Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Vývoj webových aplikací pomocí JavaScript frameworků

Bc. David Pocar

© 2021 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. David Pocar

Systémové inženýrství a informatika
Informatika

Název práce

Vývoj webových aplikací pomocí JavaScript frameworků

Název anglicky

JavaScript frameworks for web application development

Cíle práce

Diplomová práce je tematicky zaměřena na problematiku vývoje webových aplikací pomocí JavaScript (JS) frameworků.
Hlavním cílem práce je analyzovat a zhodnotit vybrané JS frameworky vhodné pro vývoj webových aplikací.
Dílčí cíle práce jsou:
- charakterizovat problematiku vývoje webových aplikací,
- zhodnocení vybraných JS frameworků,
- analyzovat možnosti využití JS frameworků pro tvorbu webových aplikací.

Metodika

Teoretická část diplomové práce se bude zakládat na analýze a rešerši odborných zdrojů zaměřených na problematiku vývoje webových aplikací a JS frameworků.

V praktické části práce budou na základě poznatků zjištěných v analytické části zhodnoceny vybrané JS frameworky a formulovány možnosti využití jednotlivých frameworků pro vybrané typy webových aplikací.

Na základě syntézy teoretických a praktických poznatků budou zpracovány závěry diplomové práce.

Doporučený rozsah práce

60–80 stran

Klíčová slova

javascript, js, framework, webové aplikace

Doporučené zdroje informací

FLANAGAN, D. *JavaScript : the definitive guide.* Sebastopol, CA: O'Reilly, 2002. ISBN 0-596-00048-0.
Green, Brad. AngularJS. Sebastopol, CA: O'Reilly, 2013. ISBN 978-1449344856.
Lopez, Lionel. React: QuickStart Step-By-Step Guide to Learning React JavaScript Library. Scotts Valley, CA: Nakladatelství Createspace Independent Publishing Platform, 2017. ISBN 978-1976210235.
Macrae, Callum. Vue.js – Up and Running. Sebastopol, CA: O'Reilly, 2018. ISBN 978-1491997246.
Myers, Mark. A Smarter Way to Learn JavaScript: The new tech-assisted approach that requires half the effort. Scotts Valley, CA: CreateSpace Independent Publishing Platform, 2014. ISBN 978-1497408180
Williamson, Ken. Learning AngularJS. Sebastopol, CA: O'Reilly, 2015. ISBN 978-1491916759.
Zakas, Nicholas C. Understanding Ecmascript 6. San Francisco, CA: No Starch Press, 2016. ISBN 978-1593277574.
Žára, Ondřej. JavaScript Programátorské techniky a webové technologie. Brno: Computer Press, 2016. ISBN 978-80-251-4573-9.

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Michal Stočes, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 29. 7. 2020

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 21. 10. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 30. 03. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Vývoj webových aplikací pomocí JavaScript frameworků“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.3.2021

Poděkování

Rád bych touto cestou poděkoval Ing. Michalovi Stočesovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při vedení této práce.

Zvláště bych rád poděkoval své rodině, která mě během celého studia a vypracovávání této diplomové práce značně podporovala a bez které bych studium nebyl schopen dokončit.

Vývoj webových aplikací pomocí JavaScript frameworků

Abstrakt

Diplomová práce pojednává o problematice vývoje webových aplikací zaměřených na interaktivitu s uživateli, které jsou dnes řešeny JavaScriptovými frameworky. Vlastní práce je zaměřena na zhodnocení právě těchto frameworků na základě zvolených kritérií.

V první části práce byla charakterizována problematika samotného vývoje webových aplikací, kde byly popsány nejdůležitější technologie a náležitosti, které jsou s vývojem spojeny. Na konci této části byl blíže popsán použitý způsob hodnocení, jenž se zakládal na předem zvoleném modelu vícekriteriální analýzy variant.

Druhá část byla věnována vlastní práci, kde bylo v první řadě potřeba vybrat vhodné JavaScriptové frameworky k samotnému hodnocení. Výběr byl realizován pomocí modelu vícekriteriální analýzy variant. K samotnému hodnocení byla vybrána jednotlivá kritéria na základě čtyř předem zvolených aspektů, a to aspektů výkonu, komunity, použitelnosti a vývoje. Vybranými kritérii byly zhodnoceny vybrané frameworky, a to opět pomocí modelu vícekriteriální analýzy variant.

Poznatky vzešlé z vlastní práce byly využity k analýze rozdílů vybraných frameworků, kterými bylo následně možné analyzovat i jejich možnosti využití.

Klíčová slova: javascript, js, framework, webové aplikace, frontend, React, Vue, Angular, vícekriteriální analýza variant

JavaScript Frameworks
For Web Application Development

Abstract

This master thesis deals with the issue of web application development focused on user interactivity. Such development includes the use of JavaScript frameworks. Original work is focused on evaluation of these frameworks based on selected criteria.

In the first part of original work, the issue of web application development itself was characterized and important technologies associated with development were described. At the end of this part, a method used to evaluate these frameworks was described in more detail. The method used was multi-criteria decision analysis.

The second part was devoted to the framework evaluation. First, it was necessary to select appropriate JavaScript frameworks for said evaluation. The model of multi-criteria decision analysis was used to select such frameworks. After that, individual criteria were selected for the evaluation itself based on four aspects, namely performance, community, usability, development. Selected frameworks were evaluated with these criteria using the multi-criteria decision analysis model.

The findings gained from original work were used to analyse the differences and possible uses of selected frameworks.

Keywords: javascript, js, framework, web application, frontend, React, Vue, Angular, multi-criteria decision analysis

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

1 Úvod

Dnešní společnost je velice závislá na užívání moderních technologií usnadňující každodenní život. Takových technologií je velké množství, které se neustále zlepšují, a navíc přibývají další. Velice užívanou technologií je právě internetová síť, ke které je v současnosti připojeno téměř 40% světové populace[1].

Internet spojuje lidi po celém světě, a to především díky sociálním sítím, jako je Facebook, Instagram nebo Twitter. Na internetu se navíc nachází nespočet služeb, které společnosti usnadňují každodenní život. Může se například jednat o e-shopy jako je Amazon a Ebay nebo e-mailové klienty typu Gmail. Dnes jsou velice populární i služby zajišťující zábavu, mezi které patří YouTube a Netflix. Dokonce i většinu informací je člověk schopen nalézt přímo na internetu, a to s pomocí různých vyhledávačů, které mimo jiné odkazují na stránky jako je Wikipedia. Aby bylo možné takové služby společnosti nabídnout, je potřeba je před samotným užitím nejdříve vytvořit.

S touto problematikou právě souvisí vývoj webových aplikací. Existuje nespočet způsobů, jak takové služby (resp. aplikace) vyvinout. Pokud se jedná právě o aplikace, kde je koncovým uživatelem člověk, a ne nějaký stroj, měla by uživateli v nejlepším případě dodat pocit spokojenosti. Neměl by se při používání konkrétní aplikace cítit špatně nebo zaostale. Proto je poslední dobou právě kladen důraz na uživatelskou přívětivost a zážitek. Jedná se především o jednostránkové aplikace, které jsou právě založeny na interaktivitě s uživatelem. Uživateli je v rámci aplikace mimo jiné dodáván dostatek informací k ujištění fungování a celkového chodu aplikace. Měl by být informován o tom, co a jak se zrovna děje.

Řešení takových aplikací je dnes nejvíce realizováno pomocí jazyka JavaScript, kterým lze stránku (resp. aplikaci) upravovat bez potřeby opakovaného načítání. JavaScriptem je možné jednoduchým způsobem zařídit danou interaktivitu. Vývoj může být však velice složitý, a to hlavně když se jedná o velice složitou aplikaci. Psát kód od úplného začátku s čistým JavaScriptem je časově i nákladově nevýhodné. Proto je dnes běžné vyvíjet aplikace různými nástroji a dalšími pomocnými prostředky, které vývoj značně usnadňují. Kdyby tomu tak nebylo, neexistovalo by tolik webových aplikací.

[1] Dostupné na https://www.internetlivestats.com/internet-users

V oblasti jazyka JavaScript existuje nespočet takových pomocných prostředků ve formě různých nástrojů a knihoven. Kombinací těchto prostředků je možné vytvořit framework, který se mimo jiné zakládá na opakovaném používání běžných potřeb vývoje. Framework tedy poskytuje vývojáři rámec (resp. prostředí) pro vývoj, a to bez potřeby se zabývat základními náležitostmi. Vývojář se tak může rovnou zabývat implementací konkrétních potřeb vyvíjené aplikace. Množství těchto frameworků není malé a každým dnem se jejich počet zvyšuje. Proto bývá obtížné vybrat vhodnou variantu k vývoji webové aplikace. Jde především o problém rozhodování na základě různých potřeb, a to jak z pohledu aplikace, tak i z pohledu vývojáře.

Snahou diplomové práce je přiblížit problematiku vývoje webových aplikací psaných pomocí JavaScriptových frameworků. Cíl práce byl zaměřen právě na dané rozhodování mezi možnými variantami frameworků, kde hlavními zdroji informací byla doporučená literatura, odborné zdroje a názory odborníků z praxe.

2 Cíl práce a metodika

2.1 Cíle práce

Diplomová práce je tematicky zaměřena na problematiku vývoje webových aplikací pomocí JavaScriptových (JS) frameworků.

Hlavním cílem práce je analyzovat a zhodnotit vybrané JS frameworky vhodné pro vývoj webových aplikací.

Dílčí cíle práce jsou:

- charakterizovat problematiku vývoje webových aplikací,
- zhodnocení vybraných JS frameworků,
- analyzovat možnosti využití JS frameworků pro tvorbu webových aplikací.

2.2 Metodika

Teoretická část diplomové práce se bude zakládat na analýze a rešerši odborných zdrojů zaměřených na problematiku vývoje webových aplikací a JS frameworků.

V praktické části práce budou na základě poznatků zjištěných v analytické části zhodnoceny vybrané JS frameworky a formulovány možnosti využití jednotlivých frameworků pro vybrané typy webových aplikací.

Na základě syntézy teoretických a praktických poznatků budou zpracovány závěry diplomové práce.

3 Teoretická východiska

Kapitola se zabývá teoretickými východisky a rešerší odborných zdrojů. Dané poznatky napomáhají k lepšímu porozumění problematiky vývoje webových aplikací, a to především ohledně technologií související s jazykem JavaScript.

3.1 Programovací paradigma

Pojem programovací paradigma se nevztahuje na konkrétní jazyk, nýbrž pouze poukazuje na způsob psaní a strukturování kódu, což ve výsledku ovlivňuje myšlenkový pochod při vytváření dané aplikace. Existuje více typů paradigmat (Toal, a další, 2017), v rámci JavaScriptu jsou důležité především:

- procedurální (imperativní),
- objektově orientované (imperativní),
- funkcionální (deklarativní).

3.1.1 Imperativní programování

Procedurální a objektově orientované programování patří mezi tzv. *imperativní programování*, kde se vývojář zaměřuje na přesný postup instrukcí (algoritmus), jak se má změnit stav aplikace. U prvního způsobu se různé instrukce shlukují do procedur, což jsou jednotlivé části programu, které vykonávají specifický úkol. Jednou z nevýhod procedurálního programování je obtížnost znovupoužití kódu.

Objektově orientované programování se zaměřuje na shlukování dat do objektů, které se vyznačují vlastnostmi a metodami. S objekty například souvisí koncepty abstrakce, dědičnosti a zapouzdření, které napomáhají ke znovupoužitelnosti kódu. Zjednodušuje vývoj i napříč různými projekty (Toal, a další, 2017).

3.1.2 Funkcionální programování

Funkcionální programování spadá pod tzv. *deklarativní programování*, kde se vývojář zaměřuje na eliminaci (případně minimalizaci) vedlejších účinků. Znamená to, že se nepopisuje způsob vykonání, ale pouze výsledek, co se má udělat (Toal, a další, 2017).

U JavaScriptu se poslední dobou prosadilo právě funkcionální programování, které je založeno na kombinování různých funkcí a jejich kompozici, ačkoliv se nejedná o plně funkcionální jazyk.

Existuje několik zásadních pojmů, které souvisí s funkcionálním programováním (Toal, a další, 2017). Mimo jiné se jedná o:

- prvotřídní funkce (first class function),
- čisté funkce (pure functions),
- vedlejší účinky (side effects),
- neměnnost (immutability).

Prvotřídní funkce

Všechny funkce jsou brány jako prvotřídní funkce. Vyznačují se tím, že mají přístup k operacím jako obyčejné proměnné nebo objekty, což umožňuje vytvářet funkční kompozice (kombinování funkcí do sebe). Prvotřídní funkce mohou:

- být uloženy do proměnné,
- být předány jako argument jiným funkcím,
- být vráceny jinými funkcemi,
- být uloženy do datové struktury,
- mít vlastnosti a metody.

Čisté funkce

Jedná se o funkce, které za předpokladu použití stejných argumentů pokaždé vrací stejný výsledek a nedochází k žádným vedlejším účinkům, jako například změna stavu aplikace. Čisté funkce se podobají matematickým funkcím, což vede k přehlednosti a předvídatelnosti kódu.

Vedlejší účinky

Vedlejšími účinky se chápe jakákoliv interakce s okolím v rámci dané funkce. Například se jedná o změnu stavu v jiné části aplikace, vypisování textu nebo volání jiných funkcí, které samy mají vedlejší účinky. V ideálním světě by se měly úplně eliminovat, ale v mnoha případech je možné je pouze minimalizovat.

Neměnnost

Důležitý koncept nejenom u funkcionálního programování, který spočívá v neměnnosti datových struktur tak, že se data přiřazují pouze jednou a lze je pak už jenom číst. Například po jakékoliv inicializaci proměnné (resp. objektu) by se jejich stav neměl dál měnit. Proto je ideální využívat výše uvedené čisté funkce, které by tuto potřebu měly eliminovat.

3.2 Základní kameny webu

Webová stránka je v obecném pojetí pouze strukturovaný dokument, který může být obohacen grafickým vzhledem. Jedná se především o dvě technologie, bez kterých by se dnešní vývoj webových aplikací neobešel. Konkrétně jde o značkovací jazyk HTML a kaskádové styly CSS. Existuje mnoho moderních webových stránek, které jsou založeny pouze na těchto dvou technologiích. Může jít například o informační stránky firmy nebo fyzické osoby.

Nicméně dnes jsou webové aplikace tak rozšířené, že běžná informační stránka nestačí. Proto se často přistupuje k přidání dodatečného chování například pomocí JavaScriptu.

3.2.1 Hypertextový značkovací jazyk (HTML)

Hypertextový značkovací jazyk (HyperText Markup Language, HTML) je značkovací jazyk, který je využíván k tvorbě a strukturování webových stránek a jiných dokumentů. Slovo *hypertext* označuje způsob nelineárního strukturování textu, který využívá různé odkazy (W3Schools, 2020).

Kód napsaný v jazyce HTML je zpracováván klientem (prohlížečem), který při jeho načtení vytvoří objektový model dokumentu (Document Object Model), neboli strom objektů celého dokumentu. DOM je velice užitečný například i při práci s JavaScriptem.

Nejnovější verze HTML má označení HTML5.

3.2.2 Kaskádové styly (CSS)

Kaskádové styly (Cascading Style Sheets, CSS) jsou kolekcí metod pro úpravu vzhledu strukturovaného dokumentu HTML. Kaskádové se jim říká kvůli jejich podstatnému rysu – definice stylů lze zanořovat do sebe. Styly lze definovat v rámci samotného HTML dokumentu nebo externího souboru CSS (World Wide Web Consortium, 2018).

Dnes je navíc zvykem CSS psát pomocí preprocesorů jako je například Sass. Jedná se o nástroj, který rozšiřuje CSS jako takový. Představil zanořování, použití proměnných, podmínek, mixinů, funkcí a mnoho dalšího. Navíc je možné rozdělit stylování do více souborů a zkompilovat je do jednoho pomocí importů. Výsledný zkompilovaný soubor lze různě dokonce optimalizovat.

3.3 Architektury webových aplikací

Webové aplikace se od statických stránek dostaly velice daleko. Je potřeba vyvíjet čím dál víc složitějších systémů, které například musí umět pracovat s jinými systémy pomocí zvolených komunikačních kanálů (API). Ústřední myšlenkou architektur je tvorba co nejlépe udržovatelných aplikací s optimálním způsobem toků dat. Samotný návrh architektury je tedy jedním z nejdůležitějších kroků při vývoji, jinak by šlo o nadlidský úkol.

Aplikace mají různé potřeby a věnují se určité problematice, která by měla koncovým uživatelům nějakým způsobem pomáhat. Je potřeba si uvědomit pro koho (resp. co) je

aplikace určena. Když bude aplikace například pro uživatele, měla by se zaměřit především na přívětivost a zážitek (požitek) uživatele. Neměl by být nucen zbytečně čekat nebo přemýšlet během používání aplikace, a to bez ohledu na její obsah. V případě aplikace pro jiné systémy (například API) je zásadní výkon a výběr vhodných komunikačních kanálů ve formě koncových bodů, které umožní propojit jiné aplikace. Někteří vývojáři by už jen na základě této informace volili určitou architekturu. Nebo by výběr alespoň značně zúžili. Je potřeba podotknout, že architektura bývá často závislá na použitém frameworku. Existují frameworky, které jsou dnes vhodnější na vývoj uživatelských aplikací (frontend), a to jak z pohledu vývojáře (vývoj a správa), tak i z pohledu uživatele (přívětivost a zážitek). Jedná se o aplikace, které jsou zaměřeny na uživatelskou přívětivost a zážitek.

Správně definovaná architektura by měla rozdělit potřeby a odpovědnosti aplikací tak, aby se s nimi co nejlépe pracovalo, ale aby nezatěžovaly celkový chod aplikace. Obecně se v programování mluví o procesu oddělení zodpovědností (Separation of concerns). Jde o princip, kterým rozdělit aplikaci tak, aby se její části z hlediska funkcionality co nejméně překrývaly. Umožňovalo by to pak snadnou správu jednotlivých vrstev.

3.3.1 Vývoj architektur

Softwarové architektury sahají mnohem dál než samotný internet. Operační systémy nebyly tehdy tak vyspělé jako v dnešní době. Programy byly vyvíjeny na velice nízké úrovni (low level) a musely s hardwarem pracovat efektivně. Proto byla jejich architektura hodně závislá na dostupném hardware. Neustálý vývoj hardware zapříčinil různé změny v softwarové architektuře (Patil, 2019).

Přibližně od roku 1970, začaly být počítačové programy rozdělovány na vrstvy. Vrstvy měly za cíl efektivně oddělit různé závislosti a potřeby programu na daném systému. Jedním z prvních rozdělení se stala trojice vrstev MVC. MVC (Model-View-Controller) bylo poprvé představeno s jazykem Smalltalk-80. Šlo především o rozdělení, kde se pohledová vrstva (view) měla starat o zobrazení dat z vrstvy doménové (modelové). Modelová vrstva (model) tedy představovala vrstvu, kde byla data různě zpracovávána a transformována. O vzhled se starala pohledová vrstva (view). Řadič (controller) představoval prostředníka, který umožňoval a kontroloval komunikaci mezi pohledovou a modelovou vrstvou. Musel řídit věci, o které se dnes stará operační systém.

Postupem času nebylo MVC šikovné řešení. Jedním z problémů bylo to, že se model někdy staral o víc, než by měl. Modelová vrstva byla proto rozdělena na ještě jednu vrstvu, ve které se udržoval stav aplikace – aplikační model. Vrstvy pohledu a řadiče nekomunikovaly s modelem přímo (Mcheick, a další, 2011). Kdykoliv bylo potřeba přistoupit k datům, musely si na to prostřednictvím aplikačního modelu registrovat události. S modelem směla komunikovat pouze vrstva aplikačního modelu. Mluvilo se o tzv. AM MVC.

MVC s aplikačním modelem mělo nevýhodu v tom, že nová vrstva úplně odřízla model od ostatních vrstev. Na pohledovou vrstvu se začalo nahlížet jinak a část funkce řadiče převzal operační systém. Proto bylo roku 1995 představeno MVP (Model-View-Presenter), kde byl tradiční řadič zaměněn za tzv. *presenter*. Presenter dohlížel na prezentační logiku a mohl měnit pohledovou vrstvu. Pohled navíc nevolal žádný řadič, nýbrž pouze delegoval uživatelské události konkrétnímu presenteru (Khaliluzzaman, a další, 2016).

MPV byl i nadále hodně používán ale z pohledu uživatelských aplikací se nejslibnější architekturou stala MVVM (Model-View-ViewModel), někdy známá jako MVB (Model-View-Binder). Podobá se předchozí architektuře s tím rozdílem, že využívá konceptu provázání dat – *data binding*. Jedná se o programovací techniku, kterou lze datové zdroje spojit mezi poskytovatelem (provider) a spotřebitelem (consumer). Data jsou následně automaticky synchronizovány (Patil, 2019). Kód, který se tradičně staral o synchronizaci mezi modelem a pohledovou vrstvou, není již potřeba, což umožňuje vyšší úroveň abstrakce. *ViewModel* je tedy vrstva, ve které se nachází takové datové zdroje, které lze v rámci provázání dat použít. Konkrétně jde o objekt vystavující vlastnosti a metody pohledové vrstvě. Kdykoliv dojde ve ViewModelu ke změně, je v pohledové vrstvě automaticky provedená změna (synchronizace).

3.3.2 Základní webové architektury

Dnes lze za nejrozšířenější architekturu webových aplikací považovat MVC. Na první pohled to může být zvláštní, že zrovna jedna z těch nejstarších. Nicméně se nejedná o klasické MVC, ale o novější *webové MVC*. Původní MVC architektura byla navržena bez ohledu na potřeby webové aplikace. Aplikace jsou právě typické tím, že jsou rozdělené na dvě části – *frontend* a *backend*. První část je zpracovávána v prohlížeči (např. kliknutí tlačítka) u klienta a druhá část na serveru (např. obnova stránky).

První známou implementací MVC na straně serveru je JSP Model 2. Používá se u webových aplikací napsaných v jazyce Java. Až na pár výjimek je nejvíce podobný původnímu MVC. Obsahuje například tzv. *front controller*, který se například stará o *http* požadavky (Cavaness, 2004).

Nejvíce používanou architekturou v rámci webových aplikací je webové MVC s kombinací jazyka JavaScript (např. k validaci formulářů). S příchodem technologie AJAX se stalo dané řešení velice oblíbené. AJAX dokáže zpracovávat požadavky a překreslovat pouze potřebné části stránky (oproti znovunačtení stránky). Architektura nemá oficiální název, jelikož existuje mnoho možných implementací, které nešly pořádně standardizovat.

V neposlední řadě existují architektury, kde je aplikace napsaná především na straně klienta (frontend) – bohaté webové aplikace (Rich Web Applications). Hlavním rozdílem oproti webovému MVC je absence vrstvy řadiče a pohledu. Přesněji bývají naimplementovány jinak nebo se starají o mnohem míň než u MVC. Na straně serveru bývá API, pohled už jen zpracovává JSON a řadič se stará o mapování požadavků na vhodná místa (Patil, 2019).

V rámci těchto architektur byly představeny knihovny *Backbone.js* a *Knockout.js*, které se svým způsobem pokoušely o implementaci výše uvedených architektur. U prvního případu se jednalo o implementaci MV*, kde řadič chyběl. Druhá knihovna celkem úspěšně implementovala architekturu MVVM. Po čase však přišel na svět první frontendový framework *Angular 1*. Implementace architektury MVVM byla velice povedená. Přišel s vymoženostmi jako je například data binding, moduly, dependency injection. *Angular* pomohl komunitě pochopit důležitost architektur a zapříčinil vznik mnoha dalších frameworků.

3.3.3 Frontendové architektury

Návrh architektur byl dřív limitován hardwarem. Prostředí se však výrazně změnilo. Webové aplikace již nejsou omezeny osobním počítačem a lze je vyvíjet na celou řadu zařízení (chytré hodinky, mobilní telefony, tablety). JavaScript se vyvinul natolik, že s ním lze tvořit nativní mobilní aplikace s pomocí různých nástrojů. Uživatelé upřednostňují aplikace s co nejlepším uživatelským zážitkem (UX). Mobilní telefony se staly mnohem více používaným zařízením než osobní počítače.

Frontendové architektury se zakládají na plném využití uživatelského rozhraní (UI) a zážitku (UX). Je kladen větší důraz na datové toky, použití komponent a manipulace DOM. Není proto již možné hromadně kategorizovat architektury jako předtím. Frameworky se zakládají na své vlastní architektuře podle vlastních potřeb.

Dnes aplikace běží v různém prostředí a očekává se od nich vysoká interaktivita. Je tedy potřeba nahlížet na datové toky v rámci celé aplikace, a ne pouze v jeho částech. Pohled (UI) a stav se stávají důležitými aspekty frontendových aplikací.

Použitím komponent lze rozdělit aplikaci jiným způsobem než tradiční vrstvy. Vrstvy dělily aplikaci podle odpovědností horizontálně. S ohledem na datové toky se však stalo stěžejní dělit aplikace podle funkcionality (vertikálně), což horizontálním způsobem jde velice obtížně. Vertikální rozdělení umožňuje zabalit konkrétní funkcionalitu aplikace do tzv. komponenty, která se snadno spravuje a přenáší (Patil, 2019). Komponenta je ve své podstatě dobrou implementací architektury MVVM (resp. MV*). Dokáže si držet svůj stav (data, vzhled, animace), jelikož zapouzdřuje strukturu (HTML – View), vzhled (CSS – View), chování (JS – ViewModel) a byznys logiku (Model). Komponenty lze navíc skládat do sebe. Může jít například o speciální políčko ve formuláři (komponenta), které je součástí formuláře (komponenta).

Manipulace DOM je důležitá z pohledu interaktivity aplikace. Prvky by měly vykazovat známku života, aby uživatel věděl, že se něco stalo. Práce s DOM je však složitá a musí se dbát na spoustu detailů, aby například nedocházelo k různým výkonnostním problémům. Frameworky (resp. knihovny) se s problematikou manipulace vypořádávají různě. Způsob řešení lze shrnout tak, že je nejdříve DOM reprezentován pomocí funkce modelu. Následně je potřeba vytvořit mechanismus k detekci změn stavu aplikace. Nakonec vymyslet způsob aktualizace DOM podle zjištěných změn (např. VirtualDOM).

3.4 Základní rozdělení aplikací

Kapitola se zabývá popisem dvou základních typů aplikací, které se na internetu vyskytují. Jedná se o aplikace *jednostránkové* a *vícestránkové*.

3.4.1 Vícestránkové aplikace (MPA)

Na internetu jsou vícestránkové aplikace (Multi Page Application) velice běžné, jelikož se jedná o základní způsob vývoje webových aplikací, který sahá až k prvním implementacím webů.

Vícestránkové aplikace jsou založené na opakovaném načítání stránky při každé změně obsahu, což zahrnuje jak načítání, tak i odesílání dat. Jsou navíc vhodnější pro velkou uživatelskou základnu. Je například využíván u velkých firem a e-commerce, kde je kladen důraz na značnou interakci s uživateli a korespondujícími daty.

Za MPA lze například považovat většinu velkých e-shopů jako je například Amazon nebo Ebay, kterým jde především o viditelnost jejich produktů v rámci vyhledávácí optimalizace (SEO). V dnešní době je pro vyhledávače stále jednodušší procházení (resp. indexace) vícestránkových aplikací než procházení těch jednostránkových. Každá stránka může mít definovaná vlastní klíčová slova, popis a jiná metadata, která značně ovlivňují hodnocení vyhledávačů vedoucí k určité viditelnosti. Jedná se však o problém, který by měl být v budoucnu vyřešen.

Obvykle kvůli větší velikosti těchto aplikací je požadován přenos většího množství dat mezi prohlížečem a serverem, což například vede i ke zpomalení aplikace. Nicméně lze tento problém z části vyřešit použitím technologie AJAX (asynchronní JavaScript), která umožňuje načítání a překreslení pouze určité části aplikace (Powell, 2008). Řešení může být však složité, jelikož záleží i na tom, jakou technologií byla aplikace vyvinuta. Pokud byl využit jazyk PHP nebo Python k tvorbě daného webu, je navíc potřebná znalost i jazyka JavaScript.

Co se týče vývoje těchto aplikací, tak je v mnoha případech nutné vyvíjet *backend*, na který lze následně napojit *frontend*. Lze ovšem využít existující API, ale stále může dojít k potřebě rozhraní, kterým budou koncové body manipulovány. Čas strávený vývojem vícestránkových aplikací bývá delší než u jednostránkové implementace.

3.4.2 Jednostránkové aplikace (SPA)

Jednostránkové aplikace (Single Page Application) jsou relativně novým způsobem fungování webových aplikací. Jejich návrh spočívá v načtení veškerého obsahu již při vstupu na web. Není potřeba načítat celou stránku při každé potřebě změny obsahu (Mozilla, 2020).

Jednou z hlavních výhod jednostránkových aplikací je zaměření na uživatelské rozhraní a celkovou přívětivost, na kterou je dnes kladen největší důraz.

Mimo HTML a CSS je k vývoji jednostránkových aplikací potřebná pouze znalost JavaScript. Vývoj je navíc nezávisle rozdělen na *frontend* a *backend*. Na obou částech lze pracovat zároveň i zvlášť, jelikož *frontend* ve většině případů využívá koncové body API, který představuje *backend* a také poskytuje funkce a správu dat.

Čas strávený na vývoji těchto aplikací bývá nižší, jelikož často využívají již existující API.

3.5 Datová uložiště

Webové aplikace se zakládají na vlastní přidané hodnotě. Musí existovat důvod a účel jejich existence. Je proto stěžejní, aby daná aplikace určitým způsobem poskytovala, zpracovávala nebo jinak transformovala konkrétní data. Může se jednat i o jednoduché zobrazení dat, která byla určitým způsobem uspořádána k lepší přehlednosti.

Data (resp. množina dat) jsou tedy důležitým aspektem webových aplikací, bez kterých by se dnešní aplikace neobešly. Představují formalizované, fyzicky zaznamenané a opakovaně interpretovatelné údaje, které se týkají objektů reálného světa – události, myšlenky, pojmy nebo jiná fakta. S daty souvisí i datový typ, který určuje rozsah možných hodnot a množinu operací, které lze na daný záznam aplikovat (Vostrovský, 2014).

S datovými uložišti souvisí databáze, což je strukturovaná kolekce dat. V rámci databáze lze informace ukládat jako jednotlivé záznamy ukládání, na které se následně aplikují další operace. Typickými operacemi může být vyhledávání, filtrování, řazení a úprava záznamů.

3.5.1 Relační databáze

Relační databáze jsou soubory datových záznamů s předem definovanými vztahy a mají jasně definované schéma, které je ve většině případů standardizováno normalizačními formami (resp. pravidly). Normalizace klade důraz na atomicitu, integritu a konzistenci dat (Vostrovský, 2014).

Položky databáze jsou uspořádány do tabulek, což jsou kolekce dat, které mají jasně definované sloupce a řádky. Každý sloupec v tabulce obsahuje určitou doménu možných dat. Řádky naopak představují kolekci souvisejících hodnot jednoho objektu a bývají často označené jedinečným identifikátorem – primárním klíčem. Vztahy řádků napříč různými tabulkami lze realizovat pomocí cizích klíčů. Mezi relační databáze například patří:

- Oracle,
- Microsoft SQL server,
- MariaDB,
- MySQL,
- PostgreSQL.

3.5.2 NoSQL databáze

Jedná se o speciální typ databází, které nutně nevyužívají dotazovací jazyk, což vyplývá z anglické zkratky „Not only SQL“ (Cudré-Mauroux, a další, 2013). Hlavními znaky těchto databází jsou nerelační přístup, volné databázové schéma a horizontální škálovatelnost (sharding, rozdělení dat mezi servery). NoSQL databáze jsou nejvíce vhodné na práci v oblasti internetu věcí (IoT) a velkých dat. Existují čtyři přístupy NoSQL databází:

- databáze dvojic klíčů a hodnot,
- databáze dokumentů,
- databáze se širokým sloupcem,
- databáze grafů.

Dvojice klíčů a hodnot jsou jednodušším typem databáze, kde každá položka obsahuje klíče a hodnoty. Hodnotu lze obvykle načíst pouze odkazem na její klíč. Jsou vhodné v případech

potřeby uložení velkého množství dat, kde k jejich načtení není nutné provádět složité dotazy. Mezi běžné případy použití patří ukládání uživatelských předvoleb nebo ukládání do cache. Známější příklady tohoto přístupu jsou:

- Redis,
- DynanoDB.

Přístup databáze dokumentů je zaměřen na uspořádání data do dokumentů, které se podobají způsobu zápisu JSON. Každý dokument obsahuje dvojici polí a hodnot, které bývají různých typů – řetězce, čísla, pole objektů. Jejich struktury se obvykle shodují s objekty, se kterými vývojáři pracují v kódu. To je velký rozdíl ve srovnání s databázemi SQL. Mezi databáze dokumentů patří:

- MongoDB,
- CouchDB.

Databáze se širokým sloupcem, nebo jen sloupcové databáze, ukládají data do tabulek, řádků a dynamických sloupců, což poskytuje velkou flexibilitu nad relačními databázemi – každý řádek nemusí mít stejné sloupce. Přístup je vhodný při ukládání velkého množství dat, který je běžně užíván při práci s technologiemi internetu věcí (IoT). Velice známými implementacemi jsou:

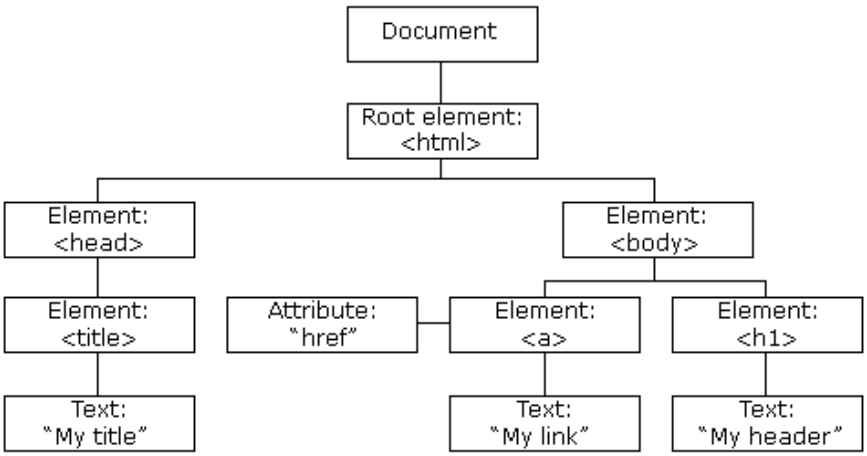
- Cassandra,
- HBase.

Databáze grafů ukládá data do uzlů a hran. Uzly obvykle obsahují informace o lidech, místech a věcech, zatímco hrany uchovávají informace o vztazích mezi uzly. Databáze grafů bývají vhodné v případech, kdy je potřeba procházet vztahy a hledat různé vzory, jako jsou sociální sítě, detekce podvodů a logistika. Mezi tyto databáze patří:

- Neo4j,
- JanusGraph.

3.6 Objektový model dokumentu (DOM)

Objektový model dokumentu (Document Object Model, DOM) je jazykově nezávislé rozhraní, které reprezentuje obsah dokumentu (stránky) strukturovaným způsobem v podobě stromu. Jednotlivé větve stromu jsou zakončeny uzly, které obsahují další objekty. DOM navíc poskytuje přístup k jednotlivým částem stromu pomocí metod. Metody umožnují různou manipulaci s DOM, což zahrnuje mazání nebo úpravy prvků webu, přiřazování různého chování atd. Na jednotlivé uzly je navíc možné navěšovat události, které při aktivaci mohou vykonat určitou úlohu.



Obrázek 1 – Objektový model dokumentu (W3Schools, 2020)

Z obrázku je zřejmé, že je DOM vytvářen od hlavního objektu dokumentu, který se dál větví na další objekty, které reprezentují jednotlivé HTML elementy. Vytvořené objekty představují ony uzly, kterým jsou přiděleny vlastnosti a metody k jejich manipulaci.

Na první pohled by se mohlo zdát, že je objektový model DOM reprezentován HTML kódem. Objektový model je však vytvořen právě až poté, co je HTML kód zpracován prohlížečem. Každý prohlížeč má svůj globální objekt, ve kterém se právě nachází daný dokument.

3.6.1 Standardizace DOM

Dřív měly webové prohlížeče vlastní tzv. přechodné DOM. Nekompatibilita různých řešení navedla mezinárodní konsorcium W3C ke standardizaci a vznikla specifikace W3C DOM, která je jazykově a platformě nezávislá. Specifikace byla rozdělena do několika na sebe navazujících úrovní (WHATWG, 2021):

- Úroveň 1 – celkový model, navigace v dokumentu a jeho manipulace
- Úroveň 2 – model událostí, jmenné prostory
- Úroveň 3 – zpracovávání událostí klávesnicí, rozhraní pro serializaci XML schémat
- Úroveň 4 – sloučení a zjednodušení předchozích standardů

3.6.2 Manipulace DOM

Manipulace DOM pomocí JavaScriptu není vždy snadná a přímočará. Ze začátku to nemusí být tak znát, ale s údržbou a rozvíjením aplikace dochází k problémům s výkonem. Proto přišly na svět dva základní koncepty manipulace DOM – Shadow DOM a Virtual DOM. Jedná se o nástroje, které byly vyvinuty s populárními frameworky jako jsou *Angular*, *React* a *Vue*, které se zakládají na komponentové tvorbě webových aplikací.

Stínový DOM

Stínový (shadow) DOM je nástroj pro snadnou manipulaci a stylování elementů (resp. komponent). Lze si jej představit jako část stromu (podstrom) nebo jako osobní DOM daného elementu. Hlavním rozdílem mezi obyčejným a stínovým modelem je způsob vytvoření a jak se chovají. Uzly se u obyčejného modelu skládají do sebe jako strom. U stínového modelu se rámcově (scoped) vytváří stínový strom (shadow tree), který je připojen k elementu, ale zároveň oddělen od potomků. Připojenému elementu se říká stínový hostitel (shadow host). Manipulace elementu (komponenty) v rámci stínového DOM je řešeno lokálně a je úplně izolované od zbytku aplikace – neobjeví se v obyčejném DOM (Casteleyn, a další, 2016). Zjednodušuje se tím i práce s kaskádovými styly tak, že je možné používat pouze jednoduché selektory a názvy tříd, a to bez složitého systému pojmenování.

Když je potřeba nějakou část aplikace překreslit, prohlížeč nemusí překreslovat úplně vše. Překreslí se pouze potřebné části, a ne celý DOM, což snižuje problémy s výkonem. Značná výhoda tkví právě v použití vývoje pomocí komponent, kde se na aplikaci nahlíží jako na seskupení samostatných komponent.

Virtuální DOM

Virtuální (virtual) DOM je koncept používaný *React* a *Vue*, kde se do paměti ukládá kopie DOM. Všechny změny dochází právě v této virtuální DOM, která prohlížeči zjistí, jakou část aplikace překreslit, aby se vyvarovalo zbytečnému překreslování. Funguje to tak, že nejdříve dojde k nějaké změně v rámci virtuálního DOM. Poté je virtuální DOM porovnáván s originálem pomocí operace *diffing*. Když jsou nalezeny změny, tak je prohlížeči jasné, které části je potřeba překreslit. Změn je možné dělat více najednou (Casteleyn, a další, 2016).

Rozdíly stínového a virtuálního DOM

Oba koncepty se zaměřují na problém s výkonem tak, že si vlastním způsobem vytváří nějakou dodatečnou instanci objektového modelu dokumentu. Virtuální DOM je přesnou kopií celého DOM a je následně porovnáván s originálem při jakékoliv změně. Stínový DOM je vytvářen po částech, kde každá část je izolována od zbytku, což zajišťuje snadnou práci s komponentami.

3.7 JavaScript

JavaScript je víceúčelový a multiplatformní skriptovací jazyk, jehož autorem je Brendan Eich. Jazyk byl vyvinutý v roce 1995 a byl značně ovlivněn programovacími jazyky C a Java. Název JavaScript byl zvolen pouze z marketingových důvodů a nijak nesouvisí s již zmíněným jazykem Java. Roku 1997 a 1998 byl jazyk standardizován asociací ECMA (European Computer Manufacturers Association) a ISO (International Organization for Standardization). Název standardizované verze je ECMAScript (Zakas, 2016).

Edice	Rok vydání	Název
1	1997	
2	1998	
3	1999	
4	2003	
5	2009	
5.1	2011	
6	2015	ECMAScript 2015 (ES2015)
7	2016	ECMAScript 2016 (ES2016)
8	2017	ECMAScript 2017 (ES2017)
9	2018	ECMAScript 2018 (ES2018)
10	2019	ECMAScript 2019 (ES2019)
11	2020	ECMAScript 2020 (ES2020)

Tabulka 1 – Vývoj verzí jazyka JavaScript (W3Schools, 2020)

3.7.1 Interpretace kódu

Kód napsaný v jazyce JavaScript je nejčastěji spouštěn v prohlížečích a je interpretován (prováděn) za běhu (runtime). V každém prohlížeči se nachází *engine* (česky „program“ nebo „stroj“), který umí procházet, zpracovávat a vyhodnocovat jednotlivé instrukce daného kódu.

Během spouštění se kód rozpadne na menší části, které se zpracovávají jednotlivě v rámci prováděcího zásobníku (execution stack, call stack). Každé této části se říká *execution*

context (česky „kontext provádění“), což je prostředí, ve kterém je daný kód spouštěn (Toal, a další, 2010).

Existují tři typy:

- global execution context (globální prováděcí kontext),
- function execution context (prováděcí kontext funkce),
- eval function execution context (prováděcí kontext funkce eval).

Globální prováděcí kontext

Jedná se o výchozí prováděcí kontext, kde se nachází globální objekt *Window* reprezentující otevřené okno (resp. kartu) prohlížeče. Kontext se vytváří ve dvou fázích – vytvářecí a prováděcí (Flanagan, 2002).

Ve vytvářecí fázi (creation phase) je v první řadě vytvořen globální objekt. Následně je alokována paměť pro potřebné proměnné a funkce, kde jsou definovány rámce působnosti (scope) a uzávěry (closure). Nakonec jsou inicializovány proměnné a funkce jsou vloženy do paměti. V prováděcí fázi (execution phase) se po řádcích zpracovává kód a proměnným se přidělují reálné hodnoty (Toal, a další, 2010).

Objekt okna obsahuje jak vlastnosti pro získávání informací v daném okně, tak i metody pro jeho manipulaci. Jednou z vlastností objektu okna je dokument, kde je možné najít spoustu vlastností a metod, které lze použít pro přístup k prvkům DOM. Javascript může přistupovat a provádět změny v objektovém modelu dokumentu. Dalším příkladem vlastnosti může být odkaz na jiný objekt (např. konzole), získání názvu okna apod. Metodami lze například zobrazit vyskakovací okno (alert) nebo vytisknout obsah v daném okně.

Prováděcí kontext funkce

Při každém volání funkce je dané funkci přiřazen nový funkční kontext. Kontext je vytvářen obdobně globální s tím rozdílem, že se nevytváří globální objekt, ale aktivační objekt, který s sebou navíc nese objekt argumentů a formální argumenty (hodnoty předané při volání funkce).

Prováděcí kontext funkce eval

Kód uvnitř funkce *eval* má také vlastní kontext. Dnes je však tato funkce zastaralá a nedoporučuje se používat z optimalizačních a bezpečnostních důvodů.

3.7.2 Rámce působnosti a uzávěry

Rámec působnosti (scope) je rozsah viditelnosti proměnných v dané části kódu. Existují globální a lokální rámce. Globální rámec působnosti je pouze jeden a je reprezentován s výše zmíněným globálním objektem okna. Patří do něj všechny proměnné, které nebyly deklarovány ve funkci nebo jiném bloku kódu. Lokální rámce jsou dvojího typu – rámec funkce a rámec bloku. Do funkčního rámce patří proměnné deklarované klíčem *var,* do blokového patří ty, které byly deklarované klíči *let* nebo *const.* Při zanořování funkcí se navíc zanořují i jejich rámce, čímž se vytváří hierarchie zvaná řetězec rámců (scope chain).

S problematikou hierarchie souvisí ještě lexikální rámce, které určují předky a potomky volaných funkcí. Jeli například definována funkce A mimo jinou funkci B a funkce A je následně volána ve funkci B, tak funkce A nemá přístup do rámce působnosti funkce B, protože funkce A je lexikálně mimo funkci B (Toal, a další, 2010).

Uzávěr (closure) souvisí se zanořováním a je to označení funkce s takovým rámcem působnosti, ve kterém byla definována – vnitřní funkce má přístup do rámce působnosti vnější funkce. Funguje to především díky lexikálnímu rámci a skutečnosti, že funkce jsou prvotřídními objekty. Což je užitečné, když je například potřeba předávat funkci jako argument nebo si ji uložit do proměnné (Flanagan, 2002).

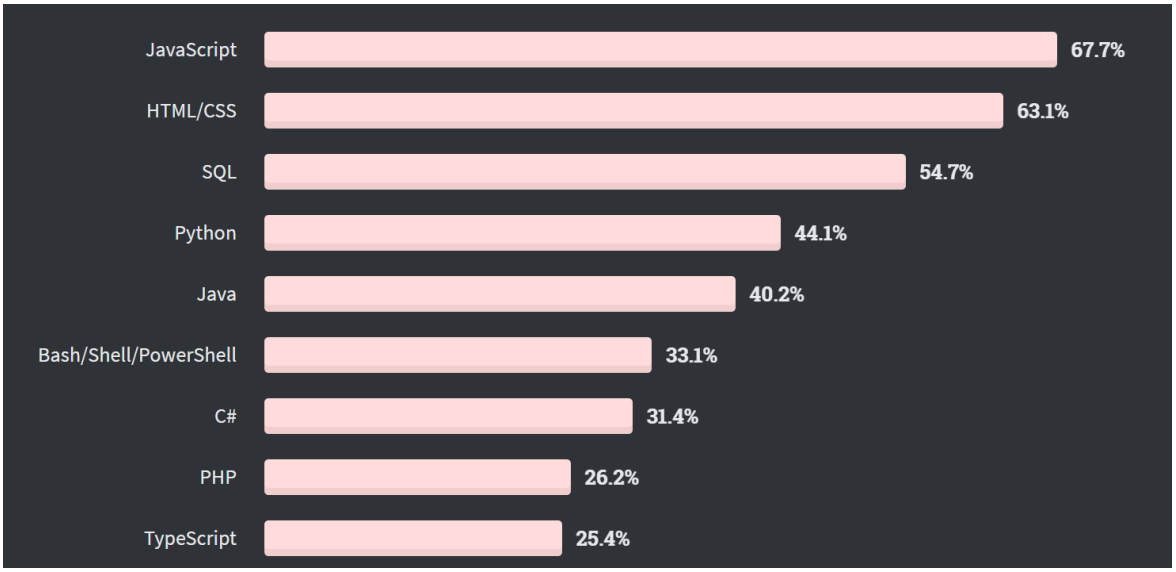
3.7.3 Popularita JavaScriptu

JavaScript si vždy držel místo pro své využití, ať se jednalo o jednoduchou manipulaci s formuláři, hry nebo komplexnější webové aplikace. V roce 2009 byl představen softwarový systém Node.js, který umožňoval vytvářet aplikace na straně serveru, což představovalo velikou změnu ve vývoji aplikací. Dále byly vydány různé frameworky a knihovy jako *Angular, React, Vue*, Express.js, které se postupem času staly nedílnou součástí vývoje webových aplikací.

Nicméně až teprve s příchodem nových verzí ECMAScript a následnou pravidelnou roční aktualizací začal jazyk jako takový značně nabírat na popularitě. Každá nová verze přinesla zásadní vylepšení a změny jako například deklarace tříd, použití modulů, asynchronní kód a mnoho dalšího. Díky tomu nebyly některé knihovny již potřeba.

Dnes se JavaScript rozšířil i do odvětví mimo webových aplikací, jako je například mobilní vývoj napříč platformami nebo internet věcí (IoT).

V roce 2020 byl navíc proveden rozsáhlý průzkum webem StackOverflow, který byl zaměřen na vývojáře po celém světě. Bylo zjištěno, že nejvíce populární technologií za daný rok byl právě JavaScript (viz Obrázek 2). JavaScript si udržuje první místo již osmým rokem v daném průzkumu.



Obrázek 2 – Nejpopulárnější technologie průzkumu (Stack Overflow , 2020)

3.7.4 Výhody a nevýhody JavaScriptu

Jedna z výhod jazyka je podpora spouštění kódu ve všech moderních prohlížečích, a to na jakékoliv platformě. Díky jednoduchosti a relativně mírné křivce učení je často vybírán jako první jazyk na naučení. Lze s ním například implementovat automatické doplňování a kontrolu prvků formulářů nebo jednoduché hry. Ke spuštění kódu je potřeba pouze prohlížeč – netřeba kompilace kódu jako u jazyka C. Další výhodou je neustálá aktualizace jazyka a jeho obrovská komunita, ze které vychází mnoho užitečných open-source knihoven a frameworků. Příchod Node.js přineslo možnost vytvářet JS aplikace na straně serveru, což bylo dříve nemožné.

Mezi nevýhody patří například skutečnost, že je obtížné vyvíjet veliké a komplexní aplikace. Kdokoliv se může podívat do zdrojového kódu, což zvyšuje potřebu dbát na bezpečnost. Některé prohlížeče můžou interpretovat určité části JavaScriptu jinak než ostatní, ačkoliv ne o tolik. JavaScript je slabě typovaný, což může zhoršit přehlednost a vývoj kódu v týmu. Jelikož je kód spouštěn na straně klienta, tak se jakákoliv chyba nebo nedopatření zobrazí veřejnosti, což může vést k dalším bezpečnostním problémům. Za nevýhodu lze považovat i to, že uživatel může na straně klienta JavaScript úplně vypnout, a to i na mobilních zařízeních. Může se například jednat o zastavení za účelem zastavení měřících nástrojů nebo zobrazování reklam.

3.8 Technologie spojené s JavaScriptem

S jazykem JavaScript je spojena spousta technologií, které značně ulehčují vývoj webových aplikací. Nejvíce užívané je prostředí Node.js a správa balíčků nástrojem *npm*. a typová nadstavba (nadmnožina) JavaScriptu.

3.8.1 TypeScript

TypeScript je moderní obdobou jazyka Javascript vyvinutý a udržovaný společností Microsoft. Jedná se o nadstavbu nebo nadmnožinu jazyka JavaScript, kde se daný kód pomocí Node.js transpiluje do běžného JS kódu. Transpilace kódu spočívá v převedení kódu z jednoho jazyka do druhého, ale s tím rozdílem, že oba jazyky mají stejnou úroveň abstrakce.

Značným rozdílem oproti JavaScriptu je ten, že je TypeScript založen na využívání technik, které vycházejí z objektově orientovaného přístupu, které JavaScript nativně nepodporuje. V rámci tohoto jazyka je možné definovat typy a rozhraní. Další novinky zahrnují výčtové typy, mixiny, genericita, moduly, zkrácená syntaxe pro anonymní funkce (Microsoft, 2020).

3.8.2 Node.js a npm

Node.js je open-source, multiplatformní a runtime prostředí pro JavaScript, který dokáže spouštět JavaScriptový kód mimo webový prohlížeč. Umožňuje vývojářům používat

JavaScript k psaní aplikací a nástrojů na straně serveru. Je postaven na JavaScript *enginu Chrome V8*, který využívá i prohlížeč Google Chrome (Node.js, 2020).

V rámci vývoje s Node.js je značně používána správa balíčků jeho výchozím nástrojem *npm* (Node Package Manager), kterým lze jednoduše spravovat knihovny a nástroje v libovolném projektu. Nástroj umožňuje instalovat, aktualizovat a mazat balíčky s ohledem na jejich závislosti. Vývojáři to značně ulehčí práci, jelikož mu je nástroj schopný například zjistit, které závislosti (resp. balíčky) jsou již neudržované.

3.9 Framework a knihovny

Vývojáři si s oblibou ulehčují práci psaním vlastních pomůcek, které jim pomáhají v psaní kódu napříč více projekty. U některých případů jsou dané pomůcky zpřístupněny ostatním ve formě knihoven a frameworků. Není totiž nutné znovu vynalézat kolo, když se o to někdo už postaral. Proto je tvorba úplně od základů s dnešní technologií a nástroji nevídaná záležitost. Stojí to navíc zbytečně čas i peníze, které by se daly využít lepším způsobem.

Za téměř každým frameworkem (resp. knihovnou) stojí určitá komunita, která je stěžejní pro existenci daného frameworku (resp. knihovnu). Komunita usiluje o řešení různých problémů a neustálé vylepšování například ve formě nových nápadů. U oblíbených implementací se komunita dokáže rozrůst natolik, že je daný nástroj populární natolik, že je používán i ve větších firmách. Což nese například výhodu možného náboru nových zaměstnanců do firmy na základě znalostí daného nástroje. Není třeba vynakládat úsilí ohledně zaučování nových pracovníků.

Využívání frameworků a knihoven přináší mnoho výhod, ale i možných nevýhod. Za nevýhodu se dá považovat konečná velikost dokončené aplikace. Framework může obsahovat nespočet pomocného kódu, který není ve výsledku používán. Další nevýhodou je nutnost se s daným nástrojem naučit a přijmout koncepty jejich autorů. Jedná se tedy o nástroje vytvořené za účelem zjednodušení psaní aplikací a případné standardizace vývoje.

Framework oproti knihovně převrací kontrolu aplikace tak, že udává vývojáři rámec nebo prostor k vytváření aplikace. Je také závislý na architektuře. Knihovna je na druhou stranu pomůckou, která nemá definovanou architekturu. Je to například kolekce nástrojů, které lze libovolně využívat kdekoliv a kdykoliv v daném kódu. Frameworky se často skládají z různých knihoven, které mohou být na sobě nezávislé.

Existují však případy, kde je komunitou konkrétní knihovna brána jako framework. Jedním z možných důvodů je ten, že si vývojář může ke konkrétní knihovně připojit spoustu jiných knihoven. Možným výsledkem je například jednoduchý framework.

3.10 Průzkumy a studie

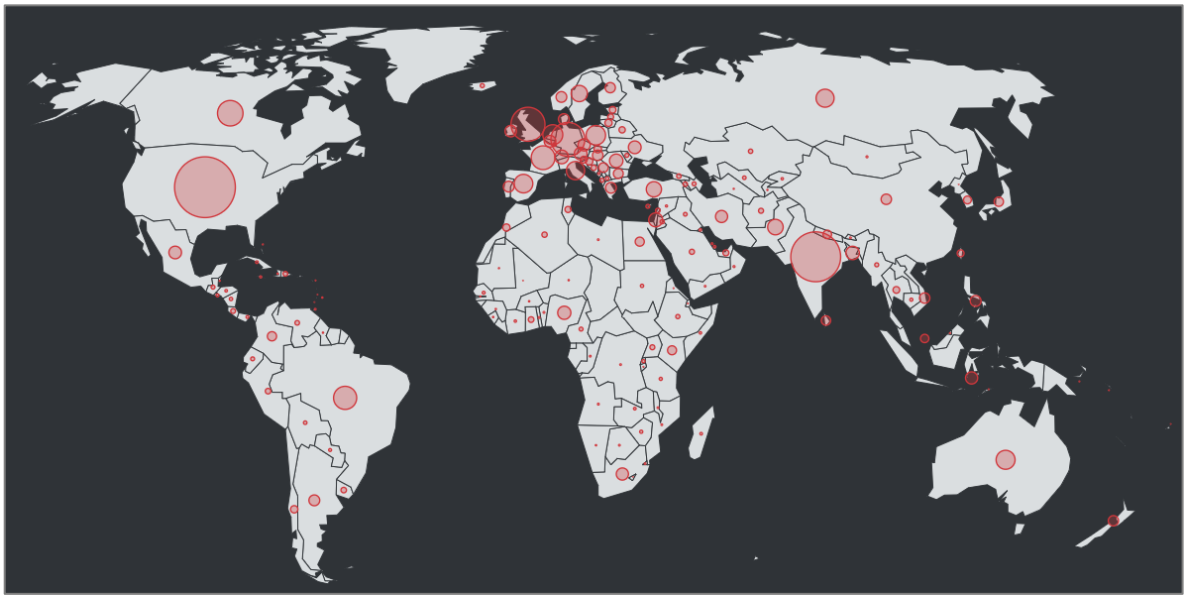
Oddíl provedených průzkumů pojednává o dvou vybraných průzkumech. Výsledky a poznatky získané z těchto průzkumů měly určitý vliv na zpracování této diplomové práce.

3.10.1 Stack Overflow Developer Survey 2020

Roku 2020 proběhl velice rozsáhlý průzkum webem Stack Overflow, který byl zaměřen na vývojáře celého světa. Odpovědělo téměř 65 tisíc respondentů. Konkrétně průzkum probíhal mezi 5. a 28. únorem (Stack Overflow, 2020).

V první části průzkumu byly zjišťovány obecné informace o respondentech (Stack Overflow, 2020), které zahrnovaly otázky ohledně:

- pracovních rolí,
- dosavadních zkušeností,
- dosaženého vzdělání.



Obrázek 3 – Demografie průzkumu Stack Overflow (Stack Overflow, 2020)

Druhá část průzkumu se zabývala technologiemi (Stack Overflow, 2020), kde se otázky týkaly například:

- nejpopulárnějších technologií,
- nejvíce oblíbených, žádaných a chtěných technologií,
- nejvíce placené technologie.

Dále byli respondenti dotazováni na otázky týkající se trhu práce (Stack Overflow, 2020). Bylo zkoumáno:

- informace o zaměstnání,
- jaké má respondent kariérní hodnoty,
- jestli hledají práci,
- jaké jsou jejich pracovní priority,
- jaká je jejich roční mzda.

Poznatky z této části byly jednou z motivací pro tuto diplomovou práci. Především se jednalo o pracovní priority vývojářů. Bylo zjištěno, že 51,3% vývojářů považuje za nejdůležitější to, v jakém jazyce, frameworku a jiných technologiích bude v daném zaměstnání pracovat (viz Příloha 15). Jedná se o velice důležitý aspekt, který by měli brát v potaz například manažeři nebo vedoucí týmů vývojářů. Je stěžejní, aby zaměstnanci nebyli nuceni pracovat v něčem, co je například nepoužívané, zbytečně složité a špatně zdokumentované. A to především, pokud existují jiné možnosti, mezi kterými lze *rozhodovat*. Zaměstnanci se jinak mohou dostat do takového stavu, že z dané společnosti raději přejdou do jiné. To by v některých případech mohlo přinést špatné následky, jelikož byla společnost připravena o pracovní sílu, která by mohla místo odchodu přinést užitek a celkový výnos dané společnosti.

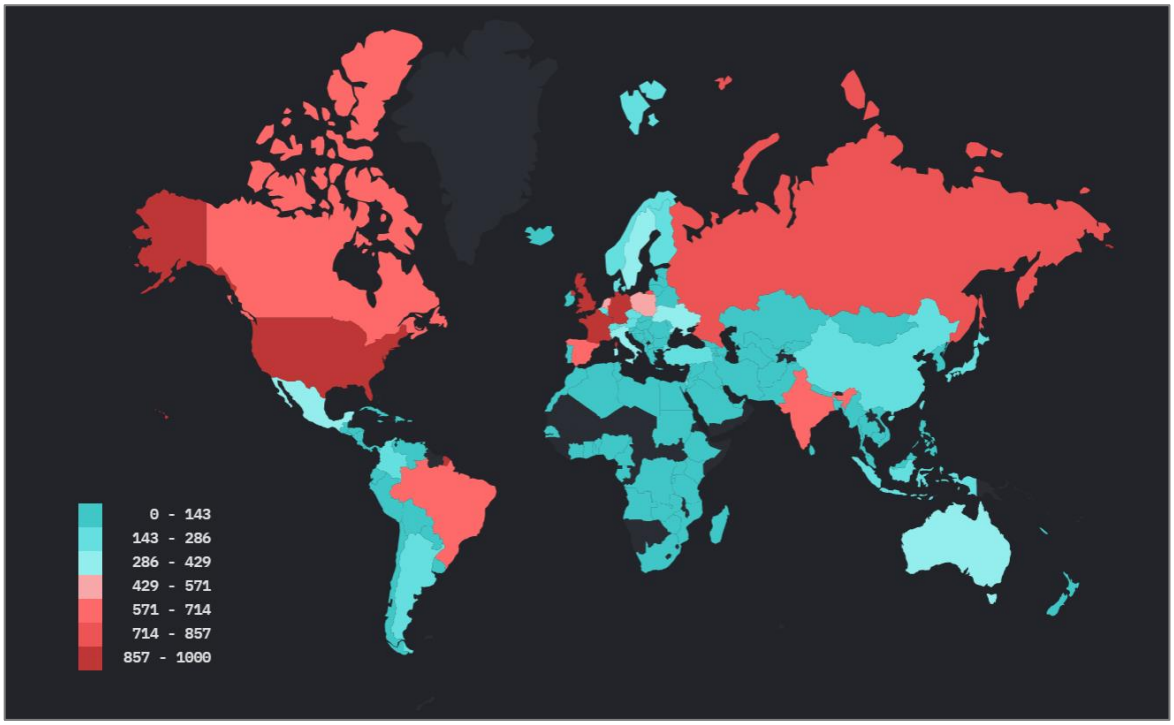
Jako poslední zkoumanou částí byla komunita Stack Overflow jako celek (Stack Overflow, 2020). Tady bylo zkoumáno:

- jak často respondent používá portál Stack Overflow,
- jak se často účastní daného průzkumu.

Poznatky z této části byly také velice přínosné při zpracování diplomové práce, a to především díky názorům, jak respondenti často využívají daný portál.

3.10.2 State of JS 2020

Od roku 2016 probíhá každý rok průzkum zaměřený na technologie spojené s jazykem JavaScript. Na daný dotazník odpovídá každý rok velké množství respondentů, kde v roce 2020 na něj odpovědělo 23 765 respondentů. Průzkum je navíc cílený na vývojáře po celém světě a konkrétně se roku 2020 získaly odpovědi ze 137 zemí (Greif, a další, 2020).



Obrázek 4 – Demografie průzkumu State of JS 2020 (Greif, a další, 2020)

V rámci průzkumu byly zjištěny názory vývojářů na různá témata (sekce) související s jazykem JavaScript. Byly zjišťovány především preference různých technologií. Šlo například o frontendové frameworky, datové vrstvy, testovací nástroje, vývojové nástroje a knihovny.

U každé sekce měli respondenti na výběr z pěti odpovědí:

- „Nepoužil(a) bych znovu“
- „Nezajímám se“
- „Použil(a) bych znovu“
- „Chci se naučit“
- „Neznám“

Na základě získaných odpovědí mohly být u jednotlivých sekcí vyjádřeny míry spokojenosti, zájmu, použití a známost daných technologií, které byly následně graficky vyjádřeny.

Míra spokojenosti byla vyjádřena jako podíl respondentů, kteří by daný framework znovu použili a suma těch, kteří již danou technologii někdy v životě použili. Zájem byl vyjádřen jako podíl respondentů, kteří by se rádi danou technologii naučili a suma těch, kteří s danou technologií ještě nepracovali. Míra použití je vyjádřena jako podíl sumy respondentů, kteří mají s danou technologií zkušenost a sumou všech odpovědí dané sekce. Známost technologie byla vyjádřena jako podíl respondentů, kteří danou technologii znají nebo s ní pracovali a sumou všech odpovědí dané sekce.

V rámci sekce frontendových frameworků byly vyjádřeny míry pouze u devíti frameworků, které měly míru známosti větší jak 10%. V následující tabulce (viz Tabulka 2) byly shrnuty výsledky za rok 2020 (Greif, a další, 2020).

	Míra spokojenosti	Míra zájmu	Míra užití	Míra známosti
React	88%	58%	80%	100%
Angular	42%	21%	56%	100%
Vue	85%	63%	49%	99%
Ember	27%	16%	11%	88%
Svelte	89%	66%	15%	86%
Preact	78%	40%	13%	77%
Stimulus	67%	27%	1%	34%
Alpine	82%	38%	3%	27%
LitElement	78%	39%	5%	17%

Tabulka 2 – Frontendové frameworky (State of JS 2020)

Z výsledků je patrné, že nejvíce známé a nejvíce používané byly za daný rok *React*, *Angular* a *Vue*. Vývojáři byli nejvíce spokojeni s prací se Svelte, který je jednoduchý na použití a přináší relativně nový způsob vývoje JavaScriptových aplikací. Je o něj také největší zájem. Za ním hned následují již velice známé frameworky *React* a *Vue*. S větším předstihem byl nejznámější *React*, který je navíc nejvíce užívaný společně s *Vue* a paradoxně i *Angular*. S *Angularem* byla nejmenší spokojenost, což zřejmě zapříčinila jeho možná složitost.

3.11 Vícekriteriální analýza variant

Ve světě je rozhodování jednou z nejběžnějších činností lidského chování a nejedná se vůbec o nic nového. Lidé se odjakživa nad něčím rozhodují podle nějakých hledisek a podmínek. Může se jednat o jednodušší věci, jako výběr šatů na taneční bál nebo výběr auta. Rozhodování bývá proto velice individuální. Záleží především na rozhodovateli, jak vidí svět kolem sebe a jaké má preference, podle kterých se rozhoduje.

Rozhodování je navíc jedna z nejvýznamnějších činností manažerů, jelikož se na něm zakládá řízení týmů nebo organizace. Kvalita rozhodování navíc ovlivňuje chod organizace tak, že má vliv na její úspěch.

Vícekriteriální analýza variant je tedy metodika, která je založena na matematickém modelování rozhodování a zabývá se vyhodnocováním vyhovujících možností. Rozhodování v rámci této analýzy spočívá ve výběru varianty, která je z pohledu určených podmínek nejvíce vyhovující. Je ovšem možné vybrat i více možností. Někdy pouze stačí, aby dané možnosti byly přípustné tak, že splňují všechny definované podmínky rozhodování, ačkoliv ne všechny musí být splněny ve stejné míře. Je totiž možné, že některé varianty vyhovují více než zbylé, ale všechny jsou přípustné (Triantaphyllou, 2000).

Obecně je postup založen na předběžném výběru variant a určení kritérií sloužících k hodnocení daných variant. Kritériím je potřeba určit i váhu, která je nezbytná při určování vyhovující varianty. Ta může být buď stejná u všech kritérií, nebo se může daná váha mezi kritérii lišit. Proto je možné volit z různých typů metod, které slouží k jejich určení.

Rozhodovací proces je založen na rozhodovacím subjektu, objektu a cílech rozhodování, kritérií a variant hodnocení (Fiala, a další, 1994). Subjektem se rozumí právě onen rozhodovatel, který volí danou variantu v rámci rozhodování. Objekt rozhodování vymezuje oblast, ve kterém je analýza prováděna. Cílem se rozumí stav, kterého je potřeba dosáhnout a nemusí být definován pouze jeden. Kritéria slouží k posouzení jednotlivých variant a rozlišují se maximalizační a minimalizační kritéria. Varianty jsou možnosti, mezi kterými subjekt rozhoduje.

K modelování vícekriteriální analýzy je tedy potřeba určit oblast nebo téma, co je potřeba analyzovat. Dále je třeba určit cíle, kterých je potřeba dosáhnout a také jaké podmínky se mají dodržovat. V dalším kroku je stěžejní uvést hlediska rozhodování, které vymezují obecný rámec, podle čeho rozhodovatel rozhoduje a neměl by zahrnovat skutečnosti, které

nesouvisí s danými hledisky. Je také potřeba určit i časový horizont, ve kterém bude rozhodnutí působit (Šubrt, 2019).

V rámci vícekriteriální analýzy jsou definovány základní typy variant (řešení), a to:

- ideální,
- bazální,
- dominantní,
- paretovské,
- kompromisní.

Ideální variantu H s ohodnocením (h1, …, h2) reprezentuje hypotetické nebo skutečné řešení, které je hodnoceno nejlepšími možnými hodnotami daných kritérií. *Bazální* varianta D s ohodnocením (d1, …, d2) má naopak nejhorší možné hodnocení. *Dominantní* varianta je za předpokladu maximalizačních kritérií takové řešení, které je v porovnání s jinou variantou lepší v rámci ohodnocení. Řešení může být nedominované, pokud neexistuje jiná varianta, která by tomuto řešení dominovala. *Paretovská* varianta je řešení, které je nedominované a je taktéž nazýváno jako efektivní. *Kompromisní* varianta je podle dané hodnoty od ideální varianty nejméně vzdálená.

Kritéria můžou nabývat různých hodnot. Může jít o informaci:

- nominální,
- ordinální,
- kardinální

Nominální informace o důležitosti mezi kritérii je vyjádřena pomocí aspiračních úrovní. Aspirační úrovně jsou založeny na stanovení nejhorších možných hodnot kritérií, podle kterých je možné jednotlivé varianty přijmout nebo odmítnout. Ordinální informace vypovídá o kvalitativním charakteru preference (důležitosti) kritérií nebo pořadí samotných variant. Na druhé straně může být informace také kardinální neboli kvantitativního charakteru. Jedná se o číselné vyjádření důležitosti kritérií nebo hodnocení variant, kde je možné vyjádřit i míru rozestupu.

3.11.1 Metody stanovení vah kritérií

U vícekriteriální analýzy variant je potřeba předně určit váhy kritérií. Váhy kritérií vyjadřují relativní důležitost v rámci všech ostatních kritérií. Jejich hodnoty spadají do intervalu $< 0; 1 >$, kde suma těchto vah je rovna 1. Výběr těchto vah je stěžejní pro jednoduché a jasné ohodnocení variant. Nejlépe se pracuje s kritérii, která jsou kvantitativního charakteru, jelikož se hodnocení na základě kvalitativní informace může někdy velice lišit podle toho, kdo je subjektem rozhodování (Brožová, a další, 2014).

Stanovit váhy kritérií je možné více způsoby. V nejjednodušším případě lze váhy určit přímo, a to v případě, když neexistuje možnost, jak určit jejich důležitost. Pokud tedy není možné určit jejich důležitost, je vhodné váhy kritérií nechat stejné. Pakliže mají kritéria mít různě definovanou důležitost, lze k tomuto problému přistoupit srovnáváním kritérií ordinálním nebo kardinálním způsobem (Šubrt, 2019).

Mezi ordinální srovnávací metody patří:

- metoda pořadí,
- Fullerova metoda.

Metoda pořadí je založena na srovnávání všech kritérií najednou. U Fullerovy metody jsou naopak kritéria srovnávána po párech. Dané metody jsou vhodné, pokud má rozhodovatel o kritériích ordinální informaci. To znamená, že je schopen seřadit kritéria podle důležitosti (Šubrt, 2019).

Mezi kardinálně srovnávací metody patří:

- bodovací metoda,
- Saatyho metoda

Bodovací metoda srovnává všechna kritéria najednou, jako tomu bylo u metody pořadí. Saatyho metodou se provádí párové srovnávání. Metody jsou založené na kardinálních informacích kritérií, kde je rozhodovatel schopen určit jak pořadí, tak i rozestupy mezi danými kritérii (Brožová, a další, 2014).

3.11.2 Metody rozhodování

Metody určené k rozhodování jsou založeny na různé metodice. Může se například jednat o vyhodnocování preferenční relace nebo mezní míry substituce jednotlivých variant. Je také možné u každé varianty pomocí vah a kritérií vyjádřit míru jejich užitku, které spadají do intervalu $< 0; 1 >$. Tato míra udává, jaký užitek varianta přináší z pohledu vybraných kritérií a jejich vah.

Modely analýzy vícekriteriálního rozhodování mají různé cíle, jako jsou například:

- nalezení kompromisní varianty,
- rozdělení řešení na efektivní a neefektivní,
- uspořádání od nejlepšího k nejhoršímu.

Na základě daného cíle a dostupných informací lze vybírat metodu (Brožová, a další, 2014):

- bodovací metodu,
- metodu pořadí,
- metodu aspiračních úrovní,
- metodu váženého součtu.

Bodovací metoda (resp. metoda pořadí) je založena na bodování jednotlivých variant pořadovými čísly podle počtu variant a pevně dané stupnici, kde se body (resp. pořadí) sečtou. U této metody není navíc potřeba vah kritérií.

Metodou aspiračních úrovní lze připouštět varianty podle splnění aspiračních úrovní, které nejhoršími možnými hodnotami kritérií udávají rozmezí přijmutí nebo odmítnutí. Připouštění je řešeno konjunktivně, disjunktivně nebo iteračně.

Metoda váženého součtu je založena na maximalizaci užitku možných variant. Kde jsou tyto varianty podle užitku pořadově seřazeny. Na základě těchto hodnot lze varianty vybírat (Šubrt, 2019).

3.11.3 Zvolený model analýzy variant

Zvolený model má množinu přípustných řešení konečnou, kde každá varianta je ohodnocena podle určených kritérií.

Na základě rešerší odborných zdrojů byl k výběru vhodných frameworků a k jejich hodnocení zvolen model Saatyho metody a metody váženého součtu. Spojením těchto metod bude vyjádřen celkový užitek možných variant tak, že váhy budou určeny Saatyho metodou a celková analýza bude provedena metodou váženého součtu.

Saatyho metoda

Jedná se o metodu kvantitativního párového porovnávání, jehož autorem je významný profesor v oblasti rozhodovacích procesů Thomas L. Saaty.

V první řadě je nutné vytvořit matici párového porovnání $S = (s_{ij})$, kterou sestavuje buď přímo subjekt rozhodování, nebo je požádán například odborník z praxe. Hodnoty této Saatyho matice náleží do jednotné bodovací stupnice, kde je možno využít i mezistupňů (2, 4, 6, 8):

- 1 – rovnocenná kritéria *i* a *j*,
- 3 – slabě preferované kritérium *i* před *j*,
- 5 – silně preferované kritérium *i* před *j*,
- 7 – velmi silně preferované kritérium *i* před *j*,
- 9 – absolutně preferované kritérium *i* před *j*.

Jelikož jsou porovnávány právě dvojice kritérií *i* a *j*, je Saatyho matice čtvercová. Z toho důvodu jsou v matici na diagonále jedničky, které vyjadřují rovnocennost stejných kritérií. Matice je navíc reciproční, kde pro jednotlivé prvky platí vztah $s_{ij} = 1/s_{ij}$. Což znamená, že hodnoty matice jsou na druhé straně diagonály převrácené. To vypovídá o tom, že když je preferováno kritérium *i* před *j*, tak musí být stejná skutečnost vyjádřena převrácenou hodnotou na opačné straně matice (Saaty, 2000).

$$S = \begin{bmatrix} 1 & s_{12} & \dots & s_{1n} \\ 1/s_{12} & 1 & \dots & s_{2n} \\ \dots & \dots & \dots & \dots \\ 1/s_{1n} & 1/s_{2n} & \dots & 1 \end{bmatrix}$$

Protože jsou preference zadávány člověkem, tak často dochází k nekonzistenci preferencí daných kritérií. Je téměř nemožné se vyhnout nekonzistenci, a to především s velkým počtem kritérií. Z toho důvodu je potřeba ověřit i konzistenci dané matice pomocí poměru konzistence (Saaty, 2000).

Ke zjištění konzistence Saatyho matice je v první řadě potřeba určit index konzistence CI.

$$CI = \frac{\lambda_{min} - n}{n - 1}$$

kde λ_{min} je největší vlastní číslo matice a n je počet kritérií. Index konzistence lze následně využit k výpočtu poměru konzistence CR:

$$CR = \frac{CI}{RI}$$

kde CI je index konzistence a RI je náhodný index, který vyjadřuje předem známý průměr indexu konzistence podle počtu kritérií (viz Tabulka 3).

Počet kritérií	3	4	5	6	7	8	9	10
RI	0,52	0,89	1,11	1,25	1,35	1,40	1,45	1,49

Tabulka 3 – Hodnoty náhodných indexů (Saaty, a další, 2007)

Saatyho matice je považována za dostatečně konzistentní, pokud poměr konzistence nepřesahuje hodnotu 10%, což vyplývá z podmínky $CR < 0{,}1$, kterou profesor Saaty definoval (Saaty, 2000). Jeli poměr konzistence větší jak 10%, jsou její hodnoty srovnávání preferencí nespolehlivé k exaktnímu vyjádření vah kritérií.

Nakonec se pro každý *i*-tý řádek Saatyho matice vypočítá geometrický průměr G_i, který je následně využit k vyjádření jednotlivých vah v_i:

$$G_i = \sqrt[n]{\prod_{j=1}^{n} s_{ij}} \qquad v_i = \frac{G_i}{\sum_{i=1}^{n} G_i}$$

Vypočítané váhy lze dál využít ve zvolené metodě rozhodování, která spadá do vícekriteriální analýzy variant.

Metoda váženého součtu

U metody váženého součtu je výpočet založen na funkci užitku u, která přiřazuje jednotlivým variantám užitek. Hodnoty užitku spadají do intervalu $< 0; 1 >$, kde je hodnota 1 přiřazena právě té nejlepší variantě a hodnota 0 té nejhorší. Celkový užitek varianty V_i je vypočten pomocí dílčích užitků a předem určených vah (Šubrt, 2019).

K výpočtu dílčích užitků je potřeba dvou vektorů, a to:

- vektor nejhorších hodnot $(d_1, \dots, d_n)$,
- a vektor nejlepších hodnot $(h_1, \dots, h_n)$.

První vektor vyjadřuje bazální variantu s označením *D* a druhý vyjadřuje variantu ideální, která se značí H. Pomocí těchto vektorů lze vyjádřit dílčí užitek kritérií jednotlivých variant. Před sestavením těchto vektorů je kvůli maximalizační povaze metody váženého součtu potřeba převést minimalizační kritéria na kritéria maximalizační jednoduchým vzorcem:

$$y'_{ij} = max_i(y_{ij}) - y_{ij}$$

kde y_{ij} je původní hodnota kritéria dané varianty a $max_i(y_{ij})$ vyjadřuje maximální hodnotu daného kritéria napříč všemi variantami. Po převedení všech minimalizačních kritérií na maximalizační lze přejít k výpočtu dílčích užitků variant pomocí vzorce:

$$u_{ij} = \frac{y_{ij} - d_j}{h_j - d_j}$$

kde y_{ij} je konkrétní hodnota kritéria dané varianty, d_j je hodnota vektoru bazální varianty a h_j je hodnota vektoru ideální varianty. Výsledkem je normalizovaná kriteriální matice R, kde prvky nabývají hodnot intervalu $< 0; 1 >$. Z této matice lze následně určit celkový užitek varianty V_i, a to pomocí skalárního součinu vektorů vah a dílčích užitků varianty (resp. sumou vážených dílčích užitků):

$$u(V_i) = \vec{v} * \overrightarrow{u_i} = \sum_{j=1}^{n} v_j * u_{ij}$$

kde v_j je hodnota váhy vektoru $\vec{v}$ a u_{ij} je hodnota vektoru dílčích užitků $\overrightarrow{u_i}$.

Z výsledných celkových užitků variant je možné určit, která varianta má největší (resp. nejmenší) celkový užitek. Tudíž lze zjistit, která varianta je nejvhodnější v rámci daného rozhodování.

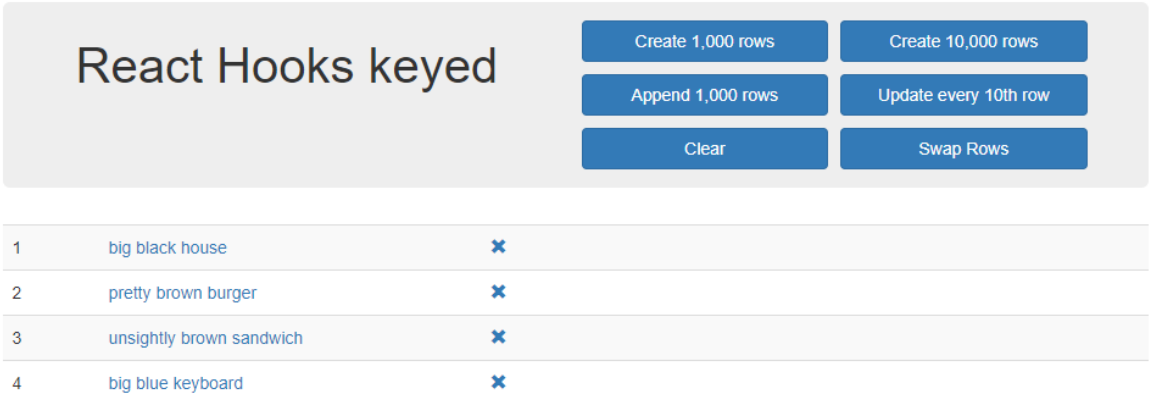
3.12 Použité nástroje měření výkonu

Měření výkonu je velice komplexní záležitost. Je potřeba, aby měřené aplikace v příslušných frameworcích byly funkčně stejné a držely se stejných pravidel. Je navíc vhodné, aby byly správně optimalizovány.

Aby mělo měření větší váhu, bylo využito existujícího projektu měření výkonu, na kterém se podílí velká část komunity. Princip spočívá v tom, že jsou vybrány konkrétní funkcionality a pravidla vývoje aplikací. Komunita na základě těchto informací dodává vlastní implementace ve vybraných frameworcích, které jsou kontrolovány a upravovány ostatními, aby byla zajištěna vhodnost a správnost implementace.

Použitý nástroj (resp. projekt) v rámci vlastní práce je *js-framework-benchmark*, se kterým přišel německý vývojář Stefan Krause. Nástroj je volně dostupný [2] pod licencí MIT a je postavený na zmíněném principu vytváření zkušebních aplikací. Díky definovaným pravidlům a funkcím je možné, aby kdokoliv z komunity přispěl svou implementací ve zvoleném frameworku. V době psaní této diplomové práce se na tomto projektu podílelo 161 vývojářů, mezi kterými lze najít i autory některých hodnocených frameworků.

Projekt je tak rozsáhlý, že je v něm možné nalézt více jak 100 implementací. Nástroj má navíc možnost vypsat výsledky měření vybraných frameworků ve formě HTML stránky. Díky úsilí komunity a některých autorů frameworků, je nástroj brán jako velice vhodný ke zjišťování výkonnostního aspektu vybraných frameworků.



Obrázek 5 – Ukázka React aplikace projektu js-framework-benchmark

[2] Dostupné na https://github.com/krausest/js-framework-benchmark

4 Vlastní práce

Vlastní práce se zabývá výběrem a následnou analýzou JS frameworků. Postupně bylo zpracováno hodnocení tří vybraných frameworků dle vybraných kritérií, které spadají pod určené aspekty.

4.1 Výběr frameworků ke zhodnocení

Vzhledem k vysoké popularitě jazyka JavaScript se každým rokem objevuje nespočet nových frameworků a knihoven. Z toho důvodu byly v rámci vlastní práce zohledněny takové frameworky, které splňovaly autorem definované podmínky, a to že frameworky jsou:

- dostatečně vyspělé,
- používané a osvědčené v praxi,
- stále udržované a aktivní,
- open-source.

Nebyly tedy zohledněny ani takové frameworky, které byly například vydané před pár měsíci.

S danými podmínkami souvisí právě komunita (resp. aktivita komunity), která je stěžejní pro existenci daného frameworku. Frameworky jsou běžně vytvářeny za účelem využití ostatními vývojáři, aby jim usnadnilo vytváření webových aplikací. V praxi se však neosvědčené a nevyvinuté frameworky nepoužívají. Je potřeba, aby se různí vývojáři podíleli na jejich vývoji ve formě kontribuce. Aby byla kontribuce vůbec možná, je zapotřebí, aby byl daný projekt licencován jako open-source. Ideální licencí je například MIT, která je z hlediska volného využití nejbenevolentnější. Různé poznatky a zpětná vazba jsou také důležité k postupnému vylepšování a potenciálnímu nasazení na produkci.

Na základě dat získaných z různých odborných zdrojů a názorů komunity ve formě průzkumu byl proveden výběr frameworků. S ohledem na doporučený rozsah diplomové práce a náročnosti byly ke zhodnocení nakonec vybrány pouze tři nejvíce vyhovující frameworky.

4.1.1 Předvýběr na základě průzkumu

Frameworků je ve světě nespočet a výběr pouze tří frameworků by byl velice obtížný i s předdefinovanými podmínkami. Proto bylo využito dotazníku *State of JS 2020*, kde byly zpracovány názory vývojářů na frontendové frameworky. Bylo detailněji zpracováno celkem devět frameworků, u kterých byla vyhodnocena míra známosti nad 10%.

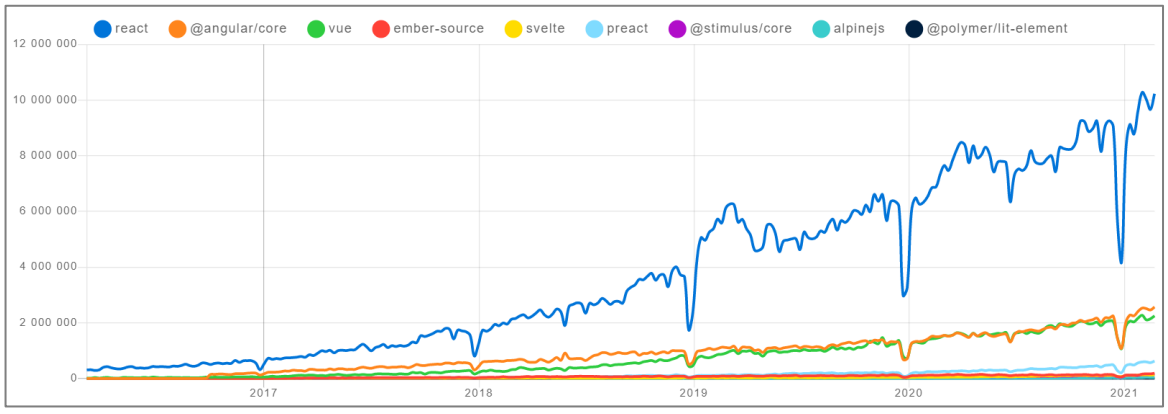
U zpracovaných frameworků bylo autorem dodatečně zjištěno, že se jedná o open-source projekty pod MIT licencí a jsou stále udržované. Vývojáři je do určité míry používají a nebyly vydané pouze před pár měsíci. Lze tedy konstatovat, že se průzkum zabýval takovými frameworky, které do jisté míry splňují autorem definované podmínky.

Předvýběr tedy zahrnuje frameworky *React*, *Angular*, *Vue*, *Ember*, *Svelte*, *Preact*, *Stimulus*, *Alpine* a *LitElement*.

4.1.2 Kritéria výběru frameworku

Vlastní práce byla zaměřena na zhodnocení pouze tří frameworků. Proto bylo potřeba s ohledem na předdefinované podmínky vybrat vhodná kritéria, kterými bylo následně možné zúžit daný výběr z devíti možností pouze na tři. K výběru kritérií (resp. frameworků) byly využity poznatky z průzkumu State of JS 2020 a výstupy z webového nástroje *npm trends*.

Prvním výstupem *npm trends* je časová řada (viz Obrázek 6), která vypovídá o vývoji počtu stahování daného frameworku za posledních pět let. Na vertikále je znázorněn počet stažení za každý týden a na horizontále je naopak znázorněn postup času.



Obrázek 6 – Vývoj počtu stažení (npm trends, 2021)

Na první pohled je patrné, že *React* je oproti ostatním nejvíce stahovaný a drží si své místo téměř od úplného začátku. Dalším zajímavým poznatkem je opakující se pokles přímo před koncem každého roku, což je zřejmě příčinou Vánoc, či jiných svátků, a Nového roku.

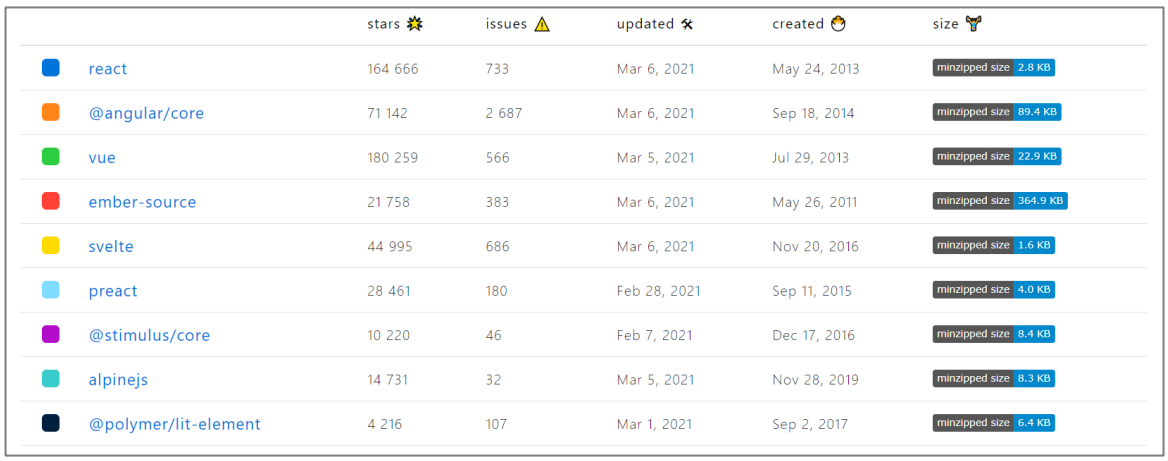
V grafu je možné vidět i náznak seskupení, a to především od roku 2019. První skupinu tvoří pouze *React*, druhou *Vue* s *Angularem* a zbytek frameworků tvoří třetí skupinu. Daná skutečnost je naznačena i ve výsledcích provedeného průzkumu *State of JS 2020*, kde je podobný rozdíl zřejmý v míře užití (viz Tabulka 2).

Poslední počty stažení z 21. února 2021 jsou zobrazeny na následující tabulce (viz Tabulka 4), kde jsou frameworky zleva doprava seřazeny od nejvíce stahovaného po nejméně stahovaného.

React	Angular	Vue	Preact	Ember	Stimulus	Svelte	Alpine	LitElement
10227002	2567681	2250073	625260	187440	133133	131977	36833	2205

Tabulka 4 – Počty stažení (npm trends, 2021)

Dalším výstupem nástroje *npm trends* je následující obrázek (viz Obrázek 7), kde jsou informace týkající se vyspělosti, použití, oblíbenosti, vyspělosti a udržovanosti daných frameworků.



Obrázek 7 – Základní informace (npm trends, 2021)

Vyspělost může být vyjádřena například rokem prvního vydání. Oblíbenost lze vyjádřit mnoha způsoby. V různých článcích a jiných odborných zdrojů je nejčastěji využívanou charakteristikou počet hvězd na GitHubu, počet stažení za týden nebo Google trendy. Byl proto vybrán počet hvězd a počet týdenního stažení (viz Tabulka 4).

Samotná délka existence však nemusí být ohledně vyspělosti dostatečně vypovídající. To samé platí pro počet hvězd z pohledu oblíbenosti. Je možné, že byl framework vytvořen kdysi dávno a začal být aktualizován až po několika letech. V případě počtu hvězd lze oblíbenost do jisté míry určit, ale u několik let starých frameworků nemusí mít velkou váhu. Když je framework používaný a oblíbený, naznačuje to jeho dobré provedení a může být považován za dostatečně vyspělý (resp. oblíbený). Proto byly zohledněny i výsledky z průzkumu *State of JS 2020*, kde byly konkrétně využity míry spokojenosti a míry užívání (viz Tabulka 2).

Na základě dostupných informací byla vybrána kritéria s ohledem na vyspělost, použití a oblíbenost, ktera navíc zohledňují autorem definované podmínky výběru. Kritéria výběru tedy zahrnují:

- rok vydání,
- počet hvězd na GitHubu,
- počet stažení za týden,
- míru spokojenosti,
- míru užití.

4.1.3 Model vícekriteriální analýzy variant

Ke konečnému výběru frameworků byla využita vícekriteriální analýza variant. Model analýzy se skládá z určení vah pomocí Saatyho metody a vyjádření celkových užitků možných variant na základě vybraných kritérií pomocí metody váženého součtu.

Vyjádření vah kritérií bylo prvním krokem vlastní vícekriteriální analýzy variant. Určení pomocí Saatyho metody je založeno na porovnávání dvojic kritérií buď subjektem rozhodování, nebo například expertem z praxe.

Na následující tabulce (viz Tabulka 5) jsou znázorněna kritéria s patřičnými zkratkami, kde bylo pro ujasnění uvedeno, jakých hodnot mohou nabývat.

Kritérium		Jednotka
Rok vydání	RV	rok
Počet hvězd	PH	kladné celé číslo
Počet stažení	PS	kladné celé číslo
Míra spokojenosti	MS	interval <0; 1>
Míra užití	MU	interval <0; 1>

Tabulka 5 – Zvolená kritéria výběru frameworků

Saatyho metoda

V rámci diplomové práce bylo porovnávání preferencí kritérií provedeno autorem, který během provádění srovnávání preferenci konzultoval svá rozhodnutí s odborníkem v praxi a tuto nápomocnou zpětnou vazbu bral následně v úvahu. V tabulce (viz Tabulka 6) jsou zobrazeny hodnoty srovnávání kritérií Saatyho matice.

	RV	PH	PS	MS	MU
RV	**1**	3	1/4	1/3	1/3
PH	1/3	**1**	1/4	1/3	1/3
PS	4	4	**1**	1	1
MS	3	3	1	**1**	1
MU	3	3	1	1	**1**

Tabulka 6 – Srovnávací tabulka kritérií výběru frameworků

Před tím, než se vypočítají geometrické průměry a váhy kritérií, bylo potřeba nejdříve ověřit konzistenci (resp. nekonzistenci) vytvořené Saatyho matice tak, že je vyjádřen index konzistence. K tomuto indexu bylo potřeba zjistit i maximální vlastní číslo matice. Dané číslo bylo vypočítáno pomocí online dostupného nástroje Wolfram, kde hodnota po zaokrouhlení činila 5,1641. Index konzistence byl tedy vypočítán následovně:

$$CI = \frac{5{,}1641 - 5}{5 - 1} \approx 0{,}0410$$

K vyjádření poměru konzistence byl potřeba jak index konzistence, tak i náhodný index, který je na internetu volně dostupný v různých odborných pracích. Potřebná hodnota

náhodného indexu byla v případě pěti kritérií $1{,}11$. Pomocí těchto hodnot bylo možné vyjádřit poměr konzistence jednoduchým způsobem:

$$CR = \frac{0{,}0410}{1{,}11} \approx 0{,}0369$$

Poměr konzistence vyšel přibližně 4%. Vytvořená Saatyho matice byla proto hodnocena jako dostatečně konzistentní a bylo možné ji využít k dalšímu kroku vyjádření vah.

Využitím hodnot z tabulky (viz Tabulka 6) byl pro každý řádek vypočítán patřičný geometrický průměr. Z geometrických průměrů byla vyjádřena jejich suma a následně i již konkrétní váhy, jak je zobrazeno v tabulce (viz Tabulka 7).

	Geometrický průměr	Váha kritéria
RV	0,6084	0,1041
PH	0,3920	0,0671
PS	1,7411	0,2979
MS	1,5518	0,2655
MU	1,5518	0,2655
∑	**5,8452**	**1,0000**

Tabulka 7 – Vypočtené váhy kritérií výběru frameworků

Podle vypočítaných vah je zřejmě, že nejvíce vlivné byly váhy kritérií počtu stažení za poslední týden, míry spokojenosti a míry užití. Je tomu tak především proto, že se jedná o kritéria, která jsou velice ovlivněna komunitou.

Počet stažení za týden vyjadřuje zájem vývojářů pracovat v daném frameworku. Navíc bylo jednoduchou analýzou časové řady této charakteristiky (viz Obrázek 6) zjištěno, že je zájem používat dané frameworky každým rokem vyšší. Míry spokojenosti a užití poukazují na výsledky průzkumu *State of JS 2020*, který byl zaměřen na vývojáře po celém světě.

V neposlední řadě byl vlivný rok vydání, který do jisté míry udává, jak je daný framework vyspělý. Nejméně vlivným kritériem byl počet hvězd na GitHubu, který do jisté míry udává oblíbenost frameworku, ale u několik let starých frameworků nemá tak velkou váhu jako byl například počet týdenního stahování.

Metoda váženého součtu

Na základě průzkumu *State of JS 2020* bylo vybráno devět frameworků: *React*, *Angular*, *Vue*, *Ember*, *Svelte*, *Preact*, *Stimulus*, *Alpine*, *LitElement*, u kterých byl následně vyjádřen celkový užitek metodou váženého součtu. Dílčí užitky byly určeny u předem vybraných kritérií, a to: *rok vydání, počet hvězd, počet stažení, míra spokojenosti* a *míra užití.*

Před samotným zpracováním bylo potřeba zajistit, aby všechna kritéria byla minimalizační. V případě hodnot vybraných kritérií bylo takto potřeba ošetřit pouze rok vydání. V následující tabulce (viz Tabulka 8) je tato úprava již zaznamenána.

	RV	PH	PS	MS	MU
React	6	164666	10227002	0,80	0,88
Angular	10	71126	2567681	0,56	0,42
Vue	6	180254	2250073	0,49	0,85
Ember	8	21758	187440	0,11	0,27
Svelte	3	44988	131977	0,15	0,89
Preact	4	28459	625260	0,13	0,78
Stimulus	3	10218	133133	0,01	0,67
Alpine	0	14726	36833	0,03	0,82
LitElement	2	4214	2205	0,05	0,78

Tabulka 8 – Hodnoty kritérií jednotlivých variant výběru frameworku

Dále bylo potřeba určit vektory ideální a bazální varianty. U bazální se jednalo o nejnižší hodnoty kritérií mezi všemi variantami. Vektor ideální varianty je složen z hodnot, které jsou naopak nejvyšší. Oba vektory jsou zobrazené v tabulce (viz Tabulka 9).

	RV	PH	PS	MS	MU
D	0	4214	2205	0,01	0,27
H	10	180254	10227002	0,80	0,89

Tabulka 9 – Bazální a ideální varianty výběru frameworku

V další řadě bylo potřeba sestavit kriteriální matici R tak, že hodnoty původní matice (viz Tabulka 8) byly normalizovány. Například míra spokojenosti $j = 4$ u *Angularu* $i = 2$ byla s odpovídajícími hodnotami z bazálního a ideálního vektoru normalizována následovně:

$$u_{2,4} = \frac{0{,}56 - 0{,}01}{0{,}80 - 0{,}01} \approx 0{,}6962$$

Stejným způsobem byly vypočteny zbylé hodnoty a výsledkem je nová normalizovaná kriteriální matice R, která je vyobrazená pomocí tabulky (viz Tabulka 10). Hodnoty této tabulky vyjadřují dílčí užitky kritérií jednotlivých variant.

	RV	PH	PS	MS	MU
React	0,6000	0,9115	1,0000	1,0000	0,9839
Angular	1,0000	0,3801	0,2509	0,6962	0,2419
Vue	0,6000	1,0000	0,2198	0,6076	0,9355
Ember	0,8000	0,0997	0,0181	0,1266	0,0000
Svelte	0,3000	0,2316	0,0127	0,1772	1,0000
Preact	0,4000	0,1377	0,0609	0,1519	0,8226
Stimulus	0,3000	0,0341	0,0128	0,0000	0,6452
Alpine	0,0000	0,0597	0,0034	0,0253	0,8871
LitElement	0,2000	0,0000	0,0000	0,0506	0,8226

Tabulka 10 – Normalizovaná matice R výběru frameworků

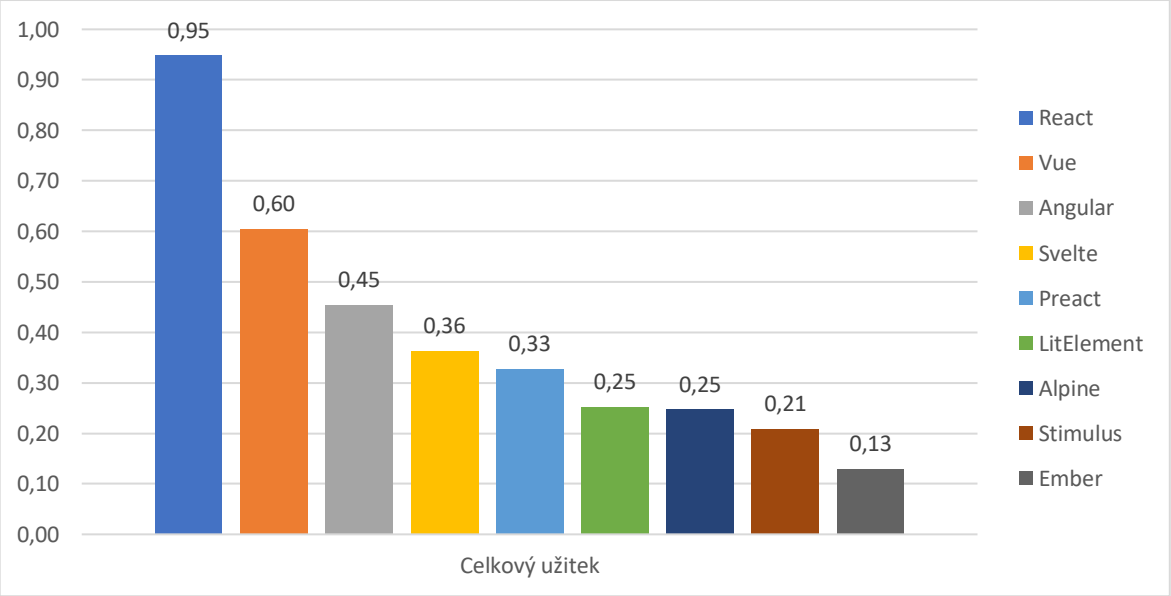
Posledním krokem metody váženého součtu bylo vyjádření celkových užitků variant s použitím vah, které byly stanoveny Saatyho metodou. K tomu bylo možné použít skalární součin (resp. vážené sumy) dílčích užitků. Celkový užitek se například u varianty *Angular* $i = 2$ vypočítal s patřičnými váhami (viz Tabulka 7) takto:

$$u(V_2) = 0{,}10 * 1 + 0{,}07 * 0{,}38 + 0{,}30 * 0{,}25 + 0{,}27 * 0{,}70 + 0{,}27 * 0{,}24 \approx 0{,}45$$

V tabulce (viz Tabulka 11) jsou zobrazeny výsledky celkového užitku variant výběru.

	Celkový užitek	Pořadí
React	0,9481	1
Vue	0,6047	2
Angular	0,4534	3
Svelte	0,3631	4
Preact	0,3277	5
LitElement	0,2526	6
Alpine	0,2473	7
Stimulus	0,2086	8
Ember	0,1290	9

Tabulka 11 – Celkové užitky variant výběru frameworků



Graf 1 – Celkové užitky variant výběru frameworků

Výsledky variant jsou zobrazeny v tabulce výše (viz Tabulka 11), kde byly varianty rovnou seřazeny právě podle hodnoty celkového užitku. Celkový užitek v tomto konkrétním modelu vícekriteriální analýzy variant vyjadřuje, jak moc je framework používaný a oblíbený.

Nejlépe si vedl *React* s celkovým užitkem 0,95, kde na druhém konci skončil Ember s užitkem 0,13. Hodnoty byly velice ovlivněné počtem týdenního stahování, mírou spokojenosti a mírou užití, které vychází z názorů komunity.

4.1.4 Vybrané frameworky

Z výsledků modelu vícekriteriální analýzy variant byly podle celkového užitku vybrány tři frameworky, a to *React*, *Vue* a *Angular*. Tyto tři frameworky si v rámci zvoleného modelu vedly nejlépe, a proto budou podrobeny podrobnějšímu hodnocení.

	Verze	Web	GitHub	Licence
React	17.0.1	reactjs.org	facebook/react	MIT
Vue	3.0.5	v3.vuejs.org	vuejs/vue-next	MIT
Angular	11.2.1	angular.io	angular/angular	MIT

Tabulka 12 – Přehled vybraných JS frameworků

React

React byl vyvinut společností Facebook v roce 2013. Je Facebookem ve velké míře využíván právě v aplikacích Facebook, Instagram a WhatsApp. Ostatní společnosti, které využívají *React* jsou například Uber, Netflix, Twitter, Reddit, Paypal, Tumblr. Poslední verzí je 17.0.1.

Vue

Vue byl na druhé straně vyvinut bývalým zaměstnancem společnosti Google Evanem You, a to v roce 2014. Za poslední roky velice rychle narostl na popularitě. Oproti *Reactu* a *Angularu* má tu nevýhodu, že není přímo podpořen právě velkou společnosti. Na druhou stranu je zčásti podporován Patreonem. Dalšími společnostmi, které *Vue* využívají ve svých projektech, jsou například Xiaomi, Alibaba, WizzAir, EuroNews. Nejaktuálnější je verze 3.0.5.

Angular

Angular, vyvinutý společností Google, byl vydán přibližně v roce 2009. V roce 2016 došlo k velké změně, kdy byl framework rozdělen na dvě rozdílné verze – starý *AngularJS* a aktuální *Angular*. *Angular* například využívá Google a Microsoft. Aktuální verze *Angularu* je 11.2.1.

4.2 Výběr kritérií ke zhodnocení

V minulé kapitole byl proveden výběr frameworků, který byl založen na zvolených kritériích. Byly vybrány pouze tři frameworky, které byly v následující části práce zhodnoceny podrobněji. V první řadě bylo však potřeba vybrat vhodná kritéria, kterými bylo následně možné zhodnotit vybrané frameworky.

Při rešerši a analýze odborných zdrojů, názorů a vlastních zkušeností, byly v první řadě vybrány čtyři aspekty, podle kterých byla kritéria zvolena. Aspekt byl brán jako určitý pohled na daný framework. V mnoha pracích na obdobné téma byly frameworky hodnoceny čistě na základě implementace testovacích aplikací. Tato diplomová práce je zaměřená na více obecné hodnocení. Nebyla tudíž vytvářena aplikace, podle které by se hodnotilo. Místo vlastní aplikace byly brány v potaz aplikace, které mají větší váhu, jelikož jsou spravovány komunitou a v některých případech i vlastníky daných frameworků.

Kritéria hodnocení byla vybírána podle aspektů:

- výkonu,
- komunity,
- použitelnosti,
- vývoje.

4.2.1 Aspekt výkonu

Výkon byl vybrán jako první aspekt výběru kritérií. Webová aplikace jako taková je v první řadě potřeba načíst, aby mohla být uživatelem používána. Při používání aplikace dochází k různým požadavkům, které různě zpracovávají data nebo je uživateli zobrazují. Takovéto potřeby můžou být v některých případech velice náročné, a to především v případech potřeby pracovat s velkým objemem dat. Dokonce i velikost výsledné aplikace může velice ovlivnit výkon například při prvním načtení stránky. Například u jednostránkových aplikací je nutné všechno nejdříve stáhnout ke klientovi. Jednostránkové aplikace jsou založeny na značné manipulaci objektového modelu dokumentu (DOM), které můžou být náročné na zdroje používaného zařízení.

Aspekt výkonu může být však pojat různě. V rámci práce byl výkon pojat jako pohled na framework, kde jsou hodnoceny aplikace napsané ve vybraných frameworcích. Je stěžejní, aby se jednalo o funkčně stejné aplikace, aby mělo srovnání vůbec nějaký význam. Aplikace jsou velice různé. Záleží, kdo ji programoval, do jaké míry ji dokázal optimalizovat a co v ní všechno je za funkcionality. Proto bylo během zkoumání a přípravy diplomové práce zjišťováno, jaké nástroje vývojáři používají ke zhodnocení výkonu aplikací v příslušném frameworku.

Aspekt výkonu byl zpracován pomocí vybraného nástroje *js-framework-benchmark*. Jedná se o projekt, kde jsou komunitou implementovány aplikace zvolených frameworků. Nástroj slouží navíc k měření zvolených implementací a jejich vyhodnocení. Ve výsledcích lze nalézt tři sekce měření zabývající se především třemi metrikami. Jedná se o manipulaci DOM, spouštěcí metriky a alokace paměti. Díky zapojení velké části komunity je projekt brán jako velice vhodný ke zjišťování výkonnostního aspektu vybraných frameworků. V příloze je ukázka aplikace daného projektu (viz Příloha 11).

Na základě těchto poznatků byla vybrána tři kritéria, a to:

- manipulace DOM,
- spouštěcí metriky,
- alokace paměti.

4.2.2 Aspekt komunity

Dalším aspektem je komunita. Jedná se o aspekt, který není tak často zmiňován v ostatních odborných pracích. Ale oprávněně, jelikož se většina zabývá především implementací nějaké aplikace. Nicméně byl tento aspekt vybrán kvůli tomu, že spousta odborníků z praxe považuje komunitu za velice důležitou. Například když chce nějaký vývojář vytvořit v daném frameworku určitou aplikaci, tak může narážet na různé problémy, které nemusí být zdůrazněné v dokumentaci. Díky komunitě, která je rozprostřena na různých webových portálech, je schopen dané problémy vyřešit. Jedná se především o portály Stack Overflow, který je zaměřen právě na vývojáře a jejich otázky. Když je komunita rozsáhlá, tak odpovědí na zmíněném portálu lze najít spousty. Ohledně této problematiky bylo navíc průzkumem Stack Overlfow zjištěno, že 28,6% respondentů navštěvuje portál každý den a 30,5% téměř každý den. (viz Příloha 16). Díky komunitě je navíc na internetu spousta jiných výukových

materiálů než jen samotná dokumentace. Může jít například o videa na YouTube, placené kurzy na Udemy nebo o různé články na internetu. Proto byla komunita vybrána jako vhodný aspekt hodnocení vybraných kritérií, potažmo frameworků.

Výběr kritérií byl poměrně přímočarý. U komunity je důležitá její velikost (resp. rozsah) a aktivita. U velikosti jde o to, kolik přibližně do komunity patří vývojářů a na jakých portálech se nachází. Nicméně pouze velikost komunity nemusí být úplně dostačující. Pojem velké komunity může být brán různě. Někteří můžou považovat sto lidí za velkou komunitu, jiní by se zase pohybovali ve statisících. Proto byla zohledněna i jejich aktivita, kde to platí i opačně. Samotná aktivita nemusí tolik znamenat, pokud se například jedná jen o deset lidí v komunitě. Na vybraných portálech, jako je Stack Overflow a Reddit, byla aktivita a velikost komunity měřena.

Byla vybrána dvě kritéria, a to:

- velikost komunity,
- aktivita komunity.

4.2.3 Aspekt použitelnosti

Třetím aspektem byla vybrána použitelnost, která představuje velice široký pojem. Použitelnost byla pojata z pohledu vývojáře. Například jak je složité se daný framework naučit a jaké má dostupné výukové materiály. Jedná se také o důležitý aspekt. Než začne vývojář daný framework, nebo jiný nástroj, používat, tak je často odkázán na to se ho nejdříve naučit. Když je nástroj velice složitý k naučení, tak je stěžejní mít k dispozici vhodné materiály, jako je dobře zpracovaná dokumentace. Pokud je dokumentace špatně strukturovaná nebo v ní chybí různé informace, je o to více obtížné se s daným nástrojem naučit pracovat. S touto problematikou může souviset i příslušná komunita. Průzkumem Stack Overflow bylo zjištěno, co vývojáři dělají v případech řešení složitých problémů a nemohou se pohnout v práci dál. Většina vývojářů, která činila 90,6%, odpověděla, že hledají odpověď na portálu Stack Overflow (viz Příloha 17). Navíc se open-source projektů často využívá pomoci komunity. Ti v rámci kontribuce můžou přidávat vlastní poznatky a zpětnou vazbu i v rámci samotné dokumentace. Z těchto důvodů byla použitelnost vybrána jako dalším aspektem výběru kritérií.

Když je vývojář odkázán na to se sám naučit nějaký framework, tak potřebuje určité výukové materiály. Nejběžnějším materiálem je právě oficiální dokumentace daného frameworku. Dále je vhodné vědět, jak moc je složité se vybraný framework naučit a jak moc je složitý při používání.

Proto byla vybrána kritéria:

- dokumentace,
- křivka učení.

4.2.4 Aspekt vývoje

Nakonec byla vybrán vývoj, který lze pojmout a hodnotit velice různě. V rámci diplomové práce byl tento pojem pojat z pohledu vývojáře, a to jakou má svobodu a možnosti při používání daného frameworku. Například jak moc je možné s frameworkem volně pracovat nebo jestli lze framework využít v již existujících projektech.

U vývoje byl výběr kritérií nejvíce obtížný. Bylo potřeba zjistit, v jakých ohledech lze vývoj vůbec hodnotit. V různých článcích dostupných na webu a v odborných pracích bylo velice často odkazováno na ekosystém a flexibilitu. Dostupný ekosystém a různé knihovny komunity mohou významně zjednodušit vývoj a nabídnout vývojáři mnohem více existujících řešení, které mohou využít i ve vlastních projektech, což by mohlo vést i k nižším nákladům. V rámci vlastního ekosystému je možné vyvíjet rychleji s předem definovaným standardem. Vývojáři nebudou nuceni si vlastní ekosystém spojovat z různých knihoven, pokud framework nabízí vlastní řešení potřebného ekosystému. S velkou a aktivní komunitou je možné narazit na spoustu těchto knihoven nebo nástrojů. Flexibilita vypovídá například o tom, do jaké míry lze framework volně používat. Jestli je například možné framework zavést do existujícího projektu. Může jít i o to, jestli je vývojář nucen se pevně řídit pravidly frameworku, což může být například struktura projektu.

Dané poznatky byly využity pro výběr kritérií aspektu vývoje a byly tedy zvoleny:

- dostupné knihovny,
- ekosystém,
- flexibilita.

4.2.5 Vybraná kritéria hodnocení

Na základě odborných zdrojů, článků dostupných na internetu a názorů komunity bylo vybráno celkem devět kritérií. Každé kritérium vypovídá o jednom ze čtyř aspektů pohledu na daný framework. Konečný výčet kritérií ne znázorněn v následující tabulce.

Aspekt	Kritérium
Výkon	Manipulace DOM
	Spouštěcí metriky
	Alokace paměti
Komunita	Velikost komunity
	Aktivita komunity
Použitelnost	Dokumentace
	Křivka učení
Vývoj	Ekosystém balíčků
	Vlastní ekosystém
	Flexibilita

Tabulka 13 – Vybraná kritéria hodnocení frameworků

4.3 Hodnocení frameworků

V následující části práce je podrobněji popsáno, jak byla zvolená kritéria hodnocena. Hodnocení bylo prováděno na základě porovnatelných poznatků jednotlivých kritérií. Je potřeba podotknout, že v rámci vlastní práce byla veškerá časově vlivná data měřena v rámci stejného dne. Kdykoliv došlo ke změně nebo přidání poznatku, tak byla příslušná data opět přeměřena.

Navíc se zde používá pojem „poznatek“ místo slova „kritérium“ nebo „podkritérium“. U každého kritéria hodnocení byly na základě analýzy a dalšího zjišťování definovány další kritéria. Mezi těmito kritérii bylo možné srovnávat a hodnotit varianty v rámci každého kritéria. Aby byl rozdíl jasnější, bylo proto zvoleno slovo „poznatek“.

U hodnocení bylo stěžejní zohlednit charakter porovnatelných informací. V určitých případech šlo o poznatky kvantitativního charakteru, které lze přirozeně hodnotit

jednodušeji. S poznatky kvalitativního charakteru to bylo na druhé straně velice obtížné, jelikož bylo potřeba daný rozdíl slovně zhodnotit a podat pádné důvody.

4.3.1 Stupnice bodování jednotlivých kritérií

Nejdříve bylo potřeba definovat stupnice, kterými se mělo hodnotit. Během vlastní práce byly prozkoušeny různé stupnice, které byly zaměřeny na poznatky jak kvalitativního, tak kvantitativního charakteru.

Byly sestaveny dvě bodovací stupnice, kde hodnoty vypovídaly o tom, jak moc se poznatky mezi sebou lišily, a to stupnice:

- kvantitativní,
- kvalitativní.

U každého kritéria se celkový počet bodů lišil právě na základě počtu zvážených poznatků.

Kvantitativní stupnice

Kvantitativní stupnice se zakládala na procentuálním rozdílu od nejlepší hodnoty poznatku. Varianta u daného poznatku získala 5 bodů, pokud se jednalo o nejlepší hodnotu mezi ostatními. Zbylé byly bodovány dle stupnice v následující tabulce (viz Tabulka 14).

Rozdíl od nejvíce vyhovující varianty v %	Bodování
$< 0;\ 20\ )$	5
$< 20;\ 40\ )$	4
$< 40;\ 60\ )$	3
$< 60;\ 80\ )$	2
$< 80;\ 100 >$	1

Tabulka 14 – Kvantitativní stupnice hodnocení

V případě, kdy se jednalo o situaci minimalizační povahy poznatku, kdy je nejmenší hodnota nejlepší, byly rozdíly vypočítány jednoduchým vzorcem:

$$x'_{ij} = 1 - \frac{min_i\ (x_{ij})}{x_{ij}}$$

kde x_{ij} je hodnota poznatku konkrétní varianty a $min_i(x_{ij})$ je minimální hodnotou mezi variantami. U maximalizační povahy poznatku se rozdíl od nejlepšího výsledku vypočítal následovně:

$$x'_{ij} = 1 - \frac{x_{ij}}{max_i\ (x_{ij})}$$

kde x_{ij} je opět hodnota poznatku varianty a $max_i(x_{ij})$ označuje maximální hodnotou poznatku mezi vybranými frameworky.

Kvalitativní stupnice

Poznatky kvalitativního charakteru byly bodovány podobně, ale s tím rozdílem, že bodování bylo založeno na slovním vyjádření. Zde byla zřejmá veliká nevýhoda informací kvalitativního charakteru, jelikož na určitou věc může být nahlíženo různými způsoby. Jedná se někdy o velice subjektivní názory.

U bodování bylo vycházeno z názorů odborníků z praxe, komunity nebo z vlastních zkušeností. I v příslušných komunitách bylo na některé poznatky nahlíženo velice různě, tudíž byly velice odlišené. Na následující tabulce je zobrazena kvalitativní stupnice (viz Tabulka 15).

Rozdíl od nejvíce vyhovující varianty	Bodování
Stejně vyhovující	5
	4
	3
	2
Nejméně vyhovující	1

Tabulka 15 – Kvalitativní stupnice hodnocení

4.3.2 Zhodnocení dle jednotlivých kritérií

Při hodnocení (resp. bodování) kvalitativních poznatků byly využity informace a názory příslušných komunit, které zahrnují příspěvky a články na různých komunikačních sítích. Autor hodnotil i na základě vlastních zkušeností, nicméně se více opíral o názory oslovených odborníků v praxi, kteří byli velice vstřícní při odpovídání na různé dotazy.

Manipulace DOM

Webové aplikace zaměřené na interaktivitu s uživatelem jsou založené na značné manipulaci prvků na stránce. Především se jedná o jednostránkové aplikace, kde na straně klienta téměř vůbec nedochází k opětovnému načítání celé stránky při pouhé potřebě změnit pár informací nebo něco uložit do databáze. Proto je kritérium manipulace podstatné pro vyjádření výkonu aplikace napsané v příslušném frameworku.

K měření tohoto kritéria bylo přistoupeno pomocí již zmíněného nástroje *js-framework-benchmark*, ke kterému komunita přidává vlastní implementace aplikací v příslušných frameworcích. Díky tomuto nástroji lze implementace měřit podle předem určených metrik. V příloze je dostupná ukázka aplikace (viz Příloha 11), ukázka spouštění (viz Příloha 12) a provádění (viz Příloha 13) jejího měření a výsledky (viz Příloha 14) vybraných frameworků.

V rámci manipulace objektového modelu dokumentu se tímto nástrojem měří, jak rychle dokáže aplikace v daném frameworku rychle zpracovávat běžné manipulace s prvky na stránce.

Byly měřeny doby potřebné k:

- vytvoření 1000 řádků tabulky přímo po načtení stránky,
- nahrazení vytvořených řádků tabulky za 1000 nových,
- částečné aktualizaci každého desátého prvku napříč 10 000 řádky tabulky,
- výběru a barevnému zvýraznění po kliknutí na vybraný řádek tabulky,
- prohození dvou řádků mezi 1000 řádků tabulky,
- odstranění jednoho řádku z 1000 řádků tabulky
- vytvoření 10 000 řádků najednou,

- přidání 1000 řádků k již existující tabulce s 10 000 řádků,
- odstranění 10 0000 řádků najednou

Výsledky hodnocení (resp. bodování) lze nalézt v příloze (Příloha 1) na stránce č. 99.

Spouštěcí metriky

Jednostránkové aplikace jsou postavené tak, aby byly načteny pouze jednou, a to v době, kdy uživatel přistoupí na danou stránku. Pokud je aplikace moc velká, je uživatel nucen využít více zdrojů například k jejímu stažení. To není vhodné u mobilních zařízení, kdy je uživatel často na vlastních datech. Proto jsou dalším důležitým kritériem výkonového aspektu metriky spojené se spouštěním aplikace.

Je stěžejní, aby potenciální uživatel stránky neodešel dřív, než se vůbec daná aplikace načte. Pokud není aplikace navíc dostatečně optimalizovaná, tak může pracovat velice pomalým způsobem, což může snadno daného uživatele odradit. V takových případech je například možné i přijít o zisk, jednalo by se například o e-shop.

Pod toto kritérium spadají metriky spojené s výkonnostními metrikami, které jsou definovány Googlem. Proto je možné následující informace změřit manuálně pomocí Lighthouse. Zvolený nástroj *js-framework-benchmark* však umí některé metriky zpracovávat automaticky. V příloze jsou dostupné ukázky měření (viz Příloha 11, Příloha 12, Příloha 13 a Příloha 14).

V rámci těchto metrik byla měřena:

- konzistentní interaktivita (TTI),
- doba spouštění skriptu,
- náklady na síťový přenos.

Konzistentní interaktivita je jednou ze šesti Lightouse metrik, která vyjadřuje, jak dlouho trvá se ujistit, že je stránka připravena k interaktivitě uspokojivým způsobem. To vypovídá o tom, že je web potřeba optimalizovat takovým způsobem, aby se z něj uživatel nezbláznil. Některé weby jsou schopny optimalizovat viditelnost obsahu na úkor její interaktivity. To může mít nepříjemné následky pro uživatele. Obsah sice vidí, ale nemůže s ním nic dělat. Proto se jedná o velice důležitou metriku.

Další metrikou je čas spouštění skriptu jako takového. Zde se jedná opět o metriku definovanou Googlem, a kterou je možné zachytit pomocí nástroje Lighthouse. U této metriky je měřeno, kolik milisekund je potřeba k parsování, kompilaci a vyhodnocení všech skriptů dané stránky.

Náklady na síťový přenos jsou opět metrikou Lighthouse. Zaměřuje se na celkovou velikost potřebných zdrojů, které je potřeba přenést po síti.

Konkrétní bodování je možné najít v příloze (Příloha 2) na stránce č. 99.

Alokace paměti

Posledním kritériem aspektu výkonu byla alokace paměti, která souvisí s již uvedenými kritérii. Kritérium bylo opět měřeno zmíněným nástrojem, kde v příloze jsou dostupné ukázky měření (viz Příloha 11, Příloha 12, Příloha 13 a Příloha 14). U alokace je měřena zátěž zdrojů používaného zařízení, jako je počítač nebo mobilní telefon.

Je zde měřena paměťová zátěž po:

- načtení stránky,
- přidání 1000 řádků,
- kliknutí na aktualizaci desátého řádku pětkrát za sebou,
- kliknutí na vytvoření 1000 řádků pětkrát za sebou,
- vytvoření a smazání 1000 řádků pětkrát za sebou.

Bodování je možné nalézt v příloze (Příloha 3) na stránce č. 99.

Velikost komunity

Během zpracovávání této práce bylo zjištěno, že je pro vývojáře důležitá komunita a její stav. Prvním takovým stavem je právě její velikost. Velká komunita může například ovlivnit jak křivku učení daného frameworku, tak i její celkové použití. Například vývojáři si s oblibou ulehčují práci psaním různým pomocných nástrojů, které ve většině případů vypustí do světa. Případně jsou cílené předem na jiné vývojáře. S menší komunitou takových nástrojů může být méně. Dalším přínosem velké komunity můžou být jiné výukové materiály. Většina vývojářů ani nemusí pročítat celou dokumentaci, jelikož mají u velké

komunity například dostupná videa na YouTube, kurzy na Udemy. Výukovými materiály mohou být i různé články na internetu, které píšou odborníci v praxi.

V rámci práce byly vybrány weby, portály a chatovací místnosti, kde bylo možné frameworky mezi sebou porovnat na základě počtu členů jim příslušné. Nebyly brány v potaz všechny možnosti každého frameworku, jelikož měl některý framework komunitu tam, kde zbylé varianty ne. Byly naopak ale brány v potaz jak oficiální, tak i neoficiální kanály. Šlo o porovnání velikosti komunity především na základě počtu členů ve vybraných kanálech.

Na GitHubu bylo možné zvážit velikost komunity třemi způsoby. Ta mohla být určena počtem hvězd, což by mohlo do jisté míry vyjádřit počet vývojářů, kteří patří do komunity příslušného frameworku. V rámci GitHub bylo dále možné určit, kolik vývojářů daný framework používá. Nakonec bylo zváženo i kolik lidí přispělo k vývoji frameworku především formou kontribuce do kódu.

Twitter byl vybrán jako další komunitní kanál. Jedná se o sociální sít, která je dnes velice využívaná. Komunita se navíc zdála být u všech variant celkem aktivní, a to jak ze strany autorů frameworku, tak i ze strany sledujících například podle počtu *retweetů*.

V neposlední řadě byly zváženy komunity na sociální síti zvané Reddit, který je poslední dobou čím dál více využívaným, a to nejenom u vývoje aplikací. Je založen právě na zakládání vlastních komunit ve formě tzv. *subredditů*, které může fungovat jako obyčejné fórum. Zde bylo možné pomocí dostupných statistik zjistit, kolik má daný subreddit například počet odběratelů. Pro každý framework byl vybrán jeden subreddit, který měl nejvíce odběratelů[3].

Asi nejvíce vlivným místem hromadění komunity příslušných frameworků je právě Stack Overflow, který je založen na vytváření otázek vývojářů, na které ostatní vývojáři mohou odpovídat. Jedná se o velice používaný kanál, kde bylo možné například zjistit, kolik příspěvků náleží danému frameworku. Proto byl zvážen i počet těchto příspěvků. Nicméně byly započteny pouze ty, které měly akceptovanou odpověď.

A nakonec byly zváženy chatovací místnosti Discord a Gitter, které jsou založeny na hromadné komunikaci v reálném čase pomocí chatu. Discord je komunikační nástroj, který je poslední dobou používaný nejenom u herních komunit. Autorovi se zdálo vhodné využít

[3] Subreddity: *r/reactjs, r/vuejs, r/angular2*

Discord při odhadnutí velikosti komunity, a to především kvůli jeho popularitě. Gitter je na druhou stranu zaměřený na vývojářské komunity. Komunita u *Angularu* značně převládala na tomto komunikačním kanálu, a proto byl pro daný framework zvolen. U *Reactu* i *Vue* byly zvoleny nejužívanější Discord servery[4] a u *Angularu* server Gitter[5].

Bylo nakonec hodnoceno:

- počet hvězd na GitHubu,
- počet využívajících na GitHubu,
- počet přispěvatelů na GitHubu,
- počet sledujících na Twitteru,
- počet odběratelů na Redditu,
- počet vyřešených otázek na Stack Overflow,
- počet členů v chatovacích místnostech.

Bodování je možné nalézt v příloze (Příloha 4) na stránce č. 100.

Aktivita komunity

Jako tomu bylo u velikosti komunity, tak i její aktivita je velice stěžejní pro existenci frameworku. Díky velikosti a aktivitě je možné do jisté míry určit například stav, v jakém se framework zrovna nachází. Například jestli je vůbec použitelný. Pokud by byl framework špatně napsaný a byl by nepoužitelný, nebyla by kolem něj velká, respektive aktivní komunita. A jak tomu bylo i u velikosti, tak aktivita značně podporuje křivku učení a celkový vývoj ve formě různých knihoven a nástrojů, kde je opět zřejmá výhoda open-source projektů.

Velice aktivní komunita dokáže značným způsobem pomáhat jiným vývojářům, a to nejenom množstvím výukových materiálů. Mohou pomáhat i například odpovídáním na různě specifické dotazy, které nelze odpovědět pouhým čtením dokumentace. U aktivní komunity je na dotazy velice rychle odpovězeno a vývojáři si do té doby nemusí vytrhat všechny vlasy. Někdy se také stává, že je dokumentace nedostačující, kde je opět možné

[4] Discord (React, Vue): https://discord.com/invite/Reactiflux, https://discord.com/invite/HBherRA

[5] Gitter (Angular): https://gitter.im/angular/angular

využít různých komunitních kanálu a zeptat se ostatních. Navíc je aktivita velice důležitá při detekci různých chyb. Spousta vývojářů potřebuje využít framework v rámci různých případů užití.

V rámci tohoto kritéria byly opět zváženy porovnatelné informace, kterými lze do jisté míry vyhodnotit aktivitu frameworků. Byly vybrány takové komunitní kanály, u kterých bylo možné volně dostupnými statistikami vyjádřit aktivitu.

U *npm* bylo možné srovnat aktivitu na základě počtu stažení frameworků. Tato informace je spjata s mírou využívání daného frameworku a nemusí vypovídat o její aktivitě na jiných sítích, jako je například Stack Overflow.

Stack Overflow je, jak již bylo zmíněno, velice vlivným zdrojem informací každého vývojáře. Zde byla aktivita měřena počtem příspěvků s akceptovanou odpovědí, jako tomu bylo u velikosti komunity. Ale s tím rozdílem, že zde byly vyfiltrovány příspěvky pouze za poslední měsíc. To má velice dobrou vypovídací schopnost ohledně aktivity, především při srovnání aktivity komunit vybraných frameworků.

Nakonec byly využity komunikační kanály sítě Reddit, kde bylo možné pomocí volně dostupných statistik[6] zjistit denní aktivitu uživatelů, i když daný subreddit neodebírají. Byly vybrány stejné subreddity, které byly zváženy u kritéria velikosti komunity. Zde byl měřen počet příspěvků a komentářů za poslední den.

Hodnoceno bylo tedy:

- počet stažení na *npm* za týden,
- počet vyřešených otázek na Stack Overflow za poslední měsíc,
- počet příspěvků na Redditu za jeden den,
- počet komentářů na Redditu za jeden den.

Výsledky hodnocení tohoto kritéria jsou v příloze (Příloha 5) na stránce č. 100.

[6] Dostupné na https://subredditstats.com (Subreddit Stats, 2021)

Dokumentace

Dokumentace, jakožto první hodnocené kritérium použitelnosti frameworku, patří mezi nejdůležitější materiály k jeho naučení a používání. Měla by obsahovat všechny náležitosti, které vývojář musí pochopit, aby byla jeho práce s frameworkem co nejjednodušší. Vlastní práce se zabývala porovnatelnými poznatky, které se nacházejí přímo v dokumentaci. Nebyly zohledněny výukové materiály třetích stran.

V první řadě je stěžejní, aby byla dokumentace úplná, aby obsahovala vše potřebné k použití příslušného frameworku. Měly by být například uvedeny informace o prvním spuštění a jak toho dosáhnout. Je tedy důležitý obsah daného frameworku.

K obsahu dokumentace je také vhodné umožnit případný zásah komunity ve formě kontribuce na GitHubu. Autor frameworku mohl například opomenout určité informace ohledně používání nebo je vysvětlení problematiky špatně pochopitelné. Komunita tímto může pomoci s obsahem dokumentace.

Při používání frameworku je také zásadní API reference kódu. Jedná se o doplněk k dokumentaci, která se ve většině případů automaticky generuje z kódu samotného frameworku. Generují se především části, které jsou cílené na používání uživatelem. Jelikož je dokumentovaný i samotný kód, tak je některými vývojáři API reference považována za nepotřebnou. Výhoda API reference například tkví v jeho jednoduchém prohledáváním. Navíc ne všichni vývojáři preferují koukat do zdrojových kódů. Tudíž je vhodné nabízet vývojářům možnost volby.

Pokud v dokumentaci existují první kroky a další návody, tak se může jednat o zlepšení použitelnosti frameworku. Vývojáři jsou v rámci těchto návodů postupně představovány různé funkcionality a možnosti frameworku. Není hlavně nucen vyhledávat si vlastní návody, jak napsat první aplikaci. Na druhou stranu existují ti, kterým to vyhovuje více. I přesto byla zohledněna i tato skutečnosti.

K návodům je vhodné přidat i tzv. *cheatsheet*. Ten může pomoci jak novým, tak i vracejícím se vývojářům, jelikož nabízí základní kusy kódu, které lze jednoduše zkopírovat a využít.

Na dokumentaci však vývojář nekouká pouze jednou. Je velice běžné, že si dokumentaci prochází vícekrát, nebo se k ní vrací v případě menších nejasností, což se může dít i u zkušenějších vývojářů. Proto je vhodné, aby byla dokumentace součástí webové stránky, která má dobře implementované vyhledávání. Vývojář není poté nucen procházet manuálně všechny stránky, aby našel právě tu informaci, kterou potřebuje.

Další dobrou možností pro vývojáře jsou interaktivní příklady přímo v dokumentaci. Vývojář získává možnost si framework vyzkoušet na jednoduchých příkladech, které jsou dostupné v dokumentaci. Není nucen si framework kvůli tomu stahovat, když si například není jistý, jestli je pro něj vhodný.

Dokumentace není často pročítána úplně celá, jelikož bývá dosti obsáhlá, a ne každý může mít takovou časovou dispozici. Proto je stěžejní umožnit vývojáři vyhledávat v rámci dokumentace. Vyhledávání by mělo být více komplexní v tom smyslu, že není uživatel nucen zadávat přímo názvy potřebných sekcí. Navíc je vhodné, když jsou výsledky strukturovány i podle jejich kontextu.

Nakonec bylo bráno v potaz i to, v kolika jazycích je dokumentace napsaná. Angličtina je samozřejmě dostačujícím jazykem, avšak úroveň tohoto jazyka není u všech lidí stejná. Některé náležitosti nebo funkce člověk může lépe pochopit ve svém rodném jazyce.

Na základě těchto poznatků bylo hodnoceno:

- obsah dokumentace,
- možnost kontribuce,
- API reference,
- první kroky a další návody,
- cheatsheet,
- interaktivní příklady,
- vyhledávání
- a dostupné jazyky.

Hodnocení dokumentace je možné nalézt v příloze (Příloha 6) na stránce č. 100.

Křivka učení

Křivka učení je založena na vlivu mnoha skutečností a aspektů. Je především ovlivněna složitostí a zaměření daného frameworku. Dále může být ovlivněna i komunitou, která pro framework vytváří různé výukové materiály nebo nabízí pomoc v rámci různých komunitních sítí. Vlivné může být i to, jestli je vývojář odkázán na používání knihoven třetích stran, aby vytvořil složitější aplikaci než *„Hello world!“*. Vlastní práce se zaměřuje

především na použití frameworku jako takového. Tudíž nebyl například zvážen dostupný počet výukových materiálů. Opět se jednalo o poznatky, které bylo možné mezi variantami srovnat.

Je nicméně nutno podotknout, že křivka učení byla hodnocena na základě předpokladu z pohledu začínajícího vývojáře. Jedná se o předpoklad, kde člověk umí pouze jazyky JS, HTML a CSS, a navíc nikdy nepracoval s frameworky. Vývojář by měl před samotným učením frameworku znát alespoň uvedené základy.

V první řadě bylo hodnoceno, jak se ve frameworku přistupuje k jazyku TypeScript. Na základě zadaného předpokladu je zřejmé, že pokud framework vyžaduje práci v této JavaScriptové nadstavbě, tak je vývojář nucen se nejdříve seznámit s tímto jazykem. To může ovlivnit křivku učení především u lidí, kteří právě neznají jiné jazyky. Jinak tomu může být u vývojářů, kteří mají určité zkušenosti například s jazykem C#. Nicméně je tato skutečnost opomíjena kvůli definovanému předpokladu.

Vybrané frameworky jsou zaměřeny na vytváření komponent s jejichž pomocí je možné různě poskládat webovou aplikaci. V rámci tohoto poznatku bylo hodnoceno, jak je jednoduché psaní těchto komponent a jaké má možnosti vykreslování. Byl obecně zohledněn i způsob jejich užití v rámci aplikace.

V mnoha frameworcích je vývojář seznamován s konceptem *dependency injection*. Tato technika vstřikování závislostí je značně používána v objektově orientovaných jazycích. Jedná se o velmi užitečnou techniku, avšak může být pro nové vývojáře velice obtížná na pochopení. Na základě tohoto poznatku bylo hodnoceno, zdali je framework závislý na použití této techniky, která může potenciálně zvýšit křivku učení.

Dále je ve frameworcích velice běžné pracovat s vazbou dat (data binding), což je další obecnou technikou programování. Jedná se o techniku spojení zdrojů dat mezi poskytovatelem a spotřebitelem. Podstatou tohoto spojení je právě jednoduchá synchronizace mezi danými stranami. V praxi je vazba dat využívána jak jednostranně, tak i oboustranně. Jednostranný způsob spočívá v provázání dat pouze jedním směrem, což zaručuje stabilitu aplikace, ale vyžaduje neměnnost. U druhého způsobu se data mohou měnit na obou stranách, což může přinést jak výhody, tak i nevýhody. Druhý způsob může být však pro vývojáře lépe pochopitelný, jelikož není člověk nucen využívat neměnnost. V rámci práce se však hodnotilo, zdali má vývojář možnost si vybrat z obou způsobů a není odkázán na využívání pouze jednoho způsobu.

Další důležitou vlastností frameworku je práce s asynchronním kódem, který v rámci JS umožňuje vykonávat více věcí najednou bez vzájemného blokování. Je to velice užitečné například při stahování dat ze serveru, které může trvat určitý čas. Běh kódu není tudíž zpomalen (resp. zastaven) čekáním na daná data. Hodnocení bylo založeno na tom, jak jednoduše lze asynchronní kód v příslušném frameworku používat.

V poslední řadě byla zvážena i skutečnost, že jsou některé frameworky postavené na velice omezeném počtu funkcí, a tudíž nejsou komplexním řešením pro vývoj webových aplikací. K daným frameworkům je proto nutné buď manuálně dané funkcionality dopsat nebo využít knihovny třetích stran, které je potřeba se dodatečně naučit využívat. Proto byla zvážena i míra potřeby používání knihoven třetích stran.

Na základě těchto poznatků byla hodnocena:

- práce s TypeScript,
- psaní komponent,
- vykreslování nebo psaní šablon,
- způsob použití komponent,
- znalost techniky dependency injection,
- práce s vazbou dat (data binding),
- práce s asynchronním kódem,
- míra potřeby práce s knihovnami třetích stran.

Konkrétní bodování křivky učení lze nalézt v příloze (Příloha 7) na stránce č. 101.

Dostupné knihovny

Nejenom vývojáři webových aplikací si rádi ulehčují práci, a to nejenom kvůli tomu, že by byli líní. Když se chce člověk například dostat z jednoho místa do druhého, nebude si vymýšlet nový způsob přepravy a využije již existujících řešení, jako je na příklad auto nebo MHD. Totéž platí i při vývoji. V mnoha případech je vhodné využívat již existující řešení, která dokážou ulehčit vývoj, což vede i k levnějšímu vývoji.

V rámci toho kritéria bylo hodnoceno pouze to, kolik dostupných knihoven třetích stran existuje pro daný framework v rámci správce balíčků *npm*.

Na základě těchto poznatků byl hodnocen:

- počet balíčků na *npm*.

Hodnocení kritéria lze nalézt v příloze (Příloha 8) na stránce č. 101.

Ekosystém

Pod ekosystémem se u frameworků rozumí knihovny a jiné nástroje, které dohromady tvoří konkrétní aplikaci. V praxi je běžné, že se využívají různé ekosystémy podle potřeb aplikace, a to především u frameworků, které mají omezený počet funkcí. Některé frameworky mají takový ekosystém, že obsahují od začátku všechno možné, co by mohlo být vývojáři užitečné při vytváření konkrétního řešení.

Má-li framework svůj vlastní komplexní ekosystém, může tím potenciálně zvyšovat stabilitu budoucí aplikace. To může však vést ke zbytečné velikosti, jelikož výsledné aplikace obsahuje nevyužité části frameworku. V takových případech je potřeba využívat různých optimalizačních technik, jako je *tree-shaking*. Při využívání knihoven třetích stran má vývojář volnost vytvořit si vlastní ekosystém. Nicméně je potřeba dbát na to, jestli jsou dané knihovny podporovány komunitou a jestli jsou udržované. V případě potřeby aktualizace aplikace na nejvyšší verzi frameworku, může využívání knihoven třetích stran být značným úskalím, pokud již nejsou dále udržované.

Mezi velice běžné potřeby JavaScriptových frameworků je práce s DOM. Touto manipulací je vývojář například schopen zajistit interaktivitu aplikace. Aby mohl navíc zpracovávat data a udržovat aplikaci v určitém stavu, potřebuje k tomu nástroje na správu stavu. Další často využívanou funkcionalitou je routování, které spočívá v překládání URL adres na konkrétní akce. V mnoha případech jsou uživateli přístupné i různé formuláře, jako je například přihlašovací formulář. Je tedy zásadní, aby měl vývojář k dispozici nástroje k validaci a zpracování formulářů. Poslední potřebou byly vybrány testovací nástroje, které jsou důležité jak u menších, tak i větších aplikací. Bylo tedy hodnoceno, jestli mají frameworky ve vlastním ekosystému nástroje na běžné potřeby aplikací.

Byla nakonec hodnocena:

- manipulace DOM,
- správa stavu,
- routování,

- validace a zpracování formulářů,
- testovací nástroje.

Výsledky hodnocení jsou zobrazeny v příloze (Příloha 9) na stránce č. 101.

Flexibilita

Na flexibilitu lze nahlížet různými způsoby. Flexibilita vývoje byla ve vlastní práci pojata jako volnost vývoje v daném frameworku. Například jestli je framework zaměřen pouze na jeden způsob řešení určitého problému, nebo jestli nabízí i jiné alternativy.

Je třeba podotknout, že flexibilitu lze brát jak kladně, tak i negativně. Někteří vývojáři preferují volnost a různé možnosti řešení problémů. Na druhé straně jsou tací, kterým vyhovuje spíše pevně daná pravidla, jak by co mělo být řešeno. Vlastní práce nahlížela na flexibilitu tím kladným způsobem.

Pokud se jedná o aplikace, které má využívat především uživatel, tak je dnes velice běžné se zaměřovat na interaktivitu a uživatelský zážitek uživatele s danou aplikací. Modernizace webů je tudíž velice častá, ale jelikož nemusí být všechny psané v jazyce JavaScript, je interaktivita častým problémem. Proto byla flexibilita pojata i ve smyslu možnosti zavedení frameworku do existujícího projektu s cílem využít pouze část frameworku (např. na formuláře) nebo postupně takto danou aplikaci postupně přepisovat.

Dále bylo zohledněno, jestli má framework striktně daná pravidla například ohledně struktury a používání určitých nástrojů. Frameworky jsou na architektuře založené, tudíž je tato vlastnost do jisté míry potřeba, aby vůbec fungovaly. Nicméně ne všechny frameworky jsou takto stejně umíněné.

V neposlední řadě byly zohledněny možnosti používání frameworku v rámci vykreslování na straně serveru (Server Side Rendering), což je technika vhodná pro aplikace vyžadující SEO optimalizaci, na což jsou čistě jednostránkové aplikace nevhodné.

Jelikož jsou mobilní zařízení pomalu používanější než samotné počítače, tak byla zvážena i možnost vývoje mobilních aplikací v rámci multiplatformního vývoje s příslušnými frameworky.

Nakonec byly zváženy i možné způsoby psaní komponent a jejich možnosti stylování.

Na základě těchto poznatků byla hodnocena:

- možnost zavedení do existujícího projektu,
- striktnost pravidel vývoje,
- možnosti vazeb dat (data binding),
- možnosti vykreslování na straně serveru,
- možnost multiplatformního vývoje,
- možnosti psaní komponent,
- možnosti vykreslování a šablonování,
- možnosti stylování.

Výsledky hodnocení lze najít v příloze (Příloha 10) na stránce č. 102.

4.3.3 Model vícekriteriální analýzy variant

K vyjádření celkového hodnocení frameworků byla opět využita vícekriteriální analýza variant.

Model je postaven ze Saatyho metody a metody váženého součtu, kde první metodou byly stanoveny váhy a druhou byly vyjádřeny celkové užitky na základě zvolených kritérií.

V první řadě bylo potřeba stanovit váhy kritérií. S pomocí Saatyho metody byla vytvořena Saatyho matice, která vyjadřuje párové porovnávání mezi zvolenými kritérii. Hodnoty všech kritérií byly bodového charakteru.

Celkový počet bodů byl však u každého kritéria jiný, a to na základě zvážených poznatků. V tabulce (viz Tabulka 16) jsou znázorněna kritéria.

Kritérium		Maximální počet bodů
Manipulace DOM	V1	45
Spouštěcí metriky	V2	15
Alokace paměti	V3	25
Velikost komunity	K1	35
Aktivita komunity	K2	20
Dokumentace	P1	40
Křivka učení	P2	40
Ekosystém balíčků	M1	5
Vlastní ekosystém	M2	25
Flexibilita	M3	40

Tabulka 16 – Kritéria hodnocení frameworků

Saatyho metoda

Srovnávání preferencí v rámci Saatyho matice bylo osloveno několik odborníků v praxi. Na základě těchto získaných srovnání bylo vybráno takové řešení, u kterého vycházel nejnižší poměr konzistence. V následující tabulce (viz Tabulka 17) je zobrazena Saatyho matice, která byla nakonec využita ve vlastní práci.

	V1	V2	V3	K1	K2	P1	P2	M1	M2	M3
V1	**1**	1/3	1/3	1/3	1/5	1/5	1/5	1/5	1/5	1/5
V2	3	**1**	3	1/5	1/5	1/5	1/5	1/5	1/5	1/5
V3	3	1/3	**1**	1/5	1/5	1/5	1/5	1/5	1/5	1/5
K1	3	5	5	**1**	1/2	1/2	1/2	1	1	3
K2	5	5	5	2	**1**	1/2	1/2	1	1	3
P1	5	5	5	2	2	**1**	1/2	1	1	3
P2	5	5	5	2	2	2	**1**	1	1	3
M1	5	5	5	1	1	1	1	**1**	1	3
M2	5	5	5	1	1	1	1	1	**1**	1
M3	5	5	5	1/3	1/3	1/3	1/3	1/3	1	**1**

Tabulka 17 – Srovnávací tabulka kritérií hodnocení frameworků

Před výpočtem vah kritérií bylo potřeba nejdříve ověřit konzistenci (resp. nekonzistenci) Saatyho matice. Na začátku bylo potřeba vyjádřit index konzistence. K tomuto indexu je navíc nutné zjistit maximální vlastní číslo matice, které bylo vypočteno pomocí online dostupného nástroje Wolfram. Tato hodnota po zaokrouhlení činila $10{,}7699$. Index konzistence mohl být vypočten následovně:

$$CI = \frac{10{,}7699 - 10}{10 - 1} \approx 0{,}0855$$

Index konzistence byl následně využit k vyjádření poměru konzistence. Nicméně bylo potřeba určit i náhodný index, který je předem definovaný podle počtu kritérií. V případě deseti kritérií hodnota RI činila $1{,}49$. Z těchto hodnot bylo možné vyjádřit poměr konzistence:

$$CR = \frac{0{,}0855}{1{,}49} \approx 0{,}0574$$

Poměr konzistence vyšel přibližně 6%. Saatyho matice byla proto považována za dostatečně konzistentní a bylo možné ji využít k dalšímu kroku, a to vyjádření vah kritérií.

K řádkům Saatyho matice byly vypočteny geometrické průměr, kde byla následně vyjádřena jejich suma, kterou bylo možné vyjádřit váhy kritérií (viz Tabulka 17).

	Geometrický průměr	Váha kritéria
V1	0,2738	0,0217
V2	0,4038	0,0320
V3	0,3241	0,0257
K1	1,3961	0,1107
K2	1,6877	0,1338
P1	1,9387	0,1537
P2	2,2270	0,1765
M1	1,8089	0,1434
M2	1,6207	0,1285
M3	0,9357	0,0742
∑	**12,6164**	**1,0000**

Tabulka 18 – Vypočtené váhy kritérií hodnocení frameworků

Z výsledků výpočtu vah je patrné, že nejvíce vlivná byla křivka učení a dokumentace, které jsou zaměřeny na použitelnost daného frameworku. Dále následovaly kritéria aspektu výkonu, a to ekosystém a knihovny, které naopak napomáhají vývojářům v jejich práci s frameworkem. Kritéria velikosti a aktivity příslušené komunity byla taky velice vlivná, jelikož díky komunitě je framework dále udržován, vyvíjen a jsou k němu vytvářeny další pomocné knihovny. Nejméně vlivnými veličinami byla kritéria výkonu. Je samozřejmě vhodné znát, jak framework přidává na výkonu aplikace. Ta se však může ve výsledku velice

lišit na základě kompetence vývojáře optimalizovat danou aplikaci. Tudíž by kritéria výkonu neměla být rozhodujícím aspektem výběru frameworku.

Je nutno podotknout, že vypočtené váhy jsou výsledkem subjektivního názoru odborníka z praxe, který byl osloven, aby provedl srovnání autorem vybraná kritéria.

Metoda váženého součtu

Metoda je založena na určení dílčích užitků variant u každé varianty, kde s pomocí určených vah lze následně vyjádřit celkový užitek variant. Dílčí užitky byly určeny u zvolených kritérií.

Jelikož jsou všechny kritéria díky bodování maximalizačního charakteru, nebylo potřeba je v rámci této metody převádět. Na následující tabulce (viz Tabulka 19) je zobrazeno hodnocení ve formě bodování jednotlivých variant a kritérií.

	V1	V2	V3	K1	K2	P1	P2	M1	M2	M3
React	40	14	24	35	20	35	33	5	12	40
Vue	43	15	25	19	8	34	36	2	17	43
Angular	39	10	23	22	6	34	28	2	25	39

Tabulka 19 – Hodnoty kritérií jednotlivých variant hodnocení frameworků

Následně byly určeny vektory ideální a bazální varianty, které byly zobrazeny v následující tabulce (viz Tabulka 9).

	V1	V2	V3	K1	K2	P1	P2	M1	M2	M3
D	39	10	23	19	6	34	28	2	12	22
H	43	15	25	35	20	35	36	5	25	37

Tabulka 20 – Bazální a ideální varianty hodnocení frameworků

Dále bylo potřeba sestavit kriteriální matici R. Hodnoty původní matice (Tabulka 19) byly normalizovány, kde výsledkem byla nová matice. Například křivka učení $j = 7$ u *Reactu* $i = 1$ byla s odpovídajícími hodnotami bazálního a ideálního vektoru normalizována následovně:

$$u_{1,7} = \frac{33 - 28}{36 - 28} = 0{,}6250$$

Obdobným způsobem byly vypočteny zbylé hodnoty. Hodnoty této tabulky vyjadřují dílčí užitky kritérií jednotlivých variant, které byly zobrazeny v následující tabulce (viz Tabulka 21).

	V1	V2	V3	K1	K2	P1	P2	M1	M2	M3
React	0,25	0,80	0,50	1,00	1,00	1,00	0,63	1,00	0,00	0,93
Vue	1,00	1,00	1,00	0,00	0,14	1,00	1,00	0,00	0,38	1,00
Angular	0,00	0,00	0,00	0,19	0,00	0,00	0,00	0,00	1,00	0,00

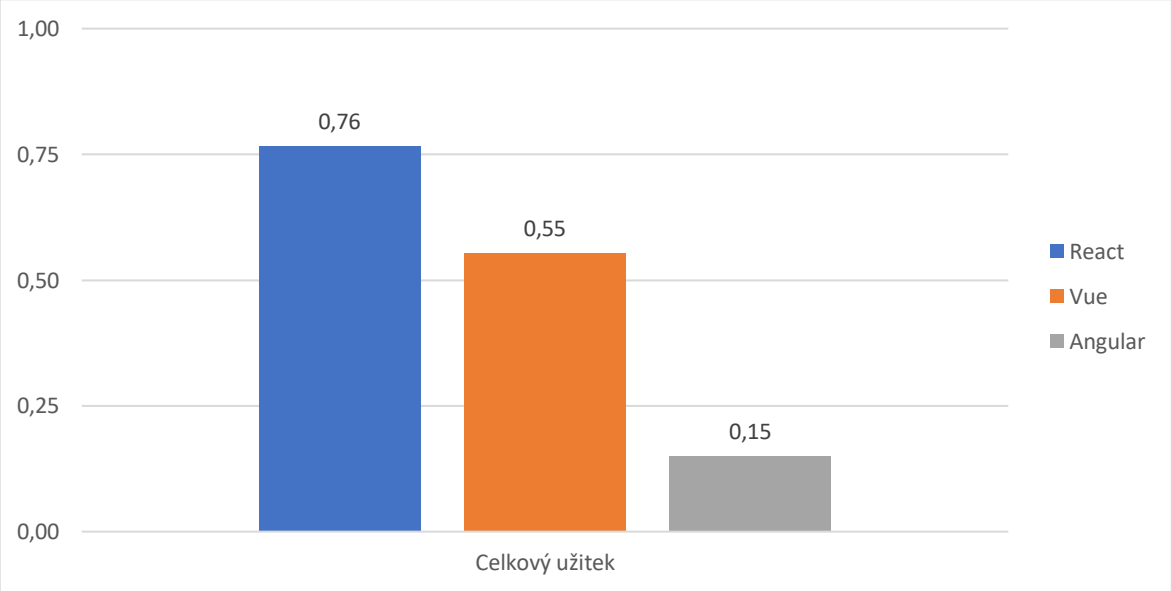
Tabulka 21 – Normalizovaná matice R výběru frameworků

Posledním krokem metody váženého součtu je vyjádření celkových užitků variant s použitím vah, které byly stanoveny Saatyho metodou. K tomu se lze dopracovat skalárním součinem dílčích užitků. Celkový užitek se například u varianty *React* $i = 1$ s patřičnými váhami vypočítal následovně:

$$u(V_1) = 0{,}25 * 0{,}02 + 0{,}80 * 0{,}03 + 0{,}50 * 0{,}03 + 1 * 0{,}11 + 1 * 0{,}13 + 1 * 0{,}15 + 0{,}63 * 0{,}18 + 1 * 0{,}14 + 0 * 0{,}13 \approx 0{,}76$$

	Celkový užitek	Pořadí
React	0,7649	1
Vue	0,5523	2
Angular	0,1492	3

Tabulka 22 – Celkové užitky variant hodnocení frameworků



Graf 2 – Celkové užitky variant hodnocení frameworků

Celkové užitky variant byly seřazeny a zobrazeny v tabulce výše (Tabulka 22). Hodnota v tomto konkrétním případě vyjadřuje, jak moc je framework jednoduchý na naučení a jaké nabízí možnosti v rámci vývoje, jako je například ekosystém nebo dostupné knihovny třetích stran. Dalšími vlivnými kritérii byla například aktivita a velikost komunity.

Nakonec si nejlépe opět vedl *React* s celkovým užitkem 0,76. Na druhém konci skončil *Angular* s celkovým užitkem 0,15, na kterém se nejvíce podílelo kritérium křivky učení.

5 Výsledky a diskuse

Kapitola se zabývá analýzou výsledků vlastní práce, která se zakládala na zhodnocení vybraných frameworků dle zvolených kritérií. Na základě výsledků byly v některých případech zjišťovány dodatečné informace, které napomohly k pochopení rozdílů mezi variantami.

5.1 Analýza zhodnocení vybraných frameworků

Vybrané frameworky si v rámci hodnocení aspektu výkonu vedly velice obdobně. Tento aspekt je však velice ovlivněn typem aplikace a do jaké míry je aplikace optimalizovaná. Tudíž velice záleží i na kompetenci vývojáře danou aplikaci optimalizovat. Největší rozdíl mezi frameworky byl patrný u aspektu komunity, kde byl na tom nejlépe *React*. U aspektu použitelnosti, a především křivce učení byl vyhodnocen *Angular* jako nejvíce náročný na naučení, kde na druhé straně se prosadil *Vue*, jakožto nejvíce jednoduchý na naučení. Aspekt vývoje byl nejvíce proměnlivý, kde u každého kritéria vedl jiný framework.

5.1.1 Manipulace DOM

React byl ve všech měřených manipulacích velice rychlý až na prohazování dvou řádků mezi 1000 řádků tabulky, kde byl ve srovnání s *Vue* velice pomalý. *Angular* měl stejný problém jako *React*, ale k navíc mu o něco déle trvalo promazat všechny prvky na stránce. *Vue* si vedl z vybraných frameworků nejlépe, avšak u barevného zvýraznění prvku byl na tom o něco hůř.

5.1.2 Spouštěcí metriky

Spouštěcí metriky vyšly u *Reactu* a *Vue* podobně, avšak byl zřejmý mírný rozdíl u stahování potřebných zdrojů, kde byl na tom stejně i *Angular*. Co se *Angularu* týče, ten si vedl podobně jako *React*, ale měl navíc velký problém s dobou kompilace a vyhodnocení potřebných skriptů na stránce. Zde byl rozdíl mnohem větší, jelikož místo 16ms, jako tomu bylo

u zbylých variant, byla doba téměř 140ms. Nicméně i přes takovýto rozdíl nebyla výrazně ovlivněna celková doba ujištění interaktivity pro uživatele.

5.1.3 Alokace paměti

U alokace paměti byly výsledky vybraných frameworků velice podobné až na pár rozdílů u *Reactu* a *Angularu*. *Vue* měl maximální počet bodů, jelikož si vedl nejlépe.

5.1.4 Velikost komunity

Velikost komunity byla vyhodnocena jako největší u *Reactu*, a to u všech srovnávaných komunikačních kanálů a poznatků. Framework je tudíž velice oblíbený a používaný, jak bylo možné vidět i na výsledcích průzkumu State of JS 2020. Díky takto velké komunitě je *React* obohacen o spousty výukových materiálů a dodatečných knihoven třetích stran.

U *Angularu* je velikost komunity značně menší, a to především v rámci chatovacích místností. Danou velikost ovlivnila i ta skutečnost, že je komunita rozdělena na dvě části, a to *AngularJS* a *Angular* 2. *AngularJS* byl kompletně přepracován na novou verzi, která je velice odlišná od své původní verze, a proto byly nakonec zpracovávány pouze informace týkající se aktuální verze.

Vue na úplném konci si vedl nejhůř. Nejslabší stránkou *Vue* byl počet využívajících vývojářů na GitHubu, kde byl přibližně 97% rozdíl od *Reactu*. Navíc počet přispěvatelů a počet zodpovězených otázek na Stack Overflow byl nejnižší mezi vybranými frameworky.

Bylo dodatečně zjištěno, že je *Vue*, oproti zbytku světa, nejvíce využívaný v Číně. Využívají ho například společnosti Xiaomi, Alibaba a Bilibili. To může být ze dvou důvodů. Autorem frameworku je původem z Číny, který svůj framework ve své rodné zemi představoval na různých konferencí. Je tam tedy velice známý. Druhým důvodem by mohlo být to, že Facebook a Google, jakožto vlastníci zbylých dvou frameworků, jsou v Číně zakázané. Tudíž čínští vývojáři dají prioritu frameworku jako je *Vue*. Navíc bylo na základě názorů komunity příslušného frameworku zjištěno, že je spousta knihoven psáno čínskými vývojáři, ke kterým většinou bývá pouze čínská dokumentace.

5.1.5 Aktivita komunity

Již z výsledků velikosti komunity bylo zřejmé, že nejlépe na tom bude s aktivitou *React*. Z výsledků je patrné, že je *React* nejvíce stahovaný a nejvíce konzultovaný (např. na Stack Overflow). Na druhé straně jsou *Vue* a *Angular*. Oba si vedly mezi sebou obdobně a od *Reactu* se výsledky lišily od 60% do 80%, což opět naznačuje na skutečnost, že je *React* nejvíce používaný. U *Angularu* byla opět počítána pouze ta část komunity, která se věnuje aktuální verzi.

5.1.6 Dokumentace

V případě vybraných frameworků jsou dokumentace velice dobře zpracované. Rozdíly byly nejvíce zřejmé u dodatečných funkcionalit procházení příslušných dokumentací a jejich jazykových možností.

Obsah dokumentace byl u *Reactu* dostačující, nicméně byl v určitých částech stručnější a vývojář je pak někdy nucen si danou problematiku vyhledat jinde na internetu. Nicméně je to i z toho důvodu, že *React* je pouze knihovnou na tvorbu uživatelských rozhraní. *Vue* dokumentace byla o něco více obsáhlá, jelikož zahrnuje i popis vlastního jednoduchého ekosystému. Nejvíce obsáhlá dokumentace byla u *Angularu*, jelikož se jedná o velice komplexní framework (resp. platformu), kde je popsána spousta náležitostí a jeho ekosystém.

Všechny frameworky v rámci dokumentace mají API referenci, tak návody, které se týkají například prvních kroků používání příslušného frameworku. Navíc je u všech možná i kontribuce ze strany komunity, kdyby náhodou bylo například potřeba něco upravit nebo přidat další návody. Dále byla hodnoceno, jestli dokumentace obsahují cheatsheet, který může být pro vybrané vývojáře velice přínosný, jelikož nemusí procházet celou dokumentaci. Cheatsheet měl oficiálně v dokumentaci pouze *Angular*, kdežto *React* ani *Vue* tuto možnost neměly. Nicméně nebyly u tohoto hodnoceny nejhůř, jelikož se tyto informace dají v dokumentaci dohledat.

Dalším hodnoceným poznatkem bylo zahrnutí interaktivních příkladů přímo do dokumentace. Zde vyhovoval pouze *Vue* (viz Příloha 34), který k příkladům přidal i interaktivní textový editor, aby si to vývojář mohl přímo v dokumentaci vyzkoušet.

Angular a *React* tuto možnost nemají, nicméně se odkazují u vybraných příkladů na jinou platformu, kde jsou dané příklady k dispozici.

V neposlední řadě bylo hodnoceno vyhledávání, kde na základě testování vyhledávání několika výrazů byly nejvíce vyhovující *React* a *Vue*. U obou případů se jednalo o velice dobře zvýrazněné a strukturované výsledky. Byly barevně zvýrazněné a uvedeny do kontextu, v jakém se nacházely. Výsledky vyhledávání u *Angularu* nebylo na druhé straně tak dobře strukturované, a ani zvýraznění nebylo tak výrazné. V příloze jsou výsledky vyhledávání po zadání klíčového slova „component“ (viz Příloha 31, Příloha 32, Příloha 33).

Co se týče jazyků, tak *React* zde měl velice navrch, jelikož měl v době psaní této práce dokumentaci k dispozici v 17 jazycích, kde dalších 31 bylo rozdělaných. *Vue* byl naopak v případě dokumentace nejaktuálnější verze přístupný pouze ve dvou jazycích včetně angličtiny. *Angular* měl k dispozici 5 jazyků, tudíž byl hodnocen o něco líp než *Vue*. Všechny ale podporují angličtinu, což je pořád uspokojivé.

5.1.7 Křivka učení

V první řadě bylo hodnoceno, jak se frameworky staví k JavaScriptové nadstavbě TypeScript. Je zde nutné připomenout, že hodnocení bylo založeno na předpokladu, že vývojář zná pouze jazyky HTML, CSS a JS, který navíc nikdy nepracoval s frameworky. U TypeScriptu se jedná v podstatě o nový jazyk, který doplňuje JavaScript o striktní typování. Všechny vybrané frameworky TypeScript podporují, ale u jediného *Angularu* je daný jazyk vynucený, což nutí vývojáře se naučit i příslušný jazyk. Na druhé straně s *Angularem* lze pracovat i bez TypeScriptu. Takto však v praxi nikdo nepracuje. Jednalo by se o mnohem složitější používání, jelikož by nemohl využívat možné deklarace pomocí dekorátorů.

Dále bylo hodnoceno psaní komponent a jejich logiky, která například spočívá s prací se stavem aplikace. V případě *Angularu* jde o značně složitý způsob psaní komponent, jelikož je k jejich definici potřeba využívat specifické konstrukce TypeScriptu. Navíc je nutné používat třídy, které jsou v JavaScriptu pojaty trochu jinak než v obyčejných objektově orientovaných jazycích, a práce s nimi může být někdy matoucí. V případě *Vue* je možné psát komponenty jako objekty se specifickými atributy, které může vývojář využívat. Psaní pomocí těchto objektů lze zapouzdřit do jednotlivých souborů, kde je navíc možné rozdělit

logiku od vykreslování. *React* je v tomto smyslu nejvíce jednoduchý. K tvorbě komponent se dříve využívaly třídy, které měly definované funkce v rámci životního cyklu komponenty. Po představení *React hooks*, bylo možné tyto komponenty psát pouze v rámci čistých funkcí. Nicméně je pořád možné dané třídy využívat. V příloze jsou ukázky kódu jednoduchých komponent na změnu hesla (viz Příloha 22, Příloha 23, Příloha 24)

Vykreslování a psaní šablon komponent je také značně rozdílné. *React* se od ostatních snaží využívat pouze jazyka JavaScript, kdežto *Angular* a *Vue* využívají šablonový systém. *React* k vykreslení komponenty využívá jazyka JSX (JavaScript XML). Jedná se o upravený JavaScript, který se na první pohled chová jako obyčejné HTML, ve kterém lze přímo JavaScript využívat. Logika komponenty se od jejího vykreslování tak neodděluje. Kód je velice jednoduchý na psaní, u složitějších komponent může vést k nepřehlednosti, ale to ovšem záleží na vývojáři, jak se k tomuto postaví. Psát lze i bez JSX, ale jedná se o složitější proces. Na druhé straně *Vue* a *Angular* používají šablony, které vzhled komponenty jasně oddělují od její logiky. Šablony se skládají z čistého HTML, který obsahuje doménově specifický jazyk. To oproti jazyka JSX vývojáře nutí si zapamatovat všechny možnosti zápisu jednosměrné vazby z uživatelského rozhraní ke zdroji a naopak (interpolace) a obousměrné vazby. *Vue* nicméně umožňuje i využití JSX. Vedle datových vazeb jsou také různé modifikátory, filtry a další direktivy. V příloze jsou uvedeny jednoduché ukázky psaní šablon (viz Příloha 25, Příloha 26, Příloha 27).

Bylo hodnoceno i využívání komponent. *React* k tomuto přistupuje nejjednodušeji, jelikož se jedná o obyčejný import a následné využití přímo v JSX. *Vue* to má obdobné, nicméně je potřeba nejdříve danou komponentu zaregistrovat do objektu, kde je využíván. Jedná se však o jednoduchý proces. U *Angularu* je to složitější, jelikož je potřeba komponenty registrovat do modulů (viz Příloha 24) a nemůžou existovat samostatně jako u *Reactu* nebo *Vue*. Vývojář je nucen dodatečně vytvářet i patřičné moduly (Angular, 2020).

V rámci hodnocení byla zohledněna i potřeba znalosti konceptu vstřikování závislostí (DI). Jedná se o techniku velice užitečnou, avšak poměrně matoucí pro vývojáře, kteří s tímto konceptem nikdy nepřišli do styku. Vstřikování závislostí je hodně používané ve frameworcích napsaných v objektově orientovaných jazycích, jako je například C# nebo PHP. Bylo hodnoceno, zdali vývojář o konceptu musí do jisté míry vědět. V případě *Reactu* a *Vue* se závislosti řeší pouze jednoduchými importy. *Angular* tuto techniku využívá. a proto je vývojář nucen znát alespoň základy.

Aby byla zaručena správná synchronizace dat v uživatelském rozhraní, jsou data provázána vazbou dat (data binding). *Vue* i *Angular* dokážou pracovat jak s jednosměrnými, tak i obousměrnými vazbami, kde obousměrné jsou jednodušší na údržbu a mohou být lépe pochopitelné. Jednosměrné vazby jsou založeny na neměnnosti a je potřeba k nim psát někdy více kódů. Proto mohou být méně pochopitelné. *React* oproti *Angularu* a *Vue* umožňuje pouze jednosměrné vazby. Zde je ale třeba podotknout, že to může být na druhé straně složitější. Vývojáři nemusí být hned jasné, v jakém případě by měl danou možnost využít. V příloze jsou ukázky práce s vazbou dat v rámci vykreslování (viz Příloha 25, Příloha 26, Příloha 27).

Důležitou součástí komponent je i asynchronní kód. U *Reactu* i *Vue* je práce s asynchronním kódem jednoduchá, a to především díky tomu, že je založená na využití obyčejných *Promises*, které náleží do samotného JavaScriptu. *Angular* nevyužívá pouze *Promises,* jako *React* a *Vue*, ale zakládá se také na využívání konceptů jako *observers, subscribers* a *pipes*.

Nakonec bylo hodnoceno i to, do jaké míry je vývojář nucen pracovat s knihovnami třetích stran. Nejvíce je tomu u *Reactu*, který je pouze knihovnou založenou na vytváření uživatelských rozhraní. Tudíž neobsahuje další ekosystém pro tvorbu aplikace. Vývojář si často k *React* složí vlastní ekosystém, který například zahrnuje práci s formuláři nebo složitější správu stavu aplikace. *Vue* je v tomto ohledu na tom o něco lépe, jelikož se jedná o jednodušší framework, který má k dispozici základní ekosystém. Co se týče *Angularu,* tak ten nepotřebuje k vytvoření aplikace téměř nic navíc, jelikož se jedná o velice komplexní framework. Většina běžně potřebných nástrojů, jako je i například zpracování formulářů, je již obsažena v jeho ekosystému.

5.1.8 Dostupné knihovny

S dostupnými knihovnami silně souvisí právě komunita. Zde je patrné, že nejvíce dostupných knihoven má právě *React*. Naopak *Angular* a *Vue* jsou oproti *React* velice pozadu. I přesto je dostupných knihoven velké množství. U *Reactu* může jít o velké množství knihoven, které jsou zaměřeny na velice specifické účely. Tudíž lze říct, že všechny frameworky mají dostatečné množství dostupných knihoven třetích stran.

5.1.9 Ekosystém

U ekosystému bylo hodnoceno, kolik nástrojů má příslušný framework ve svém ekosystému na základě běžných potřeb aplikací. Zde byl *React na* tom nejhůře, jelikož se jedná o knihovnu. Obsahuje především nástroje k manipulaci DOM a základní správu stavu. *Vue,* jakožto jednoduchý framework, má k dispozici nástroje k manipulaci DOM, správě stavu aplikace a routování. Jak u *Reactu*, tak i u *Vue* je nutné zbylé potřeby doplnit knihovnami třetích stran. Nejlépe na tom byl *Angular*, který ve svém ekosystému obsahuje vše potřebné k běžným potřebám aplikace.

5.1.10 Flexibilita

Nejvíce flexibilními frameworky byly *React* a *Vue*. *Angular*, jakožto velice komplexní framework má pevně daná pravidla vývoje. V první řadě je *React* i *Vue* možné zavádět do existujících aplikací, kde je možné takto dané aplikace i přepisovat. To samozřejmě u *Angularu* nelze, jelikož se jedná o komplexní framework, potažmo platformu. Tvorba komponent je u *Reactu* a *Vue* možné psát více způsoby, což zahrnuje i stylování. *Angular* má pouze jeden definovaný způsob. Na druhé straně má *Vue* a *Angular* možnost využívání více způsobů vazby dat, kdežto u *Reactu* toto lze řešit pouze jedním způsobem.

Co se týče vykreslování na straně serveru, tak to je možné u všech vybraných frameworků. Všechny lze využít na mobilní vývoj, a to i v rámci multiplatformního vývoje. K multiplatformnímu vývoji je potřeba dodatečných nástrojů jako je Ionic.

5.2 Analýza možností využití vybraných frameworků

Na základě poznatků z vlastní práce a dodatečného zjišťování bylo možné analyzovat možnosti využití vybraných frameworků. Na možnosti využití je možné nahlížet z více úhlů. Nejběžnější způsob je založený na potřebách budoucí aplikace. Je stěžejní, aby daný framework měl k dispozici nástroje a další funkcionality, které jsou potřebné k vývoji dané aplikace.

Frameworky však nabízejí spoustu nástrojů a technik k vývoji webových aplikací. V určitých případech může z pohledu potřeb aplikace vyhovovat najednou více frameworků,

jelikož nabízí obdobné nástroje k jejich realizaci. Proto bylo na tuto problematiku nahlíženo i z pohledu samotného vývojáře (resp. týmu vývojářů), kde byly brány v úvahu jeho dosavadní zkušenosti a preference.

V rámci vlastní práce byly analyzovány možnosti využití jak z pohledu potřeb aplikace, tak i z pohledu samotného vývojáře (resp. týmu vývojářů).

5.2.1 Z pohledu potřeb aplikace

Všechny hodnocené frameworky jsou vhodné na tvorbu webových aplikací, a to především na jednostránkové aplikace, na které jsou primárně zaměřeny. Pokud jde například o e-shop, kde je velice stěžejní indexace v rámci vyhledávacích nástrojů, tak všechny varianty umožňují velice dobrou optimalizaci SEO. Frameworky lze využít i na vícestránkové aplikace pomocí vykreslování na straně serveru, kde je možné jednotlivé stránky lépe identifikovat různými meta tagy apod.

Frameworky je možné použít na různé typy aplikací, a to i v případě potřeby mobilních zařízení. K posouzení této skutečnosti byly dodatečně naměřeny projekty Realworld [7] a TodoMVC[8]. Jedná se o komunitní projekty, kde je princip obdobný jako u měřené aplikace v rámci vlastní práce. U Realworld se jedná o aplikace napodobující publikační portál *medium.com*, napsané v různých frameworcích a ekosystémech. TodoMVC je projekt, kde se má aplikace zabývat správou seznamu úkolu. Na základě výsledků měření Lighgthouse (viz Příloha 35 a Příloha 36) je zřejmé, že výkon není závislý na frameworku, ale především o to, do jaké míry je vývojář schopen aplikaci optimalizovat. Není tedy možné tvrdit, že by byl v rámci výkonu, a to hlavně na mobilních zařízeních, konkrétní framework vhodnější než ostatní.

I v případě vývoje nativních mobilních aplikací je možné využít všech hodnocených variant. Nicméně je potřeba k příslušným frameworkům zvolit potřebný nástroj. V případě *Reactu* se jedná o nástroj *React Native*, u *Vue* je potřeba *Vue Native* a u *Angularu NativeScript*. NativeScript je možné využít i u *Vue*, jelikož se jedná o projekt, který se zaměřuje na využití

[7] Dostupné na https://github.com/gothinkster/realworld

[8] Dostupné na https://github.com/tastejs/todomvc

různých frameworků k multiplatformnímu vývoji aplikací. Ke stejnému účelu lze využít například i Ionic, který je dostupný i pro *React*.

Nicméně v případě zavedení frameworku do existující aplikace za účelem modernizace nebo pouze využití specifické funkcionality frameworku, je možné využít pouze *React* a *Vue*. Mezi takové funkcionality může například spadat využití formulářů. Tímto způsobem lze celou aplikaci postupně přepisovat do jiného frameworku (*React* nebo *Vue*). Může se totiž jednat o projekt napsaný v jakémkoliv jazyce nebo frameworku. Tedy pokud se ovšem jedná o webovou aplikaci. *Vue* je v tomto ohledu nejvíce jednoduchý v použití. Integrace totiž spočívá pouze ve vložení obyčejného CDN (případně i lokálně) do HTML. To stejné platí u *Reactu* až na jednu výjimku, která spočívá v menší konfiguraci, pokud chce vývojář využívat i JSX. *Angular* na druhé straně takto využít nelze, jelikož se jedná o komplexní framework (resp. platformu), kdežto *React* je knihovna a *Vue* jednoduchý framework.

5.2.2 Z pohledu vývojáře

Vývojáři se velice liší v rámci vlastních zkušeností a každý má svůj přístup k novým technologiím. Může se jednat o nové vývojáře, kteří umí základní jazyky jako HTML, CSS a JS. Na druhé straně může jít o vývojáře, který má zkušenosti s různými jazyky nebo dokonce dělal s různými frameworky. Je proto velice důležité brát v úvahu i dodatečné zkušenosti vývojáře.

Co se týče hodnocených frameworků, tak na základě křivky učení je *Vue* nejvíce jednoduchý. Ten se zakládá na využití obyčejného HTML, CSS a JS. V rámci dokumentace má framework mimo jiné i interaktivní příklady, které novým vývojářům mohou pomoci při pochopení fungování kódu. Pokud tedy jde o nové vývojáře, kteří nemají tolik zkušeností s programováním a nikdy nepracovali ve frameworku, tak je možné *Vue* považovat za vhodnou variantu. *React* je také v tomto případě vhodný, jelikož je celý kód založen na používání JS, avšak právě JSX může být pro některé složitější na pochopení. U nového *Reactu* může být i správa stavu aplikace ze začátku matoucí. Na druhé straně je *Angular*, který je na základě křivky učení velice složitý pro nového vývojáře. Nicméně to nemusí platit právě u vývojářů, kteří například přicházejí z jiných frameworků nebo mají zkušenosti v objektově orientovaných jazycích. Pro takové vývojáře nemusí být vynucený TypeScript vůbec problém, a naopak jim vyhovuje více.

Nicméně i nový vývojář by mohl z vlastních důvodů preferovat složitější framework. Proto by mohlo být vhodnější frameworky doporučit například na základě preferencí vývojáře. Nemusí se však jednat o preference pouze jednoho vývojáře, ale také celého týmu vývojářů.

React lze doporučit vývojářům, kteří preferují značnou volnost vývoje webových aplikací nebo si například chtějí vytvářet vlastní ekosystém s pomocí knihoven třetích stran. *React* lze doporučit i na základě jednoduchého vytváření komponent, kde si vývojář může vybrat cestu jednoduchých funkcí nebo tříd. V *Reactu* je navíc vše ve formě JavaScriptu, tudíž i šablony jsou na tomto založeny. Je navíc možné využít i striktně typovaný jazyk TypeScript, který není vynucený, jako například u *Angularu*. Nakonec je *React* vhodný pro vývojáře, kteří značně preferují velkou a aktivní komunitu. Důležitým poznatkem *Reactu* je to, že se jedná o knihovnu, a nikoliv o framework. Komunitou je někdy označován jako framework, jelikož se k němu napojuje mnoho dalších knihoven třetích stran. V některých případech se může z takového složeného ekosystému stát samostatný framework.

Vue je možné doporučit vývojářům, kterým jde naopak o to, aby byl framework co nejjednodušší na naučení. Kód je u *Vue* velice přehledný a v rámci tvorby komponent jsou technologie (resp. jazyky) jasně oddělené. Vývojář má možnost vytvářet komponenty i jednosouborovým způsobem (Single File Component), kde jsou definované bloky podle jazyka HTML, CSS a JS. *Vue* je navíc vhodný pro ty, kterým vyhovují šablonové systémy, kde je HTML obohacen o doménově specifický jazyk. V případě potřeby je možné využít i TypeScript.

Angular lze doporučit vývojářům, kteří preferují velice stabilní vývoj bez značné potřeby využívání knihoven třetích stran. Lze doporučit těm, kteří preferují striktně typovaný jazyk TypeScript, který se zakládá na technikách objektově orientovaných jazyků, jako je příklad C#. Stejně jako u *Vue* jsou komponenty *Angularu* odděleny podle jazyka. Avšak s tím rozdílem, že *Angular* všechny odděluje do patřičných souborů. Je navíc nutno podotknout, že ke stylování má pouze jedinou možnost, a to preprocesor SCSS. *Angular* stejně jako *Vue* řeší šablony s pomocí obyčejného HTML a specifickým doménovým jazykem. Nicméně díky pevně dané architektuře a jasným pravidlům vývoje jsou všechny aplikace tvořeny téměř stejným způsobem.

6 Závěr

Diplomová práce byla tematicky zaměřena na vývoj webových aplikací pomocí JavaScriptových frameworků. Hlavním cílem bylo analyzovat a zhodnotit vybrané frameworky pro vývoj webových aplikací. Dílčími cíli bylo charakterizovat problematiku vývoje, zhodnotit a porovnat vybrané JavaScriptové frameworky a nakonec analyzovat možnosti jejich využití v praxi.

V první části práce byla provedena charakteristika vývoje webových aplikací a popis různých technologií a náležitostí, které jsou spojeny s vývojem pomocí jazyka JavaScript. Byla blíže popsána metoda hodnocení, která byla založena na metodách vícekriteriální analýzy variant, a to konkrétně Saatyho metoda a metoda váženého součtu.

V rámci vlastní práce bylo nejdříve na základě dotazníku *State of JS 2020* vybráno devět možných JavaScriptových frameworků k hodnocení. S ohledem na doporučený rozsah diplomové práce byly k samotnému hodnocení vybrány pouze tři frameworky. Výběr byl založen na základních informací, které byly k daným frameworkům dostupné, a to i názory vývojářů vycházejících z výsledků provedeného průzkumu *State of JS 2020*. Kritéria výběru tedy zahrnovala *rok vydání* frameworku, *oblíbenost na GitHubu, počet stažení za týden* a *míře spokojenosti* a *užití* daného frameworku vývojáři po celém světě. Zvoleným modelem vícekriteriální analýzy variant byl vyjádřen celkový užitek všech devíti frameworků. Byly vybrány pouze tři frameworky s nejvyšším celkovým užitkem byly vybrány, a to *React, Vue* a *Angular*. Vybrané frameworky byly vyhodnocené jako dostatečně vyspělé, oblíbené a užívané vývojáři v praxi.

V další řadě bylo potřeba určit kritéria hodnocení zvolených frameworků, které bylo provedeno z pohledu (aspektu) výkonu, komunity, použitelnosti a vývoje. U výkonu se hodnotilo, jak je aplikace napsaná v příslušeném frameworku výkonná. V rámci komunity byla hodnocena velikost a aktivita komunity frameworků. Aspekt použitelnosti se zabýval obsahem dokumentace a křivkou učení spojenou s daným frameworkem. Nakonec byl hodnocen i samotný vývoj za použití vybraných frameworků. V rámci těchto aspektů byla definována kritéria: *manipulace objektového modelu dokumentu (DOM), spouštěcí metriky, alokace paměti, velikost komunity, aktivita komunity, dokumentace, křivka učení, dostupné knihovny, ekosystém* a *flexibilita*. Frameworky byly mezi sebou na základě těchto kritérií porovnány a bodovány (resp. hodnoceny) podle určených stupnic (kvantitativní a kvalitativní).

Zvoleným modelem vícekriteriální analýzy variant byly na základě hodnocených kritérií vyjádřeny celkové užitky frameworků. Výsledek byl však velice ovlivněn zvolenými váhami příslušných kritérií, které byly stanoveny na základě názoru odborníka v praxi. Nejvíce vlivným kritériem byla *křivka učení*, za kterou postupně následovaly kritéria *dokumentace, dostupné knihovny, aktivita komunity, ekosystém, velikost komunity, flexibilita, spouštěcí metriky, alokace paměti* a *manipulace DOM*.

U *Reactu* byl celkový užitek nejvyšší, jelikož je oproti zbylým frameworkům nejvíce podpořen komunitou, která je navíc spjata s dostupnými knihovnami a ekosystémem. *Vue* byl na druhém místě, a to především kvůli malé komunitě. Byl však vyhodnocen jako nejjednodušší na naučení. Na posledním místě se umístil *Angular*, jelikož má vysokou křivku učení a komunita není tak rozsáhlá jako u *Reactu*.

Angular je složitější, jelikož se jedná o komplexní framework, který v sobě nese spoustu nástrojů a funkcionalit, a má striktní pravidla vývoje. *Vue* je naopak jednoduchý framework se základním ekosystémem. Na druhé straně je *React*, který je pouze knihovnou, tudíž se běžně kombinuje s knihovnami třetích stran. Výsledek takové kombinace může vést k vytvoření takové struktury, který se podobá jednoduchému frameworku. Proto je *React* někdy považován za framework, ačkoliv tomu tak není.

Poslední částí vlastní práce byla analýza možností využití hodnocených frameworků, kde na tuto problematiku bylo nahlíženo hlavně z pohledu potřeb aplikace. Na základě poznatků získaných v rámci hodnocení bylo zjištěno, že všechny varianty je možné použít na vývoj webových aplikací, které jsou zaměřeny na uživatele, a to především ve formě jednostránkových aplikací. Je nicméně možné frameworky využít i na mobilní, případně i multiplatformní vývoj. Lze je s pomocí vykreslování na straně serveru využít i při tvorbě například e-commerce aplikací, které se zakládají na optimalizaci vyhledávačů (SEO).

Jelikož jsou hodnocené frameworky vhodné na stejné typy projektů, bylo na problematiku analýzy možností využití nahlíženo i z pohledu samotného vývojáře (resp. týmu vývojářů). Frameworky lze vývojářům doporučit na základě jejich dosavadních zkušeností a preferencí. Vývojářům, kteří preferují volnost vývoje bez striktních pravidel, kde je navíc vše řešeno JavaScriptem, je možné doporučit *React*. *Vue* lze doporučit těm, kteří nemají tolik zkušeností a chtějí něco jednoduššího, kde jsou jazyky (HTML, CSS, JS) jasně rozděleny. Těm, kteří preferují stabilní a striktní vývoj a zároveň jim vyhovuje TypeScript, který se více podobá objektově orientovaným jazykům (například C#), lze doporučit naopak *Angular*.

7 Seznam použitých zdrojů

Angular. 2020. Introduction to Angular concepts. *Angular.* [Online] 2020. https://angular.io/guide/architecture.

—. 2020. Introduction to the Angular Docs. *Angular.* [Online] 4. 12 2020. https://angular.io/docs.

Brožová, Helena, Houška, Milan a Šubrt, Tomáš. 2014. *Modely pro vícekriteriální rozhodování.* Praha : Česká zemědělská univerzita, 2014. 978-80-213-1019-3.

Casteleyn, Sven, Dolog, Peter a Pautasso, Cesare. 2016. *Current Trends in Web Engineering.* místo neznámé : Springer, 2016. 3319469630.

Cavaness, Chuck. 2004. *Programming Jakarta Struts.* Newton : O'Reilly Media, Inc., 2004. 9780596006518.

Cudré-Mauroux, Philippe, a další. 2013. NoSQL Databases for RDF: An Empirical Evaluation. [Online] 2013. [Citace: 19. 3 2021.] https://link.springer.com/chapter/10.1007/978-3-642-41338-4_20.

Edwin, Njeru Mwendi. 2014. Software Frameworks, Architectural and Design Patterns. *Journal of Software Engineering and Applications.* 2014.

Fiala, Petr, Jablonský, Josef a Maňas, Miroslav. 1994. *Vícekriteriální rozhodování.* Praha : Vysoká škola ekonomická, 1994. 80-7079-748-7.

Flanagan, D. 2002. *JavaScript : the definitive guide.* CA : O'Reilly, 2002. 0-596-00048-0.

Green, Brad. 2013. *AngularJS.* CA : O'Reilly, 2013. 978-1449344856.

Greif, Sacha a Benitte, Raphaël. 2020. Demographics. *State of JS 2020.* [Online] 2020. https://2020.stateofjs.com/en-US/demographics/.

—. 2020. Front-end Frameworks. *State of JS 2020.* [Online] 2020. https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/.

—. 2020. State of JS 2020. *State of JS 2020.* [Online] 2020. https://2020.stateofjs.com/.

Hickson, Ian a Hyatt, David. 2017. HTML5 specification - w3.org. *World Wide Web Consortium.* [Online] 2017. [Citace: 18. 7 2020.] http://www.w3.org/TR/html5/.

Internet Society. 2003. Brief History of the Internet. *Internetsociety.org.* [Online] 2003. [Citace: 12. 3 2020.] http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet.

Jazayeri, Mehdi. 2007. Some Trends in Web Application Development. [Online] 2007. [Citace: 22. 11 2020.] http://ieeexplore.ieee.org/document/4221621.

Khaliluzzaman, Md. a Chowdhury, Iftekher Islam. 2016. Pre and post controller based MVC architecture for web application. [Online] 2016. [Citace: 2. 3 2020.] http://ieeexplore.ieee.org/document/7760064.

Lopez, Lionel. 2017. *React: QuickStart Step-By-Step Guide to Learning React JavaScript Library.* Scotts Valley, CA : Createspace Independent Publishing Platform, 2017. 978-1976210235.

Macrae, Callum. 2018. *Vue.js - Up and Running.* Sebastopol, CA : O'Reilly, 2018. 978-1491997246.

Mcheick, Hamid a Qi, Yan. 2011. Dependency of components in MVC distributed architecture. [Online] 2011. [Citace: 1. 9 2020.] http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000006030542.

Microsoft. 2020. TypeScript Documentation. *TypeScript.* [Online] 12 2020. https://www.typescriptlang.org/docs.

Mozilla. 2020. SPA (Single-page application). *MDN Web Docs.* [Online] 2020. https://developer.mozilla.org/en-US/docs/Glossary/SPA.

Myers, Mark. 2014. *A Smarter Way to Learn JavaScript: The new tech-assisted approach that requires half the effort.* Scotts Valley, CA : CreateSpace Independent Publishing Platform, 2014. 978-1497408180.

MySQL. 2020. MySQL Internals Manual. *Dev.mysql.com.* [Online] 2020. [Citace: 10. 8 2020.] http://dev.mysql.com/doc/internals/en/index.html.

—. 2020. Supported Platforms: MySQL Database. *MySQL.* [Online] 2020. [Citace: 1. 12 2020.] http://www.mysql.com/support/supportedplatforms/database.html.

Node.js. 2020. About. *Node.js.* [Online] 1. 12 2020. https://nodejs.org/en/about.

npm trends. 2021. *npm trends.* [Online] 2021. https://www.npmtrends.com/react-vs-@angular/core-vs-vue-vs-ember-source-vs-svelte-vs-preact-vs-@stimulus/core-vs-alpinejs-vs-@polymer/lit-element.

Patil, Harshal. 2019. Contemporary Front-end Architectures. *webf.* [Online] 2019. https://blog.webf.zone/contemporary-front-end-architectures-fb5b500b0231.

Powell, Thomas A. 2008. Ajax: The Complete Reference. [Online] 2008. [Citace: 11. 11 2020.] https://amazon.com/ajax-complete-reference-thomas-powell/dp/007149216x.

React. 2020. Getting Started. *React.* [Online] 22. 10 2020. https://reactjs.org/docs.

Saaty, T. a & Tran, L. 2007. On the invalidity of fuzzifying numerical judgments in the Analytic Hierarchy Process. *Mathematical and Computer Modelling.* 10 2007, Sv. 46.

Saaty, T. 2000. *Fundamentals of Decision Making and Priority Theory With the Analytic Hierarchy Process.* Pittsburgh : RWS Publications, 2000. 9781888603156.

Salas-Zárate, Marí del Pilar, Alor-Hernández, Giner a Rodríguez-González, Alejandro. 2012. *Developing Lift-based Web Applications Using Best Practices.* 2012. stránky 214-223. Sv. 3.

Shklar, Leon a Rosen, Rich. 2009. *Web Application Architecture: Principles, Protocols and Practices.* Hoboken : Wiley, 2009. ISBN 9780470518601.

Stack Overflow . 2020. Programming, Scripting, and Markup Languages. *Stack Overflow Developer Survey 2020.* [Online] 2020. https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages.

Stack Overflow. 2020. Developer Survey 2020. *Stack Overflow Developer Survey 2020.* [Online] 2020. https://insights.stackoverflow.com/survey/2020.

—. 2020. Geography. *Stack Overflow Developer Survey 2020.* [Online] 2020. https://insights.stackoverflow.com/survey/2020#geography.

—. 2020. Most Important Job Factors. *Stack Overflow Developer Survey 2020.* [Online] 2020. https://insights.stackoverflow.com/survey/2020#work-most-important-job-factors.

—. 2020. Visiting Stack Overflow. *Stack Overflow Developer Survey 2020.* [Online] 2020. https://insights.stackoverflow.com/survey/2020#community-visiting-stack-overflow.

—. 2020. What do you do when you get stuck. *Stack Overflow Developer Survey 2020.* [Online] 2020. https://insights.stackoverflow.com/survey/2020#technology-what-do-you-do-when-you-get-stuck.

Subreddit Stats. 2021. *Subreddit Stats.* [Online] 2021. https://subredditstats.com.

Šubrt, Tomáš. 2019. *Ekonomicko-matematické metody.* Praha : Vydavatelství a nakladatelství Aleš Čeněk, 2019. 978-80-7380-762-7.

Toal, Ray a Dionisio, John David. 2010. *The JavaScript Programming Language.* Burlington : Jones & Bartlett Learning, 2010. 9780763766580.

Toal, Ray, a další. 2017. *Programming Language Explorations.* London : Chapman & Hall, 2017. 9781498738460.

Triantaphyllou, Evangelos. 2000. *Multi-criteria decision making methods: a comparative study.* Amsterdam : Kluwer Academic Publishers, 2000. 0-7923-6607-7.

Vostrovský, Václav. 2014. *Vytváření databází v Oracle.* Praha : Česká zemědělská univerzita v Praze, 2014. 978-80-213-1191-6.

Vue. 2020. Introduction. *Vue.js.* [Online] 1. 12 2020. https://v3.vuejs.org/guide.

W3Schools. 2020. AJAX Introduction. *W3Schools.com.* [Online] 2020. [Citace: 7. 9 2020.] https://www.w3schools.com/xml/ajax_intro.asp.

—. 2020. JavaScript HTML DOM. *W3Schools.com.* [Online] 2020. [Citace: 9. 7 2020.] https://www.w3schools.com/js/js_htmldom.asp].

WHATWG. 2021. Living Standard. *DOM Standard.* [Online] 16. 3 2021. https://dom.spec.whatwg.org.

Williamson, Ken. 2015. *Learning AngularJS.* Sebastopol, CA : O'Reilly, 2015. 978-1491916759.

World Wide Web Consortium. 2018. CSS - Cascading Style Sheets home page. *W3.org.* [Online] 2018. [Citace: 10. 1 2020.] http://www.w3.org/Style/CSS/#browsers.

—. 2018. HTML 4.01 Specification. *W3.org.* [Online] 2018. [Citace: 2. 2 2020.] http://www.w3.org/TR/html401/struct/links.html.

—. 2017. HTML 5.2 W3C Recommendation. *w3.org.* [Online] 2017. [Citace: 12. 12 2020.] https://www.w3.org/TR/html/introduction.html#introduction-history.

—. 2014. The web standards model - HTML CSS and JavaScript. *W3.org.* [Online] 2014. [Citace: 11. 11 2018.] https://www.w3.org/wiki/The_web_standards_model_-_HTML_CSS_and_JavaScript.

Zakas, Nicholas C. 2016. *Understanding Ecmascript 6.* San Francisco, CA : No Starch Press, 2016. 978-1593277574.

Žára, Ondřej. 2016. *JavaScript Programátorské techniky a webové technologie.* Brno : Computer Press, 2016. 978-80-251-4573-9.

8 Přílohy

8.1 Bodování jednotlivých kritérií

Manipulace DOM	React	Vue	Angular
Vytvoření 1000 řádků	5	5	5
Nahrazení 1000 řádků	5	5	5
Částečná aktualizace mezi 10 000 řádky	5	5	5
Zvýraznění řádku	5	3	5
Prohození dvou řádků mezi 1000 řádky	1	5	1
Odstranění řádku z 1000 řádků	5	5	5
Vytvoření 10 000 řádků najednou	4	5	5
Přidání 1000 řádků z 10 000 řádkům	5	5	5
Odstranění 10 000 řádků najednou	5	5	3
∑ (max 45)	**40**	**43**	**39**

Příloha 1 – Bodování – Manipulace DOM

Spouštěcí metriky	React	Vue	Angular
Konzistentní interaktivita (TTI)	5	5	5
Doba spouštění skriptu	5	5	1
Náklady na síťový přenos	4	5	4
∑ (max 15)	**14**	**15**	**10**

Příloha 2 – Bodování – Spouštěcí metriky

Alokace paměti	React	Vue	Angular
Zátěž při načtení stránky	5	5	4
Zátěž při přidání 1000 řádků	5	5	5
Zátěž při částečné aktualizaci (pět cyklů)	4	5	5
Zátěž při vytváření 1000 řádků (pět cyklů)	5	5	5
Zátěž při vytváření a mazání 1000 řádků (pět cyklů)	5	5	4
∑ (max 25)	**24**	**25**	**23**

Příloha 3 – Bodování – Alokace paměti

Velikost komunity	React	Vue	Angular
Počet hvězd na GitHubu	5	5	3
Počet využívajících na GitHubu	5	1	2
Počet přispěvatelů na GitHubu	5	2	5
Počet sledujících na Twitteru	5	3	5
Počet odběratelů na Redditu	5	2	1
Počet vyřešených otázek na Stack Overflow	5	2	5
Počet členů v chatovací místnostech	5	4	1
∑ (max 35)	**35**	**19**	**22**

Příloha 4 – Bodování – Velikost komunity

Aktivita komunity	React	Vue	Angular
Počet stažení na *npm* (1 týden)	5	2	2
Počet vyřešených otázek na Stack Overflow (30 dní)	5	2	2
Počet příspěvků na Redditu (1 den)	5	2	1
Počet komentářů na Redditu (1 den)	5	2	1
∑ (max 20)	**20**	**8**	**6**

Příloha 5 – Bodování – Aktivita komunity

Dokumentace	React	Vue	Angular
Obsah dokumentace	3	4	5
Možnost kontribuce	5	5	5
API reference	5	5	5
První kroky a další návody	5	5	5
Cheatsheet	4	4	5
Interaktivní příklady	3	5	3
Vyhledávání	5	5	3
Dostupné jazyky	5	2	3
∑ (max 40)	**35**	**35**	**34**

Příloha 6 – Bodování – Dokumentace

Křivka učení	React	Vue	Angular
Práce s TypeScript	5	5	3
Psaní komponent	5	4	3
Psaní vykreslování a šablon	4	5	3
Způsob použití komponent	5	4	3
Znalost techniky Dependency Injection	5	5	3
Práce s vazbou dat (data binding)	3	5	5
Práce s asynchronním kódem	5	5	3
Míra potřeby práce s knihovnami třetích stran	1	3	5
∑ (max 40)	**33**	**36**	**28**

Příloha 7 – Bodování – Křivka učení

Dostupné knihovny	React	Vue	Angular
Počet balíčků na *npm*	5	2	2
∑ (max 5)	**5**	**2**	**2**

Příloha 8 – Bodování – Dostupné knihovny

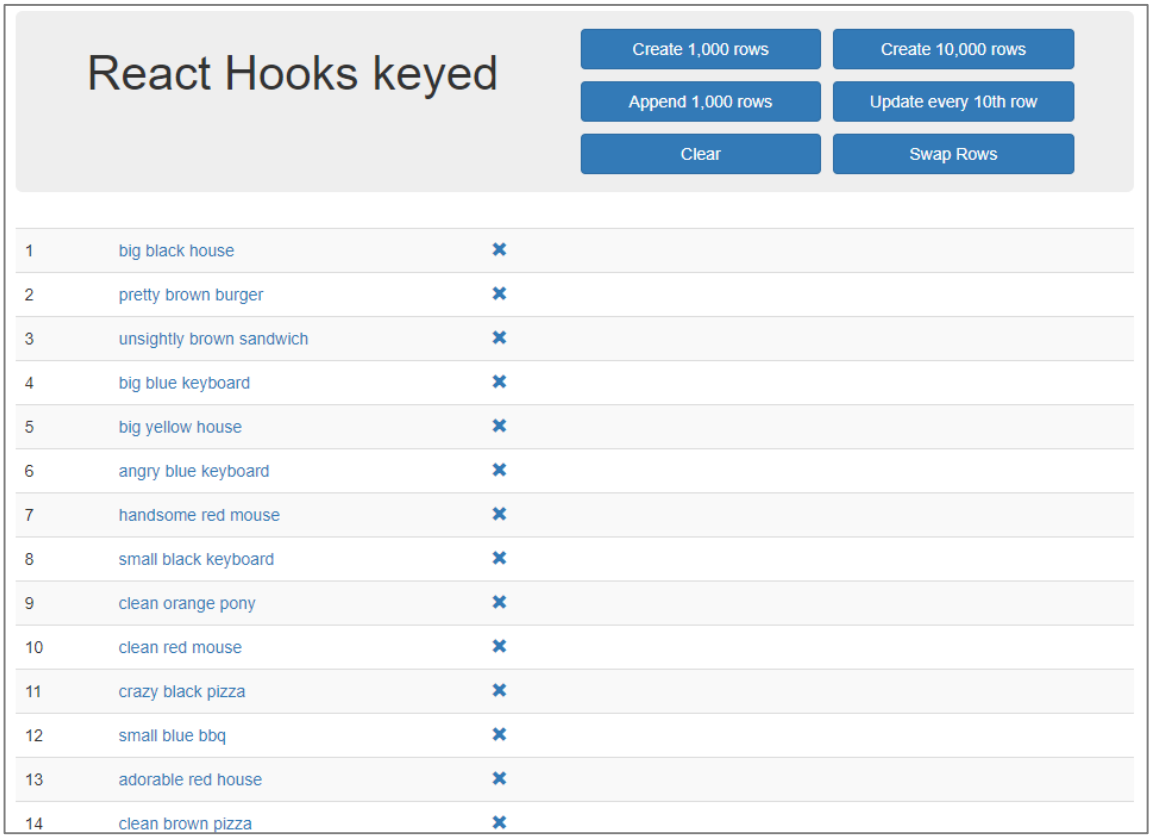
Ekosystém	React	Vue	Angular
Manipulace DOM	5	5	5
Správa stavu	4	5	5
Routing	1	5	5
Validace a zpracování formulářů	1	1	5
Testovací nástroje	1	1	5
∑ (max 25)	**12**	**17**	**25**

Příloha 9 – Bodování – Ekosystém

Flexibilita	React	Vue	Angular
Možnost zavést do existujícího projektu	4	5	1
Nemá striktní pravidla vývoje	5	3	1
Možnosti vazeb dat (data binding)	3	5	5
Možnosti vykreslování na straně serveru	5	5	5
Možnost multiplatformního vývoje	5	5	5
Možnosti psaní komponent	5	5	1
Možnosti psaní vykreslování/šablon	4	5	1
Možnosti stylování	5	4	3
∑ (max 40)	**36**	**37**	**22**

Příloha 10 – Bodování – Flexibilita

8.2 Měření výkonu s js-framework-benchmark



Příloha 11 – Ukázka aplikace projektu js-framework-benchmark

Příloha 12 – Ukázka spouštění měření aplikace projektu js-framework-benchmark

Příloha 13 - Ukázka provádění měření aplikace projektu js-framework-benchmark

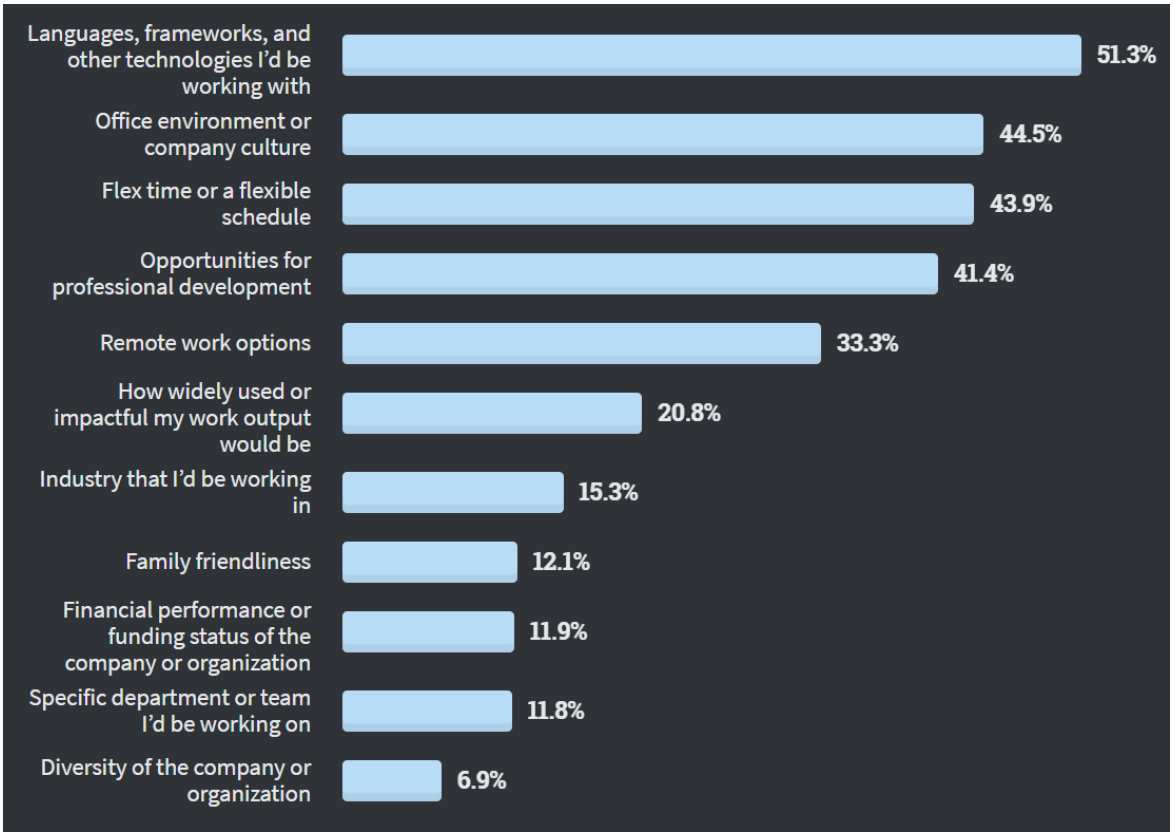
Name Duration for...	vue-v3.0.2	react-hooks-v17.0.1	angular-ng-v11.0.2
Implementation notes			
create rows creating 1,000 rows	131.8 ± 1.6 (1.00)	155.8 ± 2.2 (1.18)	161.4 ± 1.8 (1.22)
replace all rows updating all 1,000 rows (5 warmup runs).	116.1 ± 0.7 (1.00)	128.0 ± 1.1 (1.10)	130.9 ± 1.1 (1.13)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	171.6 ± 7.1 (1.08)	172.0 ± 1.6 (1.09)	158.2 ± 4.1 (1.00)
select row highlighting a selected row. (no warmup runs). 16x CPU slowdown.	161.9 ± 4.1 (2.07)	80.0 ± 7.7 (1.02)	78.2 ± 3.4 (1.00)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	54.2 ± 1.2 (1.00)	414.3 ± 2.8 (7.65)	422.3 ± 2.1 (7.79)
remove row removing one row. (5 warmup runs).	23.2 ± 0.4 (1.05)	22.2 ± 0.2 (1.00)	25.4 ± 0.8 (1.15)
create many rows creating 10,000 rows	1,103.9 ± 3.9 (1.00)	1,558.7 ± 37.3 (1.41)	1,292.8 ± 16.6 (1.17)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown	256.6 ± 4.0 (1.00)	283.6 ± 3.6 (1.11)	299.3 ± 3.5 (1.17)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown	127.7 ± 0.9 (1.00)	136.5 ± 1.6 (1.07)	246.9 ± 3.6 (1.93)
geometric mean of all factors in the table	1.10	1.38	1.47

Name	vue-v3.0.2	react-hooks-v17.0.1	angular-ng-v11.0.2
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,105.5 ± 9.8 (1.00)	2,581.4 ± 2.1 (1.23)	2,439.6 ± 3.7 (1.16)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	16.0 ± 0.0 (1.00)	16.0 ± 0.0 (1.00)	138.7 ± 4.9 (8.67)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	196.5 ± 0.0 (1.00)	260.0 ± 0.0 (1.32)	294.7 ± 0.0 (1.50)
geometric mean of all factors in the table	1.00	1.17	2.47

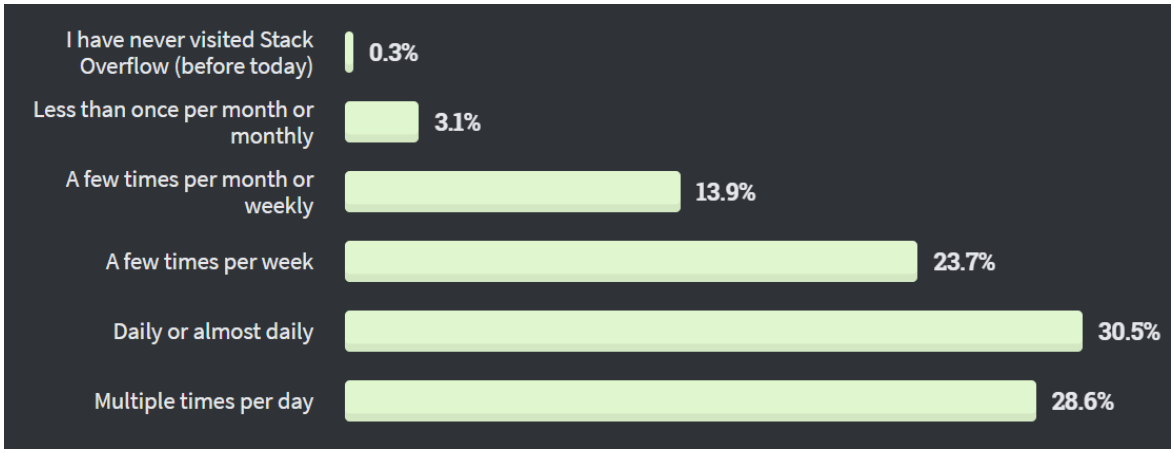
Name	vue-v3.0.2	react-hooks-v17.0.1	angular-ng-v11.0.2
ready memory Memory usage after page load.	1.2 (1.00)	1.3 (1.09)	1.8 (1.48)
run memory Memory usage after adding 1000 rows.	3.5 (1.00)	4.0 (1.14)	4.2 (1.20)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	3.7 (1.00)	4.8 (1.30)	4.6 (1.23)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	4.1 (1.00)	5.1 (1.23)	5.0 (1.21)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	2.6 (1.00)	3.1 (1.17)	3.5 (1.32)
geometric mean of all factors in the table	1.00	1.18	1.28

Příloha 14 – Výsledky měření výkonu v rámci projektu js-framework-benchmark

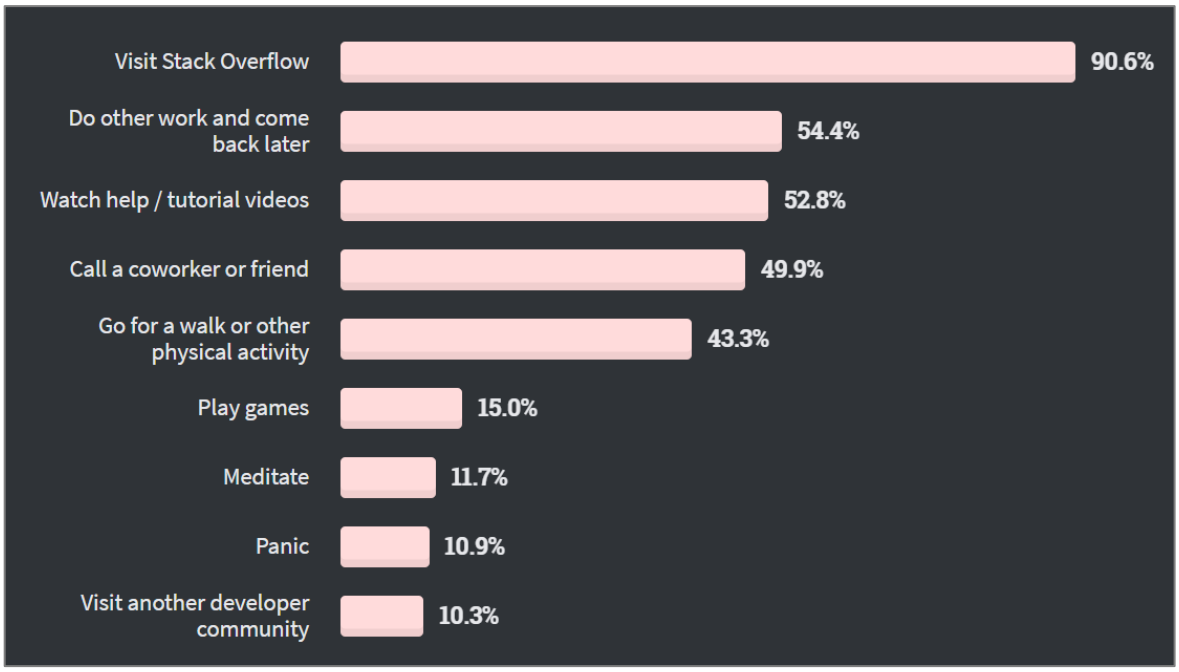
8.3 Vybrané výsledky z průzkumů



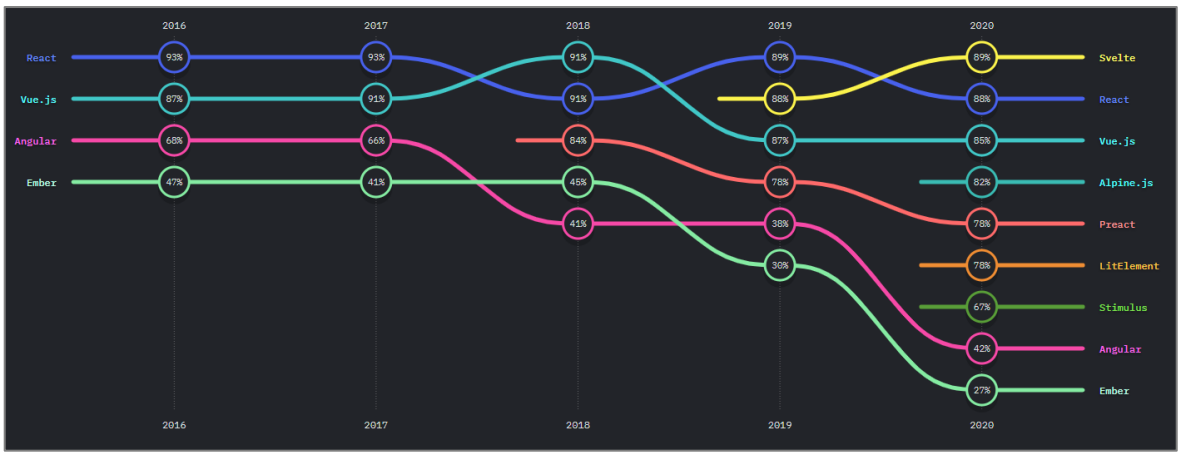
Příloha 15 – Pracovní priority vývojářů (Stack Overflow, 2020)



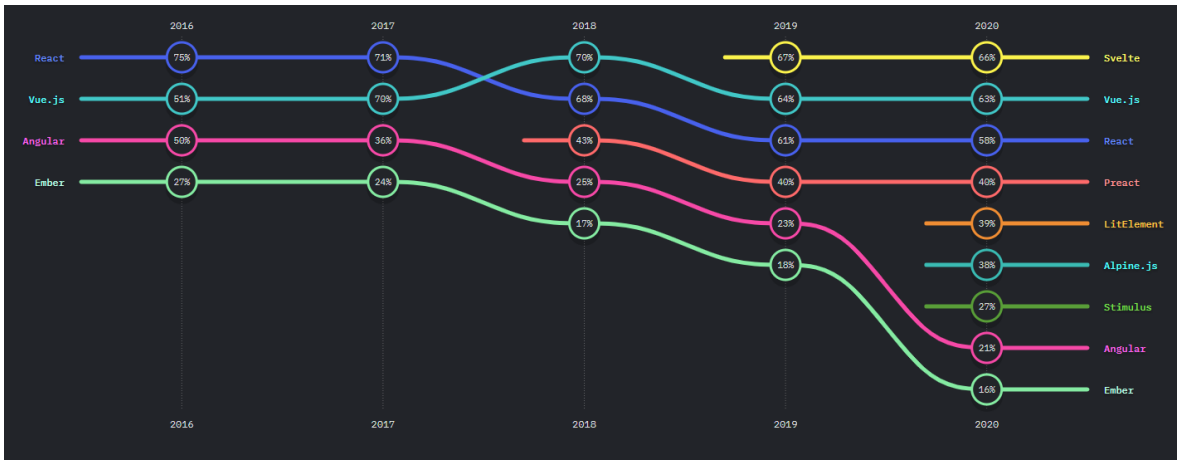
Příloha 16 – Návštěvnost portálu Stack Overflow (Stack Overflow, 2020)



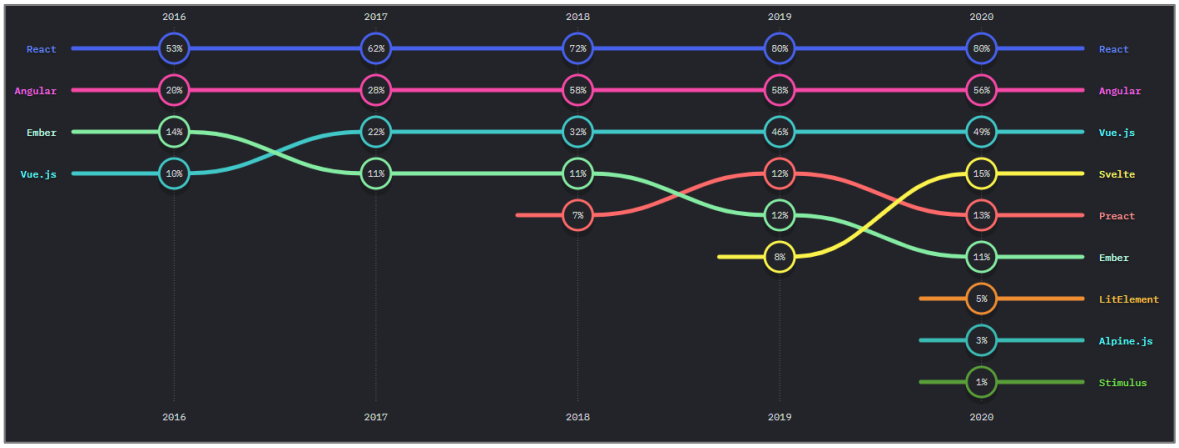
Příloha 17 – Způsob řešení problémů vývojářů (Stack Overflow, 2020)



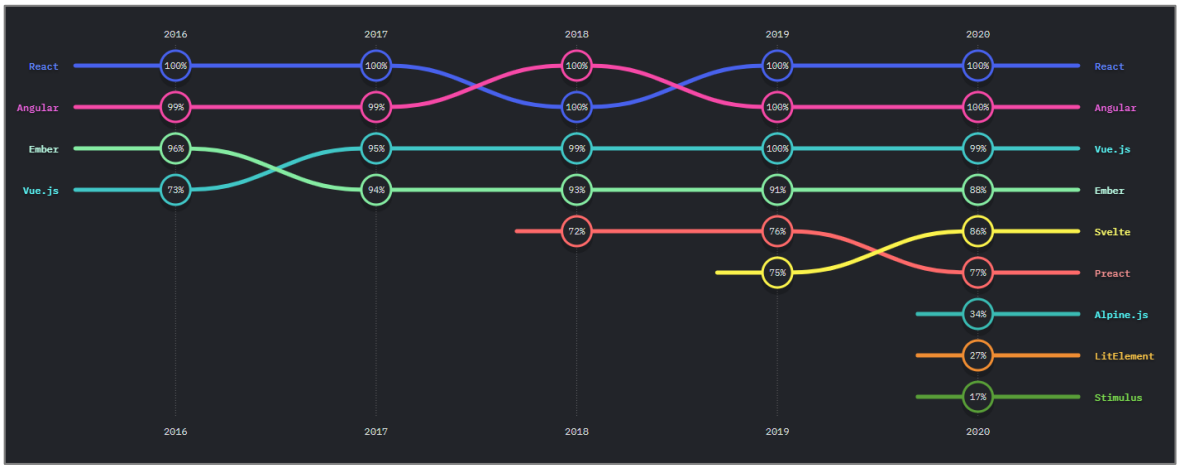
Příloha 18 – Míra spokojenosti vývojářů s frameworky (Greif, a další, 2020)



Příloha 19 – Míra zájmu vývojářů o frameworky (Greif, a další, 2020)



Příloha 20 – Míra užití frameworků vývojáři (Greif, a další, 2020)



Příloha 21 – Míra známosti frameworků vývojáři (Greif, a další, 2020)

8.4 Ukázky kódu

```
// ChangePassword.js
import { useState } from "react";
import Auth from "actions/auth";

export const ChangePassword = () => {
    const [password, setPassword] = useState("");

    const onSubmitHandler = () => {
        Auth.changePassword(password)
            .then(() => {})
            .catch(error => {});
    }

    return <div></div>;
}
```

Příloha 22 – Jednoduchá komponenta bez vykreslení v Reactu

```
<!-- ChangePassword.vue -->
<template>
  <div></div>
</template>

<script>
import Auth from 'actions/auth';

export default {
  name: 'change-password',
  data() {
    return {
      password: '',
    };
  },
  methods: {
    onSubmitHandler() {
      Auth.changePassword(this.state.password)
          .then(() => {})
          .catch(error => {});
    },
  },
}
</script>

<style>
</style>
```

Příloha 23 – Jednoduchá komponenta bez vykreslení (SFC) ve Vue

```
// ChangePassword.component.ts
import { Component } from '@angular/core';
import { Auth } from 'services/auth';

@Component({
  selector: 'change-password',
  templateUrl: './ChangePassword.component.html',
  styleUrls: ['./ChangePassword.component.scss'],
})
export class ChangePasswordComponent {
  password: string = '';

  constructor(
    private auth: Auth,
  ) {}

  onSubmitHandler() {
    this.auth.changePassword(this.password)
      .subscribe(() => {})
      .catch(error => {});
  }
}

// ChangePassword.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { ChangePasswordComponent } from './change-password.component';

@NgModule({
  imports: [CommonModule],
  declarations: [ChangePasswordComponent],
})
export class ChangePasswordModule {}
```

Příloha 24 – Jednoduchá komponenta s modulem bez vykreslení v Angularu

```
<input type="text" value={oneWayDatabindingData}/>

<SpecialButton
    size="big"
    disabled
    onClick={onClickHandler}>
    {caption}
</SpecialButton>
```

Příloha 25 – Vykreslování a data binding v Reactu

```
<input type="text" v-model="twoWayDatabinding" />

<special-button
    size="big"
    v-bind:disabled="true"
    v-on:click="onClickHandler"
>
  {{ caption }}
</special-button>
```

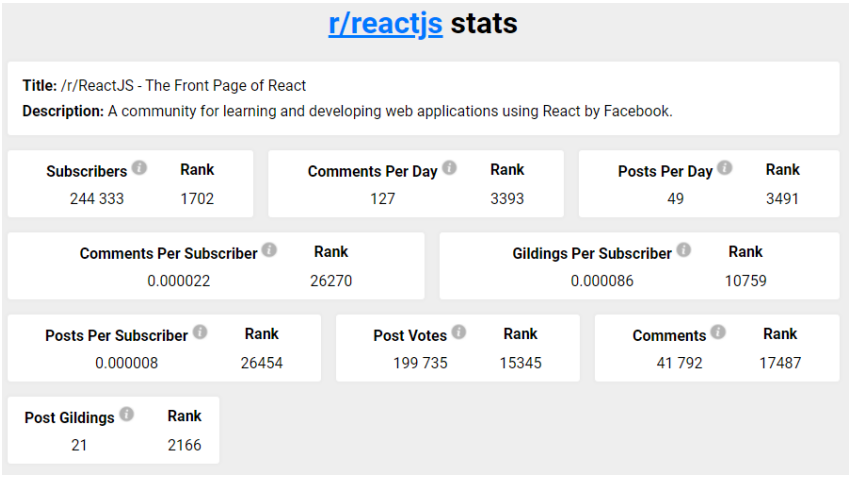
Příloha 26 – Vykreslování a data binding ve Vue

```
<input type="text" [(ngModel)]="twoWayDatabinding">

<special-button
    size="big"
    [disabled]="true"
    (click)="onClickHandler()">
    {{ caption }}
</special-button>
```

Příloha 27 – Vykreslování a data binding v Angularu

8.5 Ostatní přílohy



r/reactjs stats

Title: /r/ReactJS - The Front Page of React
Description: A community for learning and developing web applications using React by Facebook.

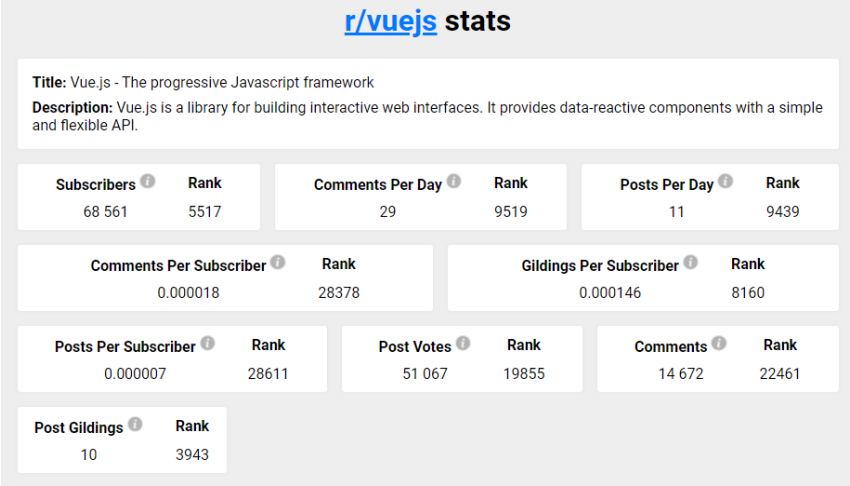
Subscribers	Rank	Comments Per Day	Rank	Posts Per Day	Rank
244 333	1702	127	3393	49	3491

Comments Per Subscriber	Rank	Gildings Per Subscriber	Rank
0.000022	26270	0.000086	10759

Posts Per Subscriber	Rank	Post Votes	Rank	Comments	Rank
0.000008	26454	199 735	15345	41 792	17487

Post Gildings	Rank
21	2166

Příloha 28 – Statistiky subredditu r/reactjs (Subreddit Stats, 2021)



r/vuejs stats

Title: Vue.js - The progressive Javascript framework
Description: Vue.js is a library for building interactive web interfaces. It provides data-reactive components with a simple and flexible API.

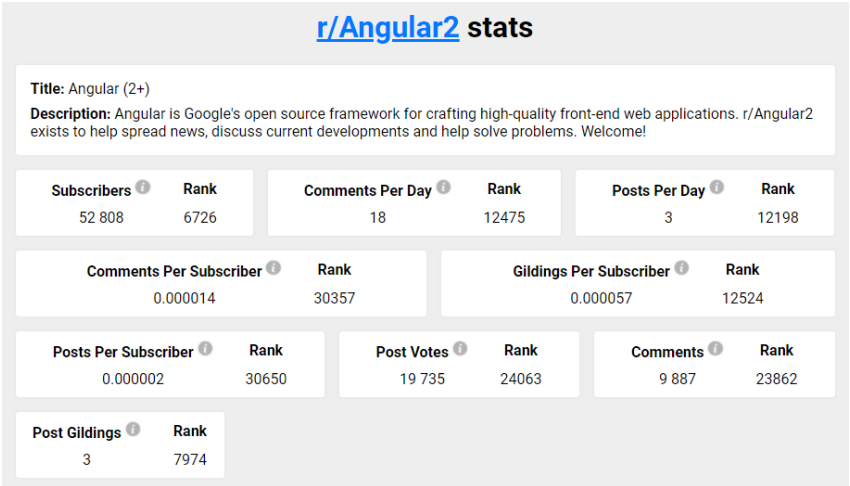
Subscribers	Rank	Comments Per Day	Rank	Posts Per Day	Rank
68 561	5517	29	9519	11	9439

Comments Per Subscriber	Rank	Gildings Per Subscriber	Rank
0.000018	28378	0.000146	8160

Posts Per Subscriber	Rank	Post Votes	Rank	Comments	Rank
0.000007	28611	51 067	19855	14 672	22461

Post Gildings	Rank
10	3943

Příloha 29 – Statistiky subredditu r/vuejs (Subreddit Stats, 2021)



r/Angular2 stats

Title: Angular (2+)
Description: Angular is Google's open source framework for crafting high-quality front-end web applications. r/Angular2 exists to help spread news, discuss current developments and help solve problems. Welcome!

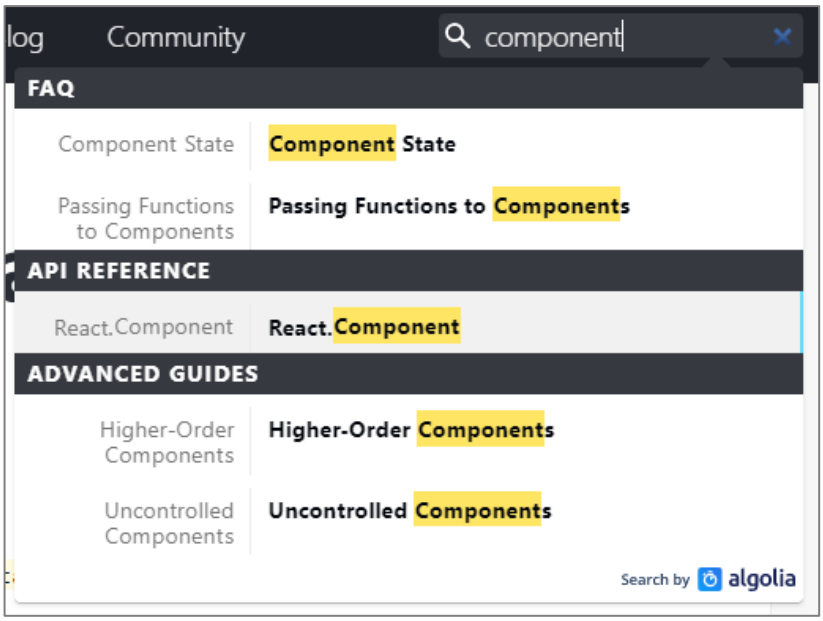
Subscribers	Rank	Comments Per Day	Rank	Posts Per Day	Rank
52 808	6726	18	12475	3	12198

Comments Per Subscriber	Rank	Gildings Per Subscriber	Rank
0.000014	30357	0.000057	12524

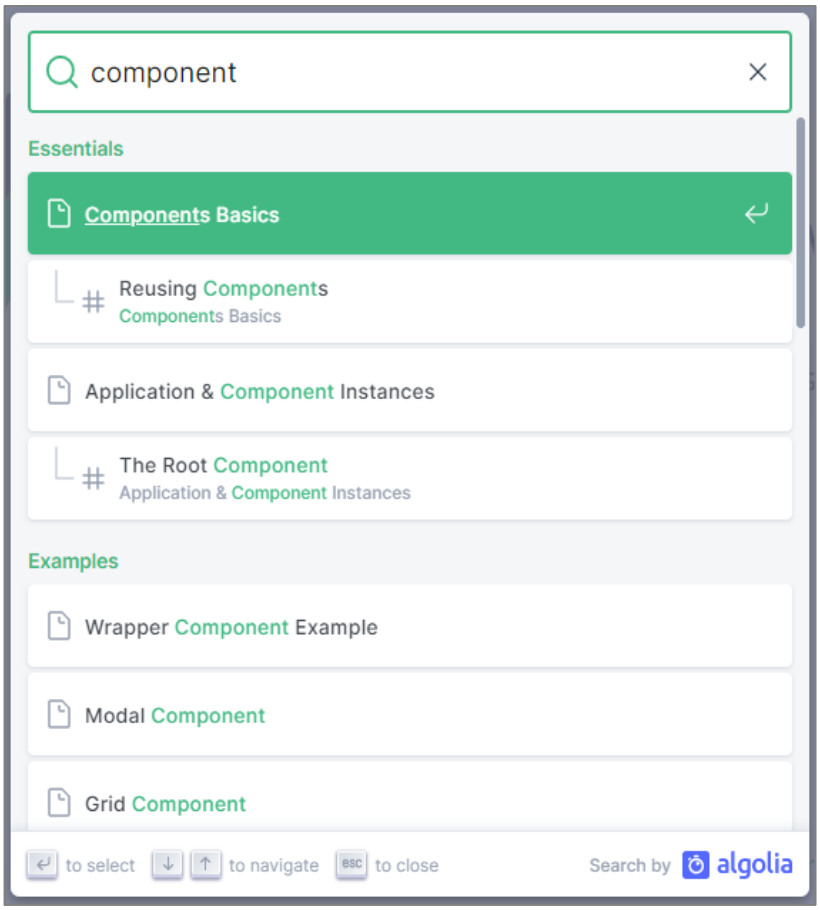
Posts Per Subscriber	Rank	Post Votes	Rank	Comments	Rank
0.000002	30650	19 735	24063	9 887	23862

Post Gildings	Rank
3	7974

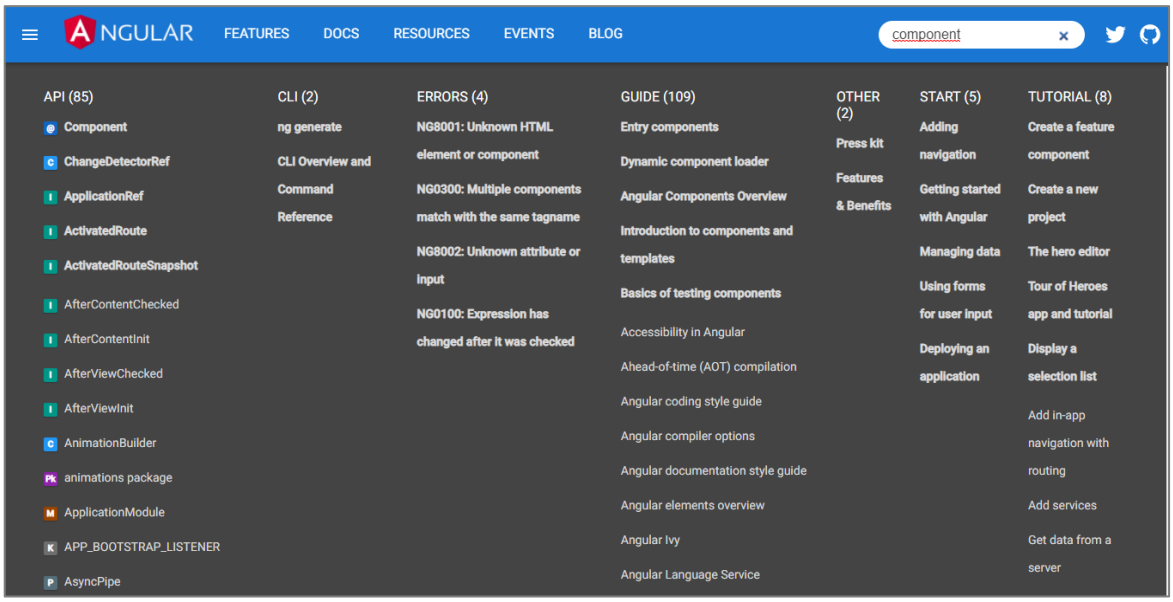
Příloha 30 – Statistiky subredditu r/angular2 (Subreddit Stats, 2021)



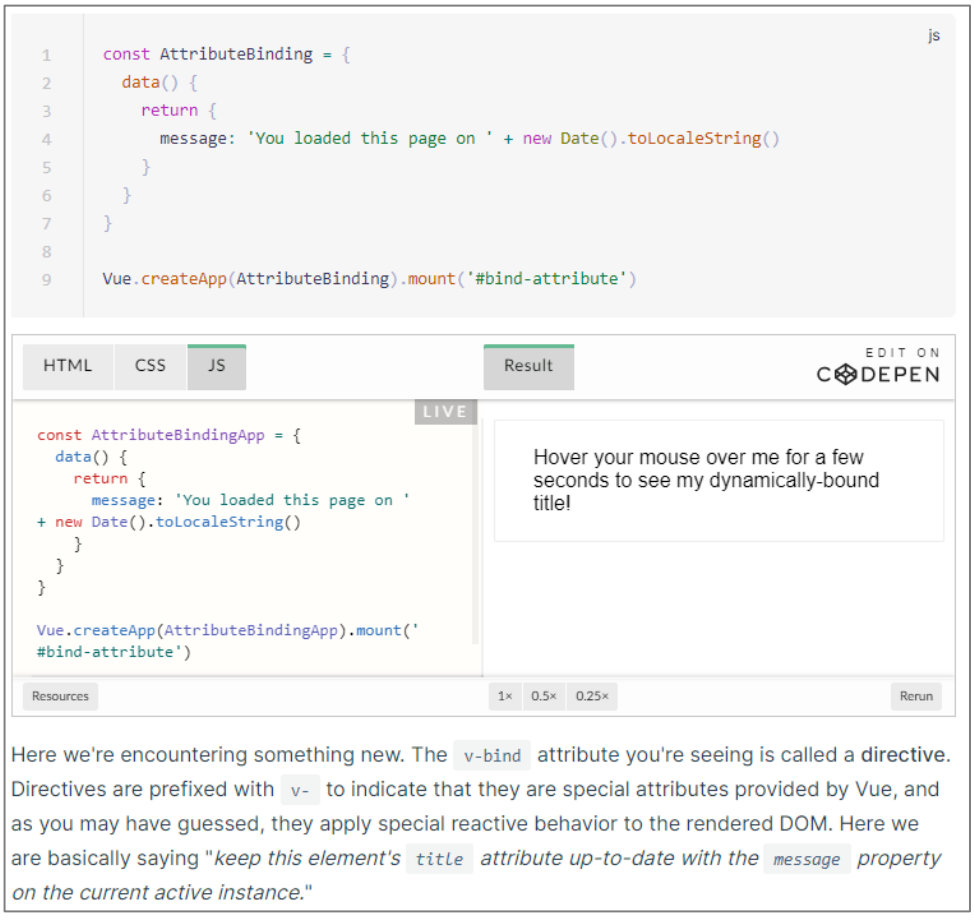
Příloha 31 – Vyhledávání na reactjs.org



Příloha 32 – Vyhledávání na v3.vuejs.org



Příloha 33 – Vyhledávání na angular.io



Příloha 34 – Interaktivní příklad ve Vue dokumentaci

Demo komunity - TodoMVC	React	Vue	Angular
Desktop	*React*	*Vue*	*Angular*
First Contentful Paint (15%) [s]	0,3	0,3	0,3
Time to Interactive (15%) [s]	0,3	0,3	0,3
Speed Index (25%) [s]	0,3	0,3	0,3
Total Blocking Time (15%) [ms]	0	0	0
Largest Contentful Paint (25%) [s]	0,3	0,3	0,3
Cumulactive Layout Shift (5%)	0	0	0
Ligthhouse 6 hodnocení	**100**	**100**	**100**
Mobil	*React*	*Vue*	*Angular*
First Contentful Paint (15%) [s]	1,2	1,1	1,4
Time to Interactive (15%) [s]	1,2	1,1	1,4
Speed Index (25%) [s]	1,2	1,1	1,4
Total Blocking Time (15%) [ms]	0	0	10
Largest Contentful Paint (25%) [s]	1,2	1,1	1,4
Cumulactive Layout Shift (5%)	0	0	0
Ligthhouse 6 hodnocení	**100**	**100**	**100**

Příloha 35 – Lighthouse měření TodoMVC

Demo komunity - Realworld (Conduit)	React	Vue	Angular
Desktop	*React/Redux*	*Vue3/Vite*	*Angular*
First Contentful Paint (15%) [s]	0,7	0,7	0,5
Time to Interactive (15%) [s]	0,7	0,7	0,9
Speed Index (25%) [s]	0,7	1,3	0,5
Total Blocking Time (15%) [ms]	0	0	10
Largest Contentful Paint (25%) [s]	1,4	1,1	0,8
Cumulactive Layout Shift (5%)	0,001	0	0,363
Ligthhouse 6 hodnocení	**96**	**96**	**96**
Mobil	*React/Redux*	*Vue3/Vite*	*Angular*
First Contentful Paint (15%) [s]	3,0	2,0	1,9
Time to Interactive (15%) [s]	4,2	2,2	5,1
Speed Index (25%) [s]	3,0	2,0	1,9
Total Blocking Time (15%) [ms]	180	40	350
Largest Contentful Paint (25%) [s]	3,6	2,8	3,8
Cumulactive Layout Shift (5%)	0,001	0,254	0,167
Ligthhouse 6 hodnocení	**83**	**92**	**79**

Příloha 36 – Lighthouse měření Realworld