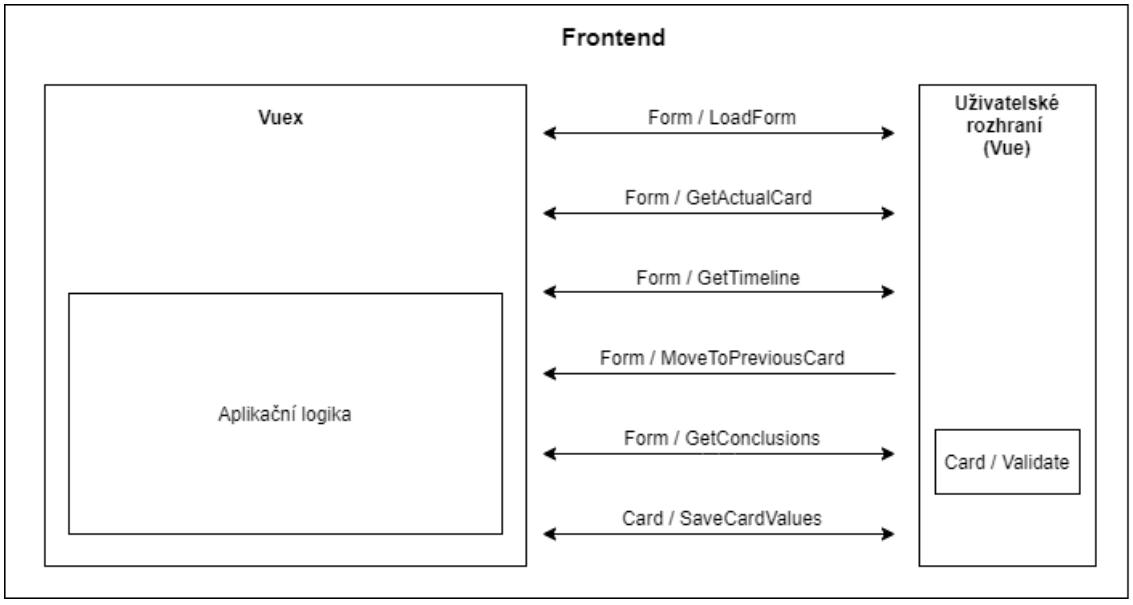
- Používá se i v ostatních projektech ve firmě.
- Vue je navrženo tak, aby nezpůsobovalo problémy při integraci s jinými implementacemi.
- Je nejjednodušší na naučení v porovnání s benefity, které přináší.
- I přes to, že má menší komunitu než React nebo Angular, jedná se o nejrychleji rostoucí framework, takže obavy o nedostatek různých knihoven a pluginů by měli být zbytečné.

5.4 Postup vývoje

Tato podkapitola popisuje, jak je framewrok Vue postupně integrován do aplikace a jaké kroky jsou podniknuty. Podkapitola začíná přidáním Vuex do webové aplikace, a změn s ním spojených. Dále následuje popis rozdělení aplikace na jednotlivé komponenty a na závěr je demonstrováno několik ukázek použití frameworku Vue.

5.4.1 **Přidání Vuex**

Nejprve dochází k přidání centralizovaného uložiště Vuex, tedy oficiální knihovny Vue zajišťující state management. Tím vzniká několik změn ve frontendu aplikace. State management aplikace byl do této doby pod aplikační logikou, ta se díky tomu nyní stává součástí Vuex. V rámci tohoto přechodu dochází také ke zjednodušení API mezi aplikační logikou a uživatelským rozhraním viz obrázek 28.



Obrázek 28: Zjednodušení API
Zdroj: vlastní

API mezi aplikační logikou a uživatelským rozhraním je zjednodušeno o následující změny:

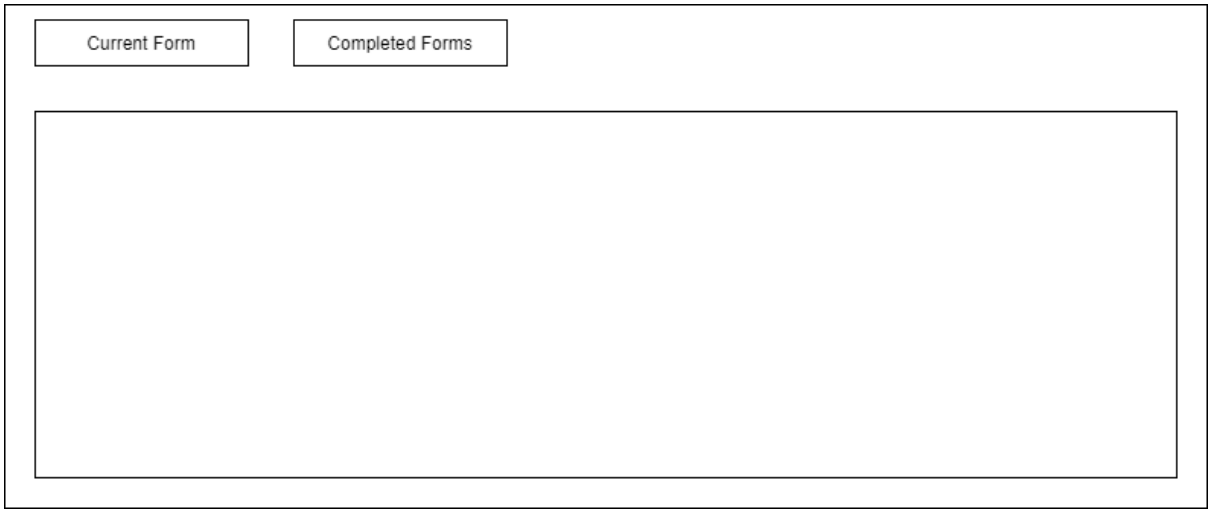
- Metoda Form / MoveToNextCard je nově součástí metody Card / SaveCardValues. Není důvod ke dvojímu volání, pokud probíhá volání metody pro uložení karty, automaticky dochází k přechodu na další kartu v dotazníku.
- Metoda Card / Validate již neexistuje a validace probíhá na straně uživatelského rozhraní.
- Metoda Form / Validate se stala součástí aplikační logiky.

V rámci refaktoringu dochází také ke změnám volání metod POST Reporting / ScriptResponse a GET Reporting / ScriptResponse, které probíhalo přímo z uživatelského rozhraní, to by se dít nemělo, a proto je volání těchto metod přesunuto do Vuex.

5.4.2 **Uživatelské rozhraní Vue**

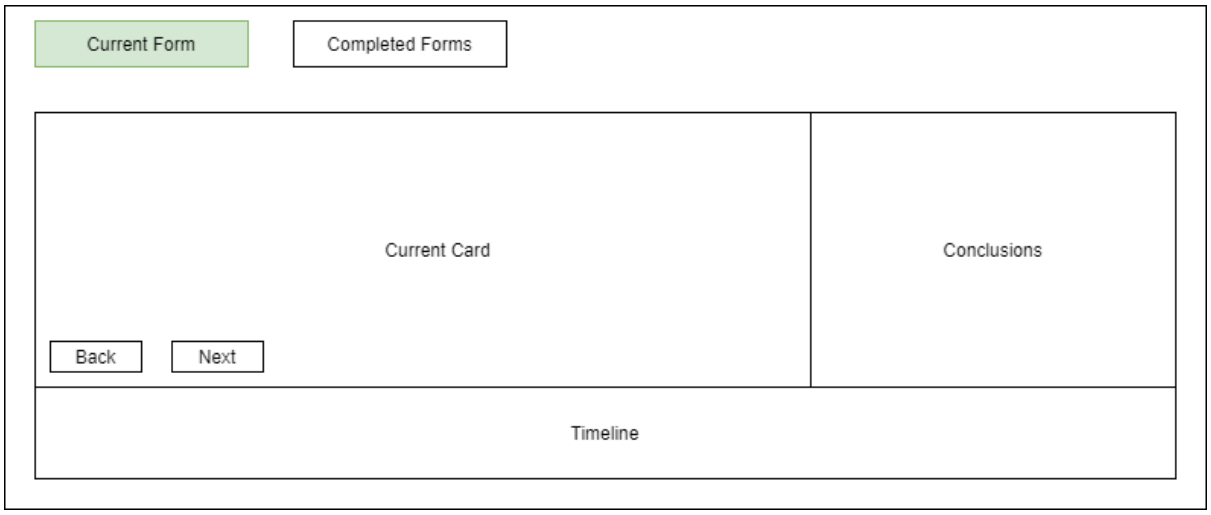
Uživatelské rozhraní je nejprve rozděleno na jednotlivé komponenty a poté jsou tyto komponenty postupně rozšiřovány o funkcionalitu. Rozdělení webu na komponenty probíhá následujícím způsobem.

Webovou aplikaci je nejprve potřeba rozdělit na dvě samostatné stránky, Current Form (aktuální dotazník) a Completed Forms (vyplněné dotazníky) viz obrázek 29.



Obrázek 29: Rozdělení uživatelského rozhraní, krok 1
Zdroj: vlastní

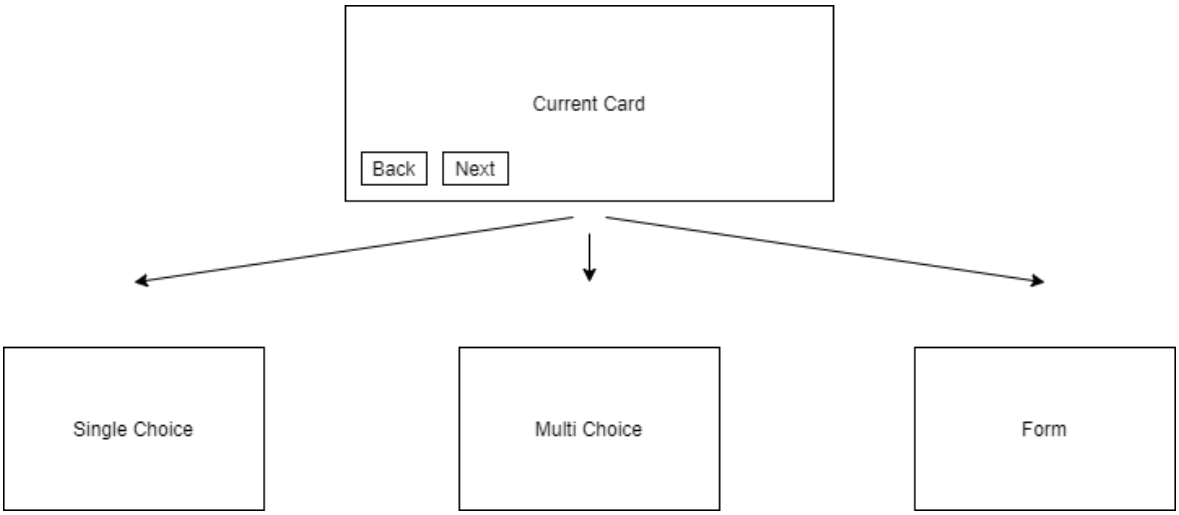
Dále je zapotřebí rozdělit komponentu Current Form na další tři komponenty jako na obrázku 30.



Obrázek 30: Rozdělení uživatelského rozhraní, krok 2
Zdroj: vlastní

- Current Card – komponenta zobrazující aktuální kartu dotazníku a tlačítka Back a Next pro přecházení mezi jednotlivými kartami
- Conclusions – komponenta zobrazující informace o aktuálním dotazníku (např. odpovědi)
- Timeline – komponenta zobrazující časovou osu aktuálního dotazníku

Pro komponentu CurrentCard je potřeba ještě vytvořit další tři komponenty, které se budou zobrazovat jednotlivě na základě typu aktuální karty dotazníku viz obrázek 31.



Obrázek 31: Rozdělení uživatelského rozhraní, krok 3
Zdroj: vlastní

5.4.3 **Ukázky implementace**

Tato podkapitola ukazuje kousky kódu, které jsou použity v aplikaci Scripts & Forms. Jak již bylo několikrát zmíněno, cílem práce není ukázat celý vývoj webové aplikace, ale pouze demonstrovat použití daného frameworku a možný postup při vývoji, a proto je v této podkapitole ukázáno pouze několik příkladů.

Vuex

V této ukázce je zobrazena část souboru *index.ts*, který je reprezentací Vuex. Nahoře jsou nejprve přidány závislosti pro Vue, Vuex a model dotazníku (*WizardForm*). Poté následuje klíčové spojení *Vue.use(Vuex)* a definice storu viz ukázka kódu 62.

```
import Vue from 'vue'
import Vuex from 'vuex'
import { WizardForm } from '@/models/form/wizardform'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    wizard: WizardForm,
  },
  mutations: {
    addWizard (state, wizardModel: WizardForm) {
      Vue.set(state, 'wizard', wizardModel)
      ...
    }...
  },
  actions: {
    addWizard (context, wizardModel: WizardForm) {
      context.commit('addWizard', wizardModel)
    },
    ...
  },
  modules: { } })
```

Ukázka kódu 62: Vuex

- *State* obsahuje stav aplikace, v tomto případě wizard (dotazník), který je typu WizardForm.
- *Mutations*, mutace, jsou jediným způsobem, jak změnit stav ve Vuex storu. Mutace *addWizard*, přidá do Vuex na základě obdržených parametrů dotazník. Mutace nemohou být volány přímo, a proto je k jejich volání použito *commit (název)*.
- *Actions* obsahuje akci *addWizard*, která volá stejnojmennou mutaci *addWizard* pomocí *context.commit ('addWizard', wizardModel)*.

Bindování dat v komponentě

Tato ukázka vychází z předchozí ukázky a naznačuje bindování, vázání dat v komponentě viz ukázka kódu Ukázka kódu 63.

```
<template>
...
      <h4> {{ card.Question }}</h4>
...
</template>

<script lang="ts">
...
export default class CardPage extends Vue {
  get card (): Card | undefined {
    const wizard = this.$store.state.wizard
    if (wizard && wizard.tree) {
      return wizard.tree.actualCard
    }
  }
...
}
</script>
```

Ukázka kódu 63: Data v komponentě

Metoda *get card*, která vrací objekt *Card* získaný z Vue storu.

- *const wizard* – definování proměnné *wizard*, do které je přiřazena hodnota ze storu
- *this.$store.state.wizard* – přístup do storu do oblasti *state*, k objektu *wizard*
- *if (wizard…)* je ověření, že proměnná *wizard* byla naplněna
- *return wizard.tree.actualCard* – vrátí aktuální kartu dotazníku

V oblasti template je HTML element <h4> pro nadpis obsahující otázku, *card.Question*, ke kartě, která byla získána v metodě *get card*. Tímto způsobem lze přistupovat ke všem ostatním vlastnostem objektu *card*, například pro zobrazení první odpovědi by bylo použito: *card.Answer1*.

Přidání dat do storu

Ukázka funkce *fetchData()*, která je volána při načítání komponenty *CurrentForm*, tato funkce má za úkol získat dotazník a uložit ho do Vuex storu.

```
fetchData () {
    axios.get(...).then(response => {
      const wizard = new WizardForm()
      wizard.Load(response.data)
      this.$store.dispatch('addWizard', wizard)
    })
  }
```

Ukázka kódu 64: Posílání dat do storu

- *axios.get* – pošle dotaz na backend a odpověď uloží do *response*
- *const wizard = new WizardForm()* – definice proměnné *wizard*, do které je přiřazen nový objekt typu *WizardForm*
- *wizard.Load(response.data)* – načte do proměnné *wizard* dotazník z odpovědi (*response*)
- *this.$store.dispatch('addWizard', wizard*) – zavolá akci storu *addWizard* a předá dotazník

Přidání komponenty

K přidání komponenty *CurrentCard* je potřeba nejprve vytvořit nový soubor (komponentu) ve složce components. Soubor je pojmenován *CurrentCard.vue*. Soubor obsahuje značky pro začátek a konec oddílů *template* a *script*. V oddílu *template* je element *div* obsahující název dané komponenty. V oddílu script jsou naimportovány dekorátory pro komponentu a pro Vue a poté definována třída, která nese název komponenty viz ukázka 65.

```
<template>
  <div>
    Current Card
  </div>
</template>

<script lang="ts">
import { Component, Prop, Vue } from 'vue-property-decorator';

@Component
export default class CurrentCard extends Vue {
}
</script>
```

Ukázka kódu 65: Vytvoření komponenty

Po vytvoření komponenty, je komponenta ještě přidána do souboru *CurrentForm.vue*. Nejprve je třeba uvést závislost na tuto komponentu.

```
import CurrentCard from '@/components/CurrentCard.vue';
```

Ukázka kódu 66: Závislost v CurrentForm.vue

A poté komponentu ještě zmínit za dekorátorem *@Component*.

```
@Component({
  components: {
    CurrentCard
  },
})
```

Ukázka kódu 67: Přidání komponenty do CurrentForm.vue

Nyní je možné ke komponentě přistupovat v oblasti *template*, ve které je komponenta uvedena, aby došlo k jejímu zobrazení.

```
<template>
  <div>
    <CurrentCard />
  </div>
</template>
```

Ukázka kódu 68: Zobrazení komponenty

5.5 Zhodnocení

Při porovnání projektu před a po refaktorování je vidět, že framework Vue sebou do projektu přinesl jasnou konstrukci, která přispěla k zjednodušení celého projektu a jeho standardizaci. Vuex (Vue Store), oficiální knihovna, která zajišťuje state management Vue aplikací, přispěla mnoha zlepšeními. Například byl zaveden jednotný přístup k ukládání a správě dat, vznikl systém pro automatickou aktualizaci dat, došlo ke zjednodušení přechodu mezi kartami dotazníku a další.

Díky přidání knihovny pro routing není již nutné oddělovat a vytahovat parametry z URL adresy vlastními funkcemi, protože Vue routing toto již umí sám. Došlo také k lepšímu navržení komponent. Nejlépe budou všechny změny vidět jako shrnutí do několika bodů:

- Celkové zjednodušení projektu a standardizace
- Vue Store
    - Jednotný přístup k ukládání dat

    - Zavedení systému, který data aktualizuje
    - Možnost nad daty provádět akce
    - Aktualizace dat probíhá automaticky (dříve ne)
    - Zjednodušení přechodu mezi kartami dotazníku
    - Obnovování časové osy (Timeline)
    - Není potřeba hlídat vztahy mezi komponentami
- Routing
    - Dříve bylo potřeba ručně oddělovat a vytahovat parametry – Vue routing už umí sám
- Lepší navržení komponent
    - Každá komponenta může mít vlastní CSS
    - Vue řeší ID v rámci komponenty

Pokud se nyní znovu podíváme na definované cíle (viz podkapitola 5.2.2 Cíl):

- Celkové zjednodušení projektu a standardizace
- Zjednodušení správy aplikace
- Optimalizace projektu pro současné prohlížeče
- Schopnost nastavit firemní design
- Lepší navržení komponent aplikace

Lze říci, že všechny stanovené cíle byly splněny.

Implementaci nového frameworku lze částečně zhodnotit i dle ekonomického hlediska. Pro firmu tento krok může představovat v krátkém časovém období nevýhodu, především v časové náročnosti celého procesu implementace. V dlouhém období bude však tato investice firmě výhodou, a to hned v několika ohledech. Za prvé vynaložený čas na implementaci vývojáři ušetří při následné údržbě aplikace, neboť nový framework je mnohem jednoduší na správu. Dále se díky sjednocení vývojové technologie s jinými produkty ve firmě zefektivní a také zkrátí doba přechodu mezi jednotlivými projekty. Další možnou výhodou je, že nově příchozí vývojáři se nemusí učit starou technologii a dojde tak k urychlení procesu jejich zaučení.

Závěr

Cílem, definovaným v úvodu této diplomové práce nesoucí název *JavaScriptové frameworky pro vývoj webových aplikací*, bylo nastínit možný postup použití konkrétního JavaScriptového frameworku na reálném projektu.

V prvé řadě se práce věnovala uvedení do problematiky vývoje moderních webových aplikací. Zabývala se také pojmy, které s touto problematikou souvisí jako například klient-server model, nebo rozdělení webových aplikací na prezenční vrstvu neboli frontend a vrstvu pracují s daty, tedy backend.

Další část práce byla zaměřena na popis tří JavaScriptových frameworků, konkrétně Angular, React a Vue. Byly vybrány, protože patří mezi nejpopulárnější a nejpoužívanější frameworky současnosti. Nejdříve byli krátce představeny a poté porovnávány podle různých hledisek. Dále byli ukázány jejich odlišnosti v syntaxi na vybraných příkladech.

Práce se poté věnovala postupu vytvoření nového projektu, a to zvlášť pro každý vybraný framework. V rámci postupu byla popsána jak samotná instalace projektu, tak instalace nutných nástrojů a závislostí. Dále bylo vysvětleno základní rozložení aplikace a popsán proces prvního spuštění nově vytvořeného projektu. Na konci kapitoly je uvedeno, jak lze provést jednoduchou úpravu aplikace.

Dále se práce posunula k samotné implementaci frameworku na určité aplikaci. V prvé řadě byla popsána webová aplikace, Scripts & Forms, která je zastřešena společností 2Ring, spol. s r.o. Popis zahrnoval funkcionalitu, uživatelské rozhraní a koncept celé aplikace. Poté jsou v práci zmíněny důvody a cíle přechodu na nové technologie. Hlavním důvodem k přechodu byla zastaralost používaných technologií, které se staly limitujícími jak z hlediska údržby, tak z hlediska dalšího vývoje. Na základě těchto důvodů došlo k přepsání aplikace s cíli celkového zjednodušení projektu a jeho standardizace, zjednodušení správy, optimalizace pro současné webové prohlížeče a schopnosti nastavit firemní design.

Pro implementaci bylo potřeba zvolit vhodný framework, a proto byla definovaná klíčová kritéria. Těmito kritérii byla firemní politika, obtížnost a časová náročnost, budoucí vývoj frameworku a v neposlední řádě také osobní názor uživatele (v tomto případě autora práce).

Po zhodnocení potřebných dat byl vybrán framework Vue, který nejlépe splňoval požadovaná kritéria. Prvním krokem samotné implementace bylo přidání knihovny Vuex, která se stará o správu stavu aplikace. Tím došlo ke spojení aplikační logiky s knihovnou Vuex, což vedlo ke zjednodušení komunikace mezi částmi prezenční vrstvy aplikace. Dalším krokem byla úprava uživatelského rozhraní, které bylo nejprve potřeba rozdělit na komponenty.

Uskutečněním výše zmíněných kroků implementace byl použit konkrétní JavaScriptový framework na reálném projektu, čímž byl splněn hlavní cíl práce. Dalším přínosem práce je pomoc při rozhodování mezi vybranými JavaScriptovými frameworky. Tato problematika byla řešena nejprve v kapitole 2, kde byli definovány hlavní rozdíly mezi danými frameworky. Bylo zjištěno, že Angular nabízí komplexní řešení pro vývoj moderních webových aplikací. React je naopak strohé řešení, které spoléhá na knihovny třetích stran, čímž ale také otevírá spoustu možností. Střední cestou mezi plně vybaveným Angularem a strohým Reactem je Vue, které nabízí vlastní oficiální knihovny, ale je možné použít i knihovny třetích stran. V kapitole 3 pak byly tyto frameworky více rozebrány z pohledu jejich syntaxe.

Seznam použité literatury

Citace

AFIFI-SABET, Keumars. 2019. What is TypeScript? *ITPro* [online]. 2019 [cit. 2020-07-25]. Dostupné z: https://www.itpro.co.uk/development/32886/what-is-typescript

AGARWAL, Harsh. 2019. ReactJS | Components. *GeeksforGeeks* [online]. 2019 [cit. 2020-07-25]. Dostupné z: https://www.geeksforgeeks.org/reactjs-components/

ANGULAR. 2020. CLI Overview and Command Reference. *Angular* [online]. Google, 2020 [cit. 2020-07-26]. Dostupné z: https://angular.io/cli

ANGULAR. 2020. Component styles. *Angular* [online]. Google, 2020 [cit. 2020-07-26]. Dostupné z: https://angular.io/guide/component-styles

ANGULAR. 2020. Dependency injection in Angular. *Angular* [online]. Google, 2020 [cit. 2020-07-25]. Dostupné z: https://angular.io/guide/dependency-injection

ANGULAR. 2020. Hooking into the component lifecycle. *Angular* [online]. Google, 2020 [cit. 2020-07-26]. Dostupné z: https://angular.io/guide/lifecycle-hooks

ANGULAR. *2020. Introduction to components and templates* [online]. Google, 2020 [cit. 2020-07-26]. Dostupné z: https://angular.io/guide/architecture-components

ANGULAR. 2020. Introduction to forms in Angular. *Angular* [online]. Google, 2020 [cit. 2020-07-26]. Dostupné z: https://angular.io/guide/forms-overview

ANGULAR. 2020. NgIf. *Angular* [online]. Google, 2020 [cit. 2020-07-26]. Dostupné z: https://angular.io/api/common/NgIf

ANGULAR. 2020. Template syntax. *Angular* [online]. Google, 2020 [cit. 2020-07-26]. Dostupné z: https://angular.io/guide/template-syntax#interpolation

ANGULAR. 2020. @angular/core. *Angular* [online]. Google, 2020 [cit. 2020-07-25]. Dostupné z: https://angular.io/api/core

ASTON, Ben. 2015. A brief history of JavaScript. *Medium* [online]. 2015 [cit. 2020-07-25]. Dostupné z: https://medium.com/@_bjma/lesson-1a-the-history-of-javascript-8c1ce3bffb17

BABEL. 2020. What is Babel? *Babel* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://babeljs.io/docs/en/

BARKER, Adam. 2018. The Super-Brief History of JavaScript Frameworks For Those Somewhat Interested. *Dev* [online]. [cit. 2020-07-25]. Dostupné z: https://dev.to/_adam_barker/the-super-brief-history-of-javascript-frameworks-for-those-somewhat-interested-3m82

BARTH, Laurie. 2019. Component Lifecycle Methods Explained. *Ten mile square* [online]. 2019 [cit. 2020-07-26]. Dostupné z: https://tenmilesquare.com/component-lifecycle-methods-explained/

DAITYARI, Shaumik. 2020. Angular vs React vs Vue: Which Framework to Choose in 2020. *Codeinwp* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

DEGROAT, T.J. 2019. The History of JavaScript: Everything You Need to Know. *Springboard Blog* [online]. 2019 [cit. 2020-07-25]. Dostupné z: https://www.springboard.com/blog/history-of-javascript/

GRZYBEK, Marek. 2018. Frontend Frameworks Code Comparison. *GitHub* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://github.com/feimosi/frameworks-code-comparison

GURU99. 2020. Difference between Website and Web Application. *Guru99* [online]. Guru99, 2020 [cit. 2020-07-22]. Dostupné z: https://www.guru99.com/difference-web-application-website.html

CHRISTENSSON, P. 2016. Client-Server Model Definition. *TechTerrms* [online]. 2016 [cit. 2020-07-22]. Dostupné z: https://techterms.com/definition/client-server_model

ISTQB. 2020. Unit Testing. *Software Testing Fundamentals* [online]. 2020 [cit. 2020-07-26]. Dostupné z: http://softwaretestingfundamentals.com/unit-testing/

KAGGA, John. 2018. Understanding React Components. *Medium* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb

KONONENKO, K. 2018. The Relationship Between HTML, CSS and JavaScript Explained by Building A City. *CodeAnalogies Blog* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://blog.codeanalogies.com/2018/05/09/the-relationship-between-html-css-and-javascript-explained/

KOPILEVYCH, Daniil. 2020. Angular vs. React vs. Vue.js – choosing a JavaScript framework for your project. *Relevant* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://relevant.software/blog/angular-vs-react-vs-vue-js-choosing-a-javascript-framework-for-your-project/

KUMAR, Amit. 2019. What Is TypeScript? *C#Corner* [online]. 2019 [cit. 2020-07-25]. Dostupné z: https://www.c-sharpcorner.com/article/what-is-typescript/

LINDLEY, Cody. 2019. What Is JSX? *React JS - React Enlightenment* [online]. 2019 [cit. 2020-07-25]. Dostupné z: https://www.reactenlightenment.com/react-jsx/5.1.html

MÁCA, Jindřich. 2018. Úvod do Node.js. *ITNetwork* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs

MAJ, Wojciech. 2020. React Lifecycle Methods diagram. *Projects Wojtek Maj* [online]. 2020 [cit. 2020-07-26]. Dostupné z: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

MDN, contributors. 2020. CSS preprocessor. *MDN web docs* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor

MDN, contributors. 2020. Introduction to the DOM. *MDN web docs* [online]. [cit. 2020-07-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

MDN, contributors. 2020. Progressive web apps (PWAs). *MDN web docs* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

MDN WEB DOCS. What is JavaScript? *2020*. *MDN web docs* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

MORRIS, Scott. 2020. JAVASCRIPT FRAMEWORKS VS LIBRARIES—WHAT'S THE DIFFERENCE? *Skillcrush* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://skillcrush.com/blog/javascript-frameworks-vs-libraries/

MORRIS, Scott. 2020. WHAT IS A JAVASCRIPT FRAMEWORK? HERE'S EVERYTHING YOU NEED TO KNOW. *Skillcrush* [online]. [cit. 2020-07-25]. Dostupné z: https://skillcrush.com/blog/what-is-a-javascript-framework/

ONLINE WEB FONTS. 2018. *Icon* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://www.onlinewebfonts.com/icon/

OPENJS FOUNDATION. 2011. What is npm? *Node.js* [online]. 2011 [cit. 2020-07-25]. Dostupné z: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

REACT. 2020. Conditional Rendering. *React* [online]. Facebook, 2020 [cit. 2020-07-26]. Dostupné z: https://reactjs.org/docs/conditional-rendering.html

REACT. 2020. Create a New React App. *React* [online]. Facebook, 2020 [cit. 2020-07-26]. Dostupné z: https://reactjs.org/docs/create-a-new-react-app.html

REACT. 2020. Forms. *React* [online]. Facebook, 2020 [cit. 2020-07-26]. Dostupné z: https://reactjs.org/docs/forms.html

REACT. *2020. React.Component* [online]. Facebook, 2020 [cit. 2020-07-25]. Dostupné z: https://reactjs.org/docs/react-component.html

REACT. 2020. Styling and CSS. *React* [online]. Facebook, 2020 [cit. 2020-07-26]. Dostupné z: https://reactjs.org/docs/faq-styling.html

ROUSE, Margaret. 2016. Single-page application (SPA). *TechTarget* [online]. 2016 [cit. 2020-07-25]. Dostupné z: https://whatis.techtarget.com/definition/single-page-application-SPA

ROUSE, Margaret. 2020. State management. *TechTarget* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://searchapparchitecture.techtarget.com/definition/state-management

SCHEPENAAR, Wilbert. 2017. Server-side vs Client-side Routing. *Medium* [online]. 2017 [cit. 2020-07-25]. Dostupné z: https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f

SINGHAL, P. 2019. Frontend vs Bakend. *GeeksforGeeks* [online]. 2019 [cit. 2020-07-25]. Dostupné z: https://www.geeksforgeeks.org/frontend-vs-backend/

SKÓLSKI, Paweł. 2016. Single-page application vs. multiple-page application. *Medium* [online]. 2016 [cit. 2020-07-25]. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

SOFTWARE TESTING HELP. 2020. What Is End To End Testing: E2E Testing Framework With Examples. *Software Testing Help* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://www.softwaretestinghelp.com/what-is-end-to-end-testing/

STENCIL, CXJS a STIMULUS. 2018. The Ultimate Guide to JavaScript Frameworks. *JavaScript Report* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://jsreport.io/the-ultimate-guide-to-javascript-frameworks/

SPEC INDIA. 2018. React vs Angular vs Vue.js: A Complete Comparison Guide. *Medium* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://medium.com/front-end-weekly/react-vs-angular-vs-vue-js-a-complete-comparison-guide-d16faa185d61

TUTORIALS POINT. 2019. Design Patterns - Decorator Pattern. *Tutorials Point* [online]. [cit. 2020-07-25]. Dostupné z: https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm

TUTORIALS POINT. 2020. Node.js - Introduction. *Tutorials Point* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

VUE.JS. 2020. Components Basics. *Vue.js* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://vuejs.org/v2/guide/components.html

VUE.JS. 2020. Conditional Rendering. *Vue.js* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://vuejs.org/v2/guide/conditional.html

VUE.JS. 2020. Form Input Bindings. *Vue.js* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://vuejs.org/v2/guide/forms.html

VUE.JS. 2020. Installation. *Vue CLI* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://cli.vuejs.org/guide/installation.html

VUE.JS. 2020. Lifecycle Diagram. *Vue.js* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram

VUE.JS. 2020. Scoped CSS. *Vue.js* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://vue-loader.vuejs.org/guide/scoped-css.html

VUE.JS. 2020. Template Syntax. *Vue.js* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://vuejs.org/v2/guide/syntax.html

VUE.JS. 2020. What is Vuex? *Vuex* [online]. 2020 [cit. 2020-07-26]. Dostupné z: https://vuex.vuejs.org/

WANYOIKE, M. 2018. History of front-end frameworks. *LogRocket* [online]. 2018 [cit. 2020-07-25]. Dostupné z: https://blog.logrocket.com/history-of-frontend-frameworks/

W3SCHOOLS. 2020. AJAX Introduction. *W3schools* [online]. 2020 [cit. 2020-07-25]. Dostupné z: https://www.w3schools.com/js/js_ajax_intro.asp

Bibliografie

AMBLER, Tim a Nicholas CLOUD. 2015. JavaScript frameworks for modern web dev. New York: Apress, Expert's voice in Web development. ISBN 978-1-4842-0663-8.

BLOKDYK, Gerardus. 2018. Comparison of Javascript Frameworks. California, USA: CreateSpace Independent Publishing Platform, ISBN 978-1963828225.

FREEMAN, Adam.2018. Pro Vue.JS 2. New York, NY: Springer Science Business Media, ISBN 978-1-4842-3804-2.

ŽÁRA, Ondřej. 2015. JavaScript: programátorské techniky a webové technologie. Brno: Computer Press, ISBN 978-80-251-4573-9.