Moderní vývoj webových aplikací a tvorba uživatelského rozhraní

Martin Půr

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin Půr**
Osobní číslo:	**A19784**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Moderní vývoj webových aplikací a tvorba uživatelského rozhraní**
Téma práce anglicky:	**Modern Web Application Development and User Interface Creation**

Zásady pro vypracování

1. Popište teoreticky základní principy a problematiku tvorby webového uživatelského prostředí.
2. Nastudujte a popište javascriptové frameworky Angular, React a Vue.
3. Pro každý z výše uvedených frameworků vyberte a popište rozšiřující knihovnu pro vývoj uživatelského prostředí.
4. Vytvořte demonstrační aplikaci v každém z vybraných frameworků a zaměřte se zejména na způsob tvorby uživatelského rozhraní pomocí rozšiřujících knihoven.
5. Porovnejte vývoj uživatelského prostředí mezi jednotlivými frameworky a zhodnoťte přínosy vybraných rozšiřujících knihoven.

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

Seznam doporučené literatury:

1. React. Reactjs [online]. USA: Facebook, 2013 [cit. 2021-9-27]. Dostupné z: https://reactjs.org/
2. Vue. Vuejs [online]. USA: Evan You, 2014 [cit. 2021-9-27]. Dostupné z: https://vuejs.org/
3. Angular. Angular [online]. USA: Google, 2016 [cit. 2021-9-27]. Dostupné z: https://angular.io/
4. Material UI. Material UI [online]. Francie: Material-UI SAS, 2014 [cit. 2021-9-27]. Dostupné z: https://mui.com/
5. Vuetify. Vuetify [online]. USA: John and Heather Leider, 2016 [cit. 2021-9-27]. Dostupné z: https://vuetifyjs.com/en/
6. BANKS, Alex a Eve PORCELLO. Learning React: functional web development with React and Redux. Sebastopol, CA: O'Reilly Media, 2017. ISBN 9781491954621.
7. MOHAMMED, Zama Khan. Angular Projects: build nine real-world applications from scratch using Angular 8 and TypeScript. Birmingham: Packt publishing, 2019. ISBN 1838559353.
8. MACRAE, Callum. Vue.js: up and running : building accessible and performant web apps. Sebastopol: O'Reilly, 2018. ISBN 1491997249.

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne 12.5.2022

Martin Půr v. r.
podpis studenta

ABSTRAKT

Cílem bakalářské práce je porovnat vývoj demonstrační aplikace ve frameworcích Angular, React a Vue za pomoci rozšiřujících knihoven pro tvorbu uživatelských rozhraní. Úvodem práce pojednává o problematice návrhu uživatelského rozhraní. Dále práce obsahuje představení jednotlivých frameworků a rozšiřujících knihoven. Následně se práce věnuje vývoji tří variant demonstrační aplikace, vždy se jedná o framework a jeho rozšiřující knihovnu. Poslední část se pak věnuje shrnutí vývoje všech aplikací a porovnání výhod použitých nástrojů.

Klíčová slova: Uživatelské rozhraní, Angular, Angular Material, React, Material UI, Vue, Vuetify

ABSTRACT

The aim of the bachelor thesis is to compare the development of a demonstration application in Angular, React and Vue frameworks with the help of extension libraries for creating user interfaces. The introduction deals with the issue of user interface design. Furthermore, this thesis contains an introduction to individual frameworks and extension libraries. Subsequently, this thesis deals with the development of three variants of the demonstration application, i tis always a framework and its extension library. The last part then summarizes the development of all applications and compares the benefits of the tools used.

Keywords: User interface, Angular, Angular Material, Reacr, Material UI, Vue, Vuetify

Tímto bych chtěl poděkovat vedoucímu bakalářské práce panu Ing. Radku Valovi Ph.D. za cenné rady, ochotu a odborné vedení mé práce.

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

OBSAH

ÚVOD

V dnešní době existuje mnoho nástrojů pro moderní vývoj webových aplikací a zejména pro začínajícího vývojáře je složité si vybrat nástroje pro tvorbu svých prvních webových aplikací. Proto se tato práce zabývá vývojem a porovnáním tří moderních frameworků a jejich rozšiřujících knihoven pro tvorbu webových uživatelských rozhraní. Tato práce vznikla za účelem zjednodušení výběru nástrojů pro vývoj webových aplikací a za účelem poukázání na vhodnost použití rozšiřujících knihoven při tvorbě uživatelského rozhraní.

V teoretické části nastíním problematiku uživatelského rozhraní. Dodržením důležitých principů vývoje uživatelského rozhraní lze u uživatele vytvořit zážitek, který je klíčový pro to, aby se návštěvník v aplikaci cítil příjemně a chtěl danou aplikaci použít i příště. Dále představím tři moderní frameworky Angular, React a Vue, které se podle [1] řadí mezi pět nejpoužívanějších nástrojů pro vývoj moderních webových aplikací, nejpoužívanějším frameworkem je React, který používá přibližně 40 % respondentů. Pro každý framework představím i jednu rozšiřující knihovnu, určenou pro tvorbu uživatelského rozhraní. Jedná se o knihovny Angular Material, Material UI a Vuetify. Při výběru těchto knihoven jsem se zaměřil na množství nabízených komponent a na přehlednost a srozumitelnost dokumentace.

V praktické části nejprve navrhnu demonstrační aplikaci a poté ji vytvořím ve třech variantách pomocí již zmíněných nástrojů. Důraz bude kladen na použití velkého počtu předpřipravených komponent a atributů rozšiřujících knihoven. Mnoho lidí dnes webové aplikace používá i na mobilních telefonech, tudíž bude kladen důraz i na responzivní design. Na závěr shrnu vývoje všech tří aplikací a porovnám hlavní výhody každé varianty použitých nástrojů, aby se nerozhodný vývojář mohl snadněji rozhodnout, jaké nástroje chce pro tvorbu vlastní webové aplikace použít.

I. TEORETICKÁ ČÁST

1 UŽIVATELSKÉ ROZHRANÍ

Uživatelské rozhraní je nástroj, pomocí kterého uživatel ovládá softwarovou aplikaci nebo hardwarové zařízení. Kvalitní uživatelské rozhraní je vytvořeno tak, aby uživatele práce s ním bavila a vytvářela v něm pocit příjemné a dobře odvedené práce. Bezmála všechny softwarové aplikace obsahují grafické uživatelské rozhraní (GUI). To znamená, že uživatel pomocí myši nebo klávesnice obsluhuje grafické ovládací prvky aplikace a vidí jejich projevující se chování v závislosti na uživateli nebo čase [2].

1.1 Webové uživatelské prostředí

Ve spojení s webovými aplikacemi se budeme bavit zejména o grafických či textových prvcích a jejich rozmístění. Tyto prvky mají více významů, od jednoduché práce a interakce s uživatelem přes získávaní užitečných vstupních dat po reagovaní na změny v aplikaci provedené uživatelem. Veškeré změny provedené uživatelem a reakce uživatelského rozhraní jsou nejčastěji prezentovány přes webový prohlížeč [3].

Základním znakem moderních grafických uživatelských rozhraní je oddělenost obsahu (výstupu aplikace) a jeho grafické interpretace. To má pozitivní vliv obzvláště při rozsáhlých úpravách aplikace, aktualizacích, ale i rovněž při tvorbě nových GUI aplikací pro další platformy (např. mobilní telefony). Při rozšíření aplikace není díky oddělenosti třeba měnit celou aplikaci, ale stačí pouze vytvořit další GUI, a to ke stávající aplikaci připojit, což pomáhá nejen k úspoře času, ale hlavně k úspoře financí [3].

1.2 Proč potřebujeme grafické uživatelské prostředí?

Grafické rozhraní podstatně ovlivňuje vnímání celé aplikace uživatelem. Kromě toho, že vytváří první dojem, tak dále stanovuje, jakým způsobem budou aplikaci uživatelé dále vnímat a jak s ní budou pracovat. To je důležité i proto, zda aplikaci bude uživatel používat v delším časovém horizontu nebo zda ji doporučí někomu ze svých přátel či okolí. Tedy zda s ní budou uživatelé spokojeni, jestli pro ně bude práce s aplikací zábavná, a hlavně jestli jim pomůže splnit jejich cíle [3].

Takovému zaměření na uživatele, a od něj vyplívajícímu se navrhování aplikace se říká *User Experience Design* (UXD) nebo také *User-Centered Design* (UCD). *User Experience Design* se snaží dosáhnout co nejlepšího spojení několika disciplín. Těmi jsou použitelnost,

design ve smyslu vizuálního řešení i informační architektury a komunikace uživatele s aplikací. Stará se tedy nejen o formu webové aplikace, ale i o její obsah a chování [3].

1.3 Fáze procesu tvorby

Při vývoji uživatelského rozhraní je důležité postupovat podle předem definovaného postupu, který ovšem v závislosti na naši aplikaci můžeme lehce poupravit. To znamená, že některým bodům věnujeme větší důležitost než jiným, neměli bychom však žádný opomenout, protože podcenění vývoje může mít negativní dopad na funkcionalitu aplikace [4].

Dle [3] obsahuje proces tvorby uživatelského rozhraní následující kroky:

- Analýza funkcionality systému
- Definování chování uživatele v systému
- Určení základní struktury systému
- Tvorba jednoduchého vzoru aplikace
- Návrh grafické stránky aplikace
- Testování funkcionalit a ověření uživatelské analýzy

1.4 Principy designu

Nejdůležitějším elementem, který bychom měli vzít v potaz při vývoji aplikace a jejího designu, je uživatel a jeho způsob používání naší aplikace. Proto bychom měli klást větší důraz na dodržení několika principů [4].

Dle [4] jsou těmito principy:

- Uspokojení potřeb a cílů uživatele v systému
- Přehlednost a pochopitelnost funkcí
- Reagování na změny v systému provedené uživatelem
- Atraktivní grafické rozhraní pro vytvoření uživatelského zážitku
- Zpětná vazba od uživatelů

Zmíněné principy lze shrnout tak, že navržená aplikace by měla mít jasnou a srozumitelnou strukturu, jednoduché ovládání, přívětivé vizuální uživatelské rozhraní, aby poskytla uživateli všechny důležité informace a odezvy na jeho chování ve správný čas a na správném místě. Rovněž by měla aplikace předpokládat, že uživatel není robot a někdy udělá chybu,

na kterou by měla být připravena a být schopna na ni reagovat. Jako správný softwarový produkt by měla být aplikace otevřena dalším případným úpravám a rozšířením [4].

2 JAVASCRIPTOVÉ FRAMEWORKY

Velmi užitečným nástrojem při tvorbě webových aplikací je tzv. framework. Framework je uskupení několika knihoven a zdrojového kódu, mezi jeho největší přednosti patří zobecnění určitých typů problémů, s nimiž se vývojáři mohou setkat a znovupoužitelnost zdrojového kódu. Tím významně sníží čas potřebný k vývoji aplikace [5].

Framework není vázán na jeden programovací jazyk a lze se s ním setkat kromě JavaScriptu například u jazyků C# (.NET framework) a PHP (Laravel framework). JavaScriptový framework je tedy framework usnadňující vývoj v JavaScriptu. JavaScriptové frameworky dnes používá více a více vývojářů, díky rostoucí komunitě roste i počet problémů, které tyto frameworky dokáží vyřešit. Efektivně strukturovaný kód a prvky, které by vývojář bez frameworku neměl k dispozici mu ušetří mnoho času [5].

Obzvláště u JavaScriptových frameworků je důležitá již zmíněná znovupoužitelnost, a to ve smyslu předpřipravených GUI komponent, které autoři frameworku nabízí dalším vývojářům pro urychlení práce. Pro začlenění komponenty do aplikace stačí často vložit pouze pár řádků kódu. Bez frameworku by bylo velmi složité a neefektivní tyto komponenty vytvořit. Rovněž umožňují JavaScriptové frameworky jednoduchý přístup k libovolnému elementu stránky, za účelem nastavení události nebo přidání animace [5].

2.1 JavaScript

JavaScript je objektově orientovaný programovací jazyk, využívaný při tvorbě webových stránek. První zmínka o jeho použití je z roku 1995, kdy se JavaScript objevil v prohlížeči *Netscape navigator 2.0.* JavaScript byl dříve používán pouze na straně klienta, kde pracoval spolu s HTML a kaskádovými styly, aby napomáhal vylepšovat zážitek uživatele na webové stránce [6].

Na straně klienta JavaScript kontroluje, jestli jsme správně vyplnili formulář a stará se o animace, události či dynamické změny v HTML dokumentu. Zhruba od roku 2010 se začal JavaScript dostávat čím dál tím více do popředí a stále více lidí o tento jazyk projevuje zájem. Díky tomu vznikly frameworky pro rychlejší webová rozhraní, jako je Angular, React nebo Vue. Ty umožňují používat JavaScript i na straně serveru pro generování obsahu HTML stránek [6].

2.2 Angular

Angular je javascriptový designový framework a vývojová platforma pro vytváření efektivních a sofistikovaných jednostránkových aplikací s přehlednými uživatelskými rozhraními [7].

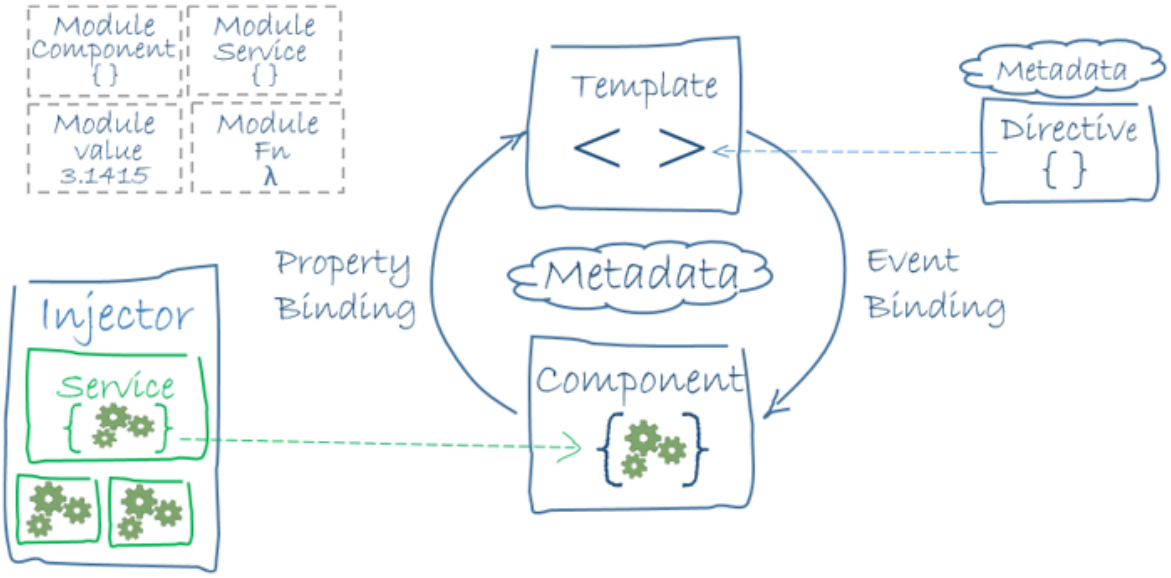
Webové aplikace se v dnešní době více a více podobají nativním desktopovým aplikacím, kvůli zvyšující se složitosti se stávají populárními tzv. jednostránkové aplikace. Ty se dnes používají, za účelem vytvoření zkušeností na straně klienta, které zvyšují zážitek uživatele při práci s aplikací v závislosti na rychlosti a odezvě. Jednostránkové aplikace mají tu výhodu, že se jejich celý obsah načte do webového prohlížeče pouze poprvé a při další interakci s uživatelem se načítají pouze další potřebná data bez znovunačtení celého obsahu stránky. V minulosti se totiž webové stránky, které byly vykreslovány na straně serveru, při každé interakci musely nově celé načíst [8].

2.2.1 Architektura

Architektura Angular aplikace je založena na svých základních konceptech. Jádrem Angularu jsou moduly *NgModules*, které se skládají z komponent a tvoří sady funkčně souvisejících kódů. Každá Angular aplikace obsahuje vždy alespoň jeden (kořenový) modul, který umožňuje spuštění aplikace. V průběhu vývoje se zejména u složitějších aplikací počet modulů obvykle zvětšuje [9].

Komponenty určují to, co uživatel vidí v prohlížeči a definují pohledy (*views*). To jsou sady prvků obrazovky, s nimiž Angular pracuje a upravuje je na základě logiky a dat programu. Komponenty spolupracují se službami (*services*), které jim přidávají další funkcionalitu. Tato funkcionalita není přímo spojena s komponentou, ale může být do ní přidána jako závislost. Tím se stává kód efektivnějším a opakovaně použitelným [9].

K tomu, aby uživatel věděl, jak moduly, komponenty a služby správně používat slouží tzv. dekorátory. Dekorátory označují typ třídy a metadata, která potřebuje Angular pro práci s moduly, komponenty nebo službami. V případě metadat pro třídu komponent je daná komponenta přiřazena k šabloně (*template*) a ta definuje pohled. V šabloně je kombinován klasický HTML kód s direktivami Angularu a vazebnými značkami, tím je umožněno Angularu upravit HTML před vykreslením na obrazovku. Metadata pro třídu služeb jsou důležitá pro to, aby Angular mohl vkládat závislost služby do komponenty [9].



Obrázek 1. Angular diagram [9]

2.2.2 Instalace

Pro vygenerování nového projektu se často používá balíček Angular CLI. K jeho instalaci je ovšem třeba technologie Node.js, která podporuje i mnoho dalších nástrojů pro usnadnění práce nebo přidání nové funkcionality.

2.2.2.1 Node.js

Pro instalaci Node.js je nutné si stáhnout instalační soubor pro požívanou počítačovou platformu pomocí odkazu na stránce https://nodejs.org/en/download/. Po dokončení stahování instalačního souboru je třeba jej spustit a proklikat instalační proceduru do konce bez nutnosti změny některého atributu. Správnost instalace lze ověřit v příkazové řádce následujícím příkazem: `node -v`. Pakliže proběhla instalace v pořádku, tak se zobrazí číslo nainstalované verze Node.js.

2.2.2.2 Angular CLI

Angular CLI (*Command Line Interface*) je oficiální nástroj, který je určen pro tvorbu a práci s Angular projekty. Je používán, aby vývojář nemusel zbytečně ztrácet čas s konfiguracemi různých nástrojů jako jsou TypeScript nebo Webpack [10]. Vývojáři stačí zadat příkaz pro vytvoření projektu a Angular CLI následně provede všechny nezbytné konfigurace za něj. Pro instalaci tohoto balíčku slouží příkaz: `npm install -g @angular/cli`.

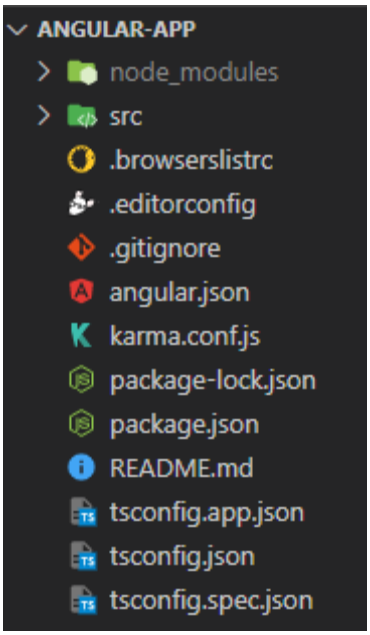
```
> npm install -g @angular/cli
> ng new angular-app
> cd angular-app
> ng serve
```

Obrázek 2. Angular CLI instalace a vytvoření projektu

Na obrázku 2 lze vidět postup, který popisuje cyklus od nainstalování Angular CLI přes vytvoření nového projektu až po jeho spuštění. Příkaz `cd angular-app` znamená přesun do nově vytvořeného projektu, kde jej můžeme spustit.

2.2.3 Struktura základního projektu

Po vytvoření nového projektu se v jeho složce objeví řada souborů a složek, tomuto uspořádání se říká struktura adresáře.



Obrázek 3. Adresářová struktura Angularu

Na obrázku 3 lze vidět základní adresářovou strukturu v nově vygenerovaném projektu frameworku Angular. Každý soubor nebo adresář má svůj význam.

- node_modules – Zde se nachází všechny knihovny třetích stran, které jsou nainstalovány pomocí příkazu: `npm install` [10].
- src – Obsahuje zdrojový kód aplikace, většina práce se dělá zde. Je zde umístěna složka app, ve které jsou moduly a komponenty [10].
- .browserslistrc – Důležitý soubor, který podporuje CSS [10].

- .editorconfig – Pomáhá udržovat konzistentnost na projektu, na kterém pracuje více vývojářů, kteří preferují různé kódovací styly nebo pracují v různých editorech [11].
- .gitignore – Soubory a složky obsažené v tomto textovém souboru budou ignorovány Gitem, díky čemuž se zamezí odkládání velkých souborů na GitHub [12].
- angular.json – Zde je konfigurace pro CLI, například konfiguraci příkazů *serve* a *test* [10].
- karma.conf.js – Obsahuje konfiguraci pro testovací nástroj Karma, například jsou zde uvedeny frameworky, které budou provádět testování nebo port, na němž se bude testovat [10].
- package-lock.json a package.json – Obsahují základní informace o projektu (název, popis a závislosti), v souboru package-lock.json jsou navíc závislosti rozepsány podrobněji [10].
- README.md – Soubor, který obsahuje podrobný popis projektu na GitHubu a tím pomáhá ostatním vývojářům se snadněji připojit ke tvorbě dané aplikace [13].
- soubory tsconfig – Konfigurační soubory pro TypeScript, v němž bývá psána logika služeb, nebo komponent [10].

2.2.4 Vestavěné direktivy

Vestavěné direktivy jsou třídy, které přidávají nebo upřesňují chování prvků stránky a rovněž napomáhají k lepší orientaci v kódu. Angular nabízí mnoho vestavěných direktiv, které se používají ke správě formulářů, seznamů a celkově toho, co uživatel vidí [14].

2.2.4.1 Direktivy atributů

Mezi jedny z nejrozšířenějších direktiv patří tzv. direktivy atributů, tyto direktivy pracují s vizuální stránkou a daty prvků HTML stránky. Podle [14] mezi nejčastěji používané direktivy atributů patří:

- NgClass – Upravuje kaskádové styly.
- NgStyle – Upravuje styly v šabloně HTML.
- NgModel – Slouží jako obousměrná datová vazba mezi formulářovým polem a proměnnou, která toto pole obsluhuje.

```
<div [ngClass]="isSpecial ? 'special' : ''">This div is special</div>
```

Obrázek 4. Použití direktivy NgClass [14]

Pokud má atribut *isSpecial* pravdivou hodnotu, tak se prvku přiřadí třída kaskádových stylů *special*.

2.2.4.2 Strukturální direktivy

Strukturální direktivy řeší převážně rozložení prvků HTML dokumentu. Upravují nebo mění strukturu DOM, většinou přidáváním, odebíráním a manipulací s hostitelskými prvky, k nimž jsou připojeny. U dlouhých seznamů významně zkracují množství kódu, které je potřebné k zobrazení všech položek seznamu. Mezi nejvíce používané strukturální směrnice dle [14] patří:

- NgIf – V závislosti na dané podmínce zobrazuje nebo skrývá dílčí část šablony.
- NgFor – Pro každý prvek z kolekce provede svůj vnitřní kód.
- NgSwitch – Direktiva umožňující přepínaní mezi několika pohledy v závislosti na proměnné.

```
<div *ngFor="let item of items">{{item.name}}</div>
```

Obrázek 5. Použití direktivy NgFor [14]

Pro každý uzel (*item*) ze seznamu (*items*) se vytvoří element obsahující název uzlu (*item.name*).

2.2.5 Interpolace

Pro odkazování na data objektů se v Angularu používají dvojité složené závorky, které jsou známé pod pojmem interpolace. Princip interpolace je prostý, interpolace se podívá na výraz mezi složenými závorkami a zjistí, která komponenta nebo modul daný výraz podporuje. Poté vykreslí výsledek jako řetězec uvnitř HTML dokumentu [8].

2.3 React

React nebo ReactJS je veřejně dostupná javascriptová knihovna. Jelikož se stará pouze o vrstvu zobrazení, tak je používána k tvorbě uživatelských rozhraní pro webové aplikace. Hlavním stavebním kamenem je komponenta, což je malý opakovaně použitelný a izolovaný kus kódu. Spojením několika různých komponent se dají vytvořit i složitější uživatelská rozhraní. Komponenty, které tvoří vizuální stránku aplikace jsou první důležitou částí Reactu, to druhou je pak HTML dokument, kde jsou komponenty vykreslovány [15].

ReactJS je sice javascriptová knihovna, ale jelikož se tato práce ve své praktické části zaměřuje na uživatelské rozhraní, tak nám bude plně dostačovat tento nástroj a nebudeme používat javascriptový framework React Native.

2.3.1 React Native

Plnohodnotným a rovněž veřejně dostupným javascriptovým frameworkem je React Native. React Native se používá pro vývoj aplikací na platformy iOS, Android a Windows, cílí tedy spíše na mobilní zařízení než na webový prohlížeč. Sice React Native také používá komponenty, na rozdíl od ReactJS, ale používá nativní komponenty, a ne ty webové [15].

2.3.2 Component Based Architecture

Představením Reactu společností Facebook v roce 2013 se změnil způsob tvorby uživatelských rozhraní. React přišel s novým konceptem zvaným *Component Based Architecture* [16].

Component Based Architecture je architektura, která si klade za cíl rozložení návrhu uživatelského rozhraní do menších komponent a tím vzniká vyšší úroveň abstrakce. Komponenta totiž představuje dobře komunikačně vybavenou logickou jednotku, která může obsahovat metody, události a vlastnosti. Při vývoji tedy nevzniká mnoho rozsáhlých problémů, ale objevují se spíše menší problémy spojené s určitou komponentou [17].

Nejdůležitější vlastností u komponent je v případě této architektury opětovná použitelnost, protože mnohdy potřebujeme použít již vytvořenou komponentu na jiném místě pouze s lehce pozměněnými vlastnostmi či daty. Mezi největší výhody *Component Based* architektury patří: snadný vývoj, snížené náklady, opakovaná použitelnost, spolehlivost a nezávislost [17].

2.3.3 Co je to komponenta?

Komponenta je modulární, vyměnitelný a opakovaně použitelný softwarový objekt, který může obsahovat sadu funkcí, které rozšiřují jeho chování. Komponenta může v rámci aplikace spolupracovat s jinými komponentami, obsahovat několik menších komponent nebo být součástí větší komponenty, za účelem kombinace různých funkcionalit. Všechny komponenty v rámci architektury mají specifikované rozhraní a doporučené chování, které by měly splňovat [17].

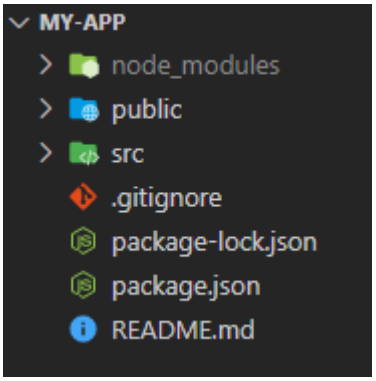
2.3.4 Instalace

Pro instalaci Reactu slouží příkaz *create-react-app*. Ten automaticky nainstaluje veškeré potřebné balíčky a provede nutnou konfiguraci. Po dokončení konfigurace je aplikace připravena k vývoji. Pro použití příkazu *create-react-app* je třeba stejně jako např. u Angularu mít nainstalovaný Node.js, viz 2.3.2.1.

Pro vytvoření nové React aplikace slouží příkaz: `npx create-react-app my-app`. Po dokončení instalace je vhodné přejít do adresáře, do něhož byla aplikace nainstalována, pomocí příkazu `cd my-app`. Poté už zbývá pouze spustit nově vytvořenou aplikaci příkazem: `npm start`.

2.3.5 Struktura základního projektu

Ve složce nově vytvořeného projektu byla vygenerována struktura adresáře, obsahující základní potřebné soubory a adresáře. Význam některých souborů byl již vysvětlen viz kapitola 2.2.3, proto se zaměříme na ty, které se od Angularu liší nebo u něj nejsou.



Obrázek 6. Adresářová struktura ReactJS

- public – Kořenová složka aplikace, obsahuje soubor index.html, což je soubor, jehož obsah vidíme, když spustíme aplikaci příkazem `npm start` [18].
- src – Složka obsahující soubory, se kterými vývojář nejčastěji pracuje, to znamená komponenty, testy, CSS soubory a jiné [18].

2.3.6 JavaScriptové výrazy

JavaScriptové výrazy jsou javascriptové proměnné, které jsou po obou stranách obklopeny složenou závorkou. JavaScriptové výrazy se v Reactu používají v tu chvíli, když je potřeba předat komponentám hodnotu nějaké proměnné. Pro zobrazení hodnoty libovolné proměnné

v HTML prvku nebo komponentě je nutné proměnnou vložit na požadované místo v šabloně a z obou stran přidat složenou závorku. Proměnná se při startu aplikace vyhodnotí a její výsledek bude vrácen [19].

```
<h1>{this.props.name}</h1>
```

Obrázek 7. JavaScriptový výraz v Reactu

2.4 Vue

Vue je progresivní javascriptový framework pro tvorbu uživatelských rozhraní. Vue je navržen tak, aby se mohl snadno integrovat do jiných knihoven nebo projektů, protože jeho základní knihovna se stará jen o vrstvu zobrazení. V kombinaci s dalšími knihovnami a nástroji dokáže vytvořit důmyslné jednostránkové aplikace. I Vue využívá principu komponent, takže také dokáže znatelně zrychlit a zefektivnit proces vývoje uživatelských rozhraní [20] [21].

2.4.1 Instalace

K instalaci Vue aplikace je třeba stejně jako např. u Angularu mít nainstalovaný nástroj Node.js, viz 2.3.2.1. Po instalaci Node.js je už samotná instalace Vue jednoduchá.

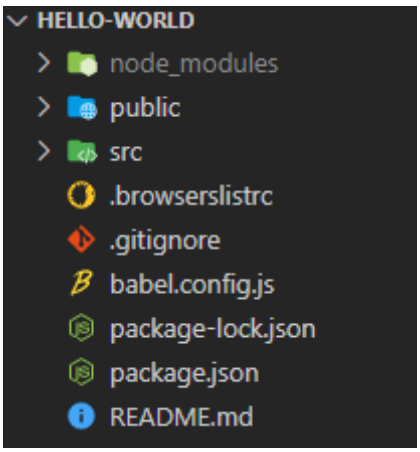
I Vue má svůj nástroj pro vytváření projektů, tímto nástrojem je balíček Vue CLI, který lze nainstalovat příkazem: `npm install -g @vue/cli`. Po úspěšně provedené instalaci lze vytvořit nový projekt. Pro vytvoření projektu jsou určeny dva příkazy: `vue create hello-world` a `vue ui`.

Zvolením první možnosti budeme v příkazové řádce vyzváni k výběru předvolby projektu. Můžeme si vybrat z výchozího základního nastavení nebo z varianty *Manually select features*, což znamená ruční výběr funkcí, které potřebujeme pro svůj projekt.

Druhá varianta pro vytvoření projektu nabízí grafické rozhraní, kde si nejprve v horním menu zvolíme záložku *Create*, následně klikneme na *Create a new project here*, poté napíšeme jméno projektu a stejně jako u první varianty zvolíme předvolbu nástrojů. Nově vytvořený projekt spustíme v jeho složce příkazem: `npm run serve`.

2.4.2 Struktura základního projektu

Stejně jako u předchozích frameworků se i u Vue vygeneruje po instalaci základní adresářová struktura. Význam některých souborů a adresářů byl vysvětlen u předchozích frameworků viz kapitoly 2.2.3 a 2.3.4.



Obrázek 8. Adresářová struktura Vue

- src – Složka obsahující komponenty, soubor main.js, kde je vytvořena instance Vue a jiné soubory [22].
- babel.config.js – Soubor konfigurace celého projektu, který spolupracuje například i se složkou knihoven *node_modules* [23].

2.4.3 Direktivy

Vue se hodně soustřeďuje na způsob zobrazení dat v prohlížeči. Vue používá šablony, v nichž je klasické HTML doplněno speciálními vlastnostmi, kterým se říká direktivy. Ty se používají především k podmíněnému vykreslování prvků stránky a přehlednější manipulaci s daty [24].

```
<div id="app">
  <p v-if="isMorning">Good morning!</p>
  <p v-if="isAfternoon">Good afternoon!</p>
  <p v-if="isEvening">Good evening!</p>
</div>

<script>
  var hours = new Date().getHours();
  new vue({
    el: '#app',
    data: {
      isMorning: hours < 12,
      isAfternoon: hours >= 12 && hours < 18,
      isEvening: hours >= 18
    }
  });
</script>
```

Obrázek 9. Vue direktiva v-if [24]

Pomocí objektu *data* se předávají Vue informace o datech, kterém chceme zobrazit v šabloně. Na obrázku 9 lze vidět definici proměnných: *isMorning*, *isAfternoon* a *isEvening*, z těchto definic lze usoudit, že vždy bude mít pravdivou hodnotu pouze jedna z proměnných [24].

Na obrázku 9 je zobrazen příklad direktivy *v-if*, která se zapisuje jako vlastnost HTML tagu nebo komponenty následovaná rovnítkem a podmínkou. V příkladu jsou sice uvedeny tři prvky s direktivou *v-if*, nicméně už víme, že v libovolném čase bude jen jedna podmínka pravdivá, tudíž se ve webovém prohlížeči zobrazí vždy jen jeden pozdrav [24].

2.4.4 Interpolace

Dalším možným zpřehledněním kódu je interpolace, pomocí které lze předávat data šabloně pro vykreslení jejich hodnot. Ve Vue používáme pro interpolaci dvojité složené závorky [24].

3 NÁSTROJE PRO TVORBU UŽIVATELSKÉHO ROZHRANÍ

V dnešní době se na vývoji uživatelských prostředí nepoužívají pouze frameworky. Kvůli většímu důrazu na uživatelův zážitek se dnes vyvíjejí podpůrné nástroje nejen pro lepší prožitek uživatele, ale i pro zefektivnění práce vývojáře. Hlavně u javascriptových frameworků jsou takové nástroje v podobě rozšiřujících knihoven čím dál tím více populární.

V následujících částech této kapitoly bude stručně představena jedna rozšiřující knihovna pro tvorbu uživatelského rozhraní pro každý již představený javascriptový framework.

3.1 Angular Material

Angular Material (AM) je rozšiřující knihovna Angularu pro tvorbu uživatelských rozhraní. AM nabízí vysoce kvalitní a spolehlivé komponenty a atributy, které umožňují dosáhnout pěkného rozhraní za relativně málo času.

3.1.1 Instalace

Předpokládejme, že už máme vytvořený Angular projekt a nyní do něj chceme nainstalovat AM, toho docílíme příkazem: `ng add @angular/material`. Během instalace budeme postupně vyzváni k volbě motivu, vybrat si můžeme z několika předpřipravených variant nebo si můžeme vytvořit motiv svůj. Poté budeme dotázáni, zda chceme přidat globální typografii stylů, a nakonec jestli chceme do projektu přidat animace pro prohlížeč.

3.1.2 Komponenty

Jak již bylo zmíněno AM obsahuje velké množství snadno a opakovaně použitelných komponent. Pro použití jakékoliv komponenty je nutné v hlavičce souboru *app.module.ts* importovat modul odpovídající dané komponentě a přidat jej ve stejném souboru do sekce *@NgModule/imports*. Název modulu je pro každou komponentu uveden v oficiální dokumentaci AM ve složce API.

3.1.3 Ikony

Kromě široké řady komponent nabízí AM i sadu ikon. Tyto ikony se do HTML dokumentu zapisují pomocí tagu *mat-icon*, pro použití tohoto tagu musíme stejně jako u komponent importovat odpovídající modul (*MatIconModule*). Seznam ikon přehledně rozdělený do

několika kategorií je uveden na stránce: https://www.angularjswiki.com/angular/angular-material-icons-list-mat-icon-list/.

3.2 Material UI

Material UI nebo zkráceně MUI je rozšiřující knihovna Reactu, která nabízí mnoho základních i složitějších komponent a tím pomáhá vývojářům zrychlit vývojový proces aplikace. Co se týče knihoven pro tvorbu uživatelských rozhraní ve frameworku React, tak MUI patří k těm nejpopulárnějším.

3.2.1 Instalace

K instalaci MUI a uložení potřebných závislostí v již vytvořené React aplikaci slouží příkaz: `npm install @mui/material @emotion/react @emotion/styled`.

3.2.2 Komponenty

MUI se také pyšní velkým počtem předpřipravených komponent. Každá komponenta má rezervovaný svůj vlastní tag, abychom tag komponenty mohli použít, tak jej musíme importovat v hlavičce souboru z *@mui/material*. Například pro komponentu *Button* by import vypadal následovně: `import Button from '@mui/material/Button'`.

3.2.3 Ikony

Společnost Google vytvořila více než 1900 veřejně dostupných ikon. MUI nabízí balíček, ve kterém je každá z více než 1900 ikon zpracována jako snadno použitelná komponenta. K instalaci tohoto balíčku je určen příkaz: `npm install @mui/icons-material`. Pro použití libovolné ikony je třeba ji importovat z *@mui/icons-material*. Kdybychom tedy chtěli importovat ikonu domečku (*Home* ikonu), tak bychom to udělali následovně: `import HomeIcon from '@mui/icons-material/Home'` [25].

3.3 Vuetify

Vuetify je rozšiřující knihovna pro tvorbu uživatelských rozhraní frameworku Vue. Vuetify má za cíl dosažení co nejlepšího uživatelského zážitku. Snaží se také vývojářům poskytovat co nejširší sadu komponent. Vuetify splňuje specifikaci *Material Design*, což je designový systém umožňující tvořit velmi kvalitní aplikace se zaměřením na zkušenosti uživatele, a to na mnoha různých platformách (Android, iOS nebo web) [26] [27].

3.3.1 Instalace

Vuetify se instaluje do již existujícího projektu pomocí příkazu: `vue add vuetify`, anebo pomocí příkazu: `vue ui`, který zobrazí grafické rozhraní, pomocí něhož lze v záložce *plugins* Vuetify nainstalovat.

3.3.2 Komponenty

Vuetify obsahuje širokou řadu jednoduše použitelných komponent. Nabízí i takové komponety jako *Carousels*, *Parallax* nebo *Steppers*. Výhodou je, že před použitím komponenty od Vuetify je nemusíme nikde importovat. V dokumentaci jsou obsáhlejší komponenty jako například lišty, formulářové prvky nebo tabulky rozděleny do několika menších článků.

3.3.3 Ikony

Ani ikony se nemusí ve Vuetify importovat, pro jejich použití stačí do tagu *v-icon* napsat „mdi-NazevIkony“. Zkratka MDI znamená *Material Design Icons*, což jsou ikony, které Vuetify nabízí.

II. PRAKTICKÁ ČÁST

4 DEMONSTRAČNÍ APLIKACE

V této části se zaměřím na návrh a tvorbu jednoduché vzorové aplikace. Nejprve navrhnu základní funkcionalitu a strukturu aplikace a poté ji pomocí představených javascriptových frameworků a jejich rozšiřujících knihoven implementuji.

4.1 Návrh demonstrační aplikace

Hlavním cílem vzorové aplikace je demonstrace tvorby a použití nejčastějších prvků uživatelského rozhraní, které je shodně implementováno pomocí všech tří vybraných UI knihoven. Cílem aplikace není dodat uživatelskou aplikaci ve smyslu hotového produktu, ale spíše prototyp vhodný ke srovnání. Proto byla vybrána jednoduchá, ale zásadní funkcionalita formulářových vstupů a zpracování dat, která je popsána níže.

Návrh vzorové aplikace by se měl řídit doporučenými fázemi vývoje grafického rozhraní. Proto se na každou fázi zaměřím a popíši její stav vzhledem k demonstrační aplikaci. Jedinou výjimkou je fáze testování, na kterou se již tato práce nezaměřuje.

4.1.1 Analýza funkcionality systému

Hlavní funkcionalitou systému bude obsluha dat formuláře, která do něj uživatel vloží a odešle. Vedlejší funkcionalitou bude možnost smazání vložených dat do formuláře a upozornění uživatele, když nezadá důležité údaje (jméno, příjmení a email).

4.1.2 Definování chování uživatele v systému

Uživatel bude v aplikaci vyplňovat formulář, mazat vyplněná pole a odesílat zvolená data. Odesílat vyplněný formulář bude uživatel moci pouze pokud zadal alespoň všechny požadované informace ve správném formátu.

4.1.3 Určení základní struktury systému

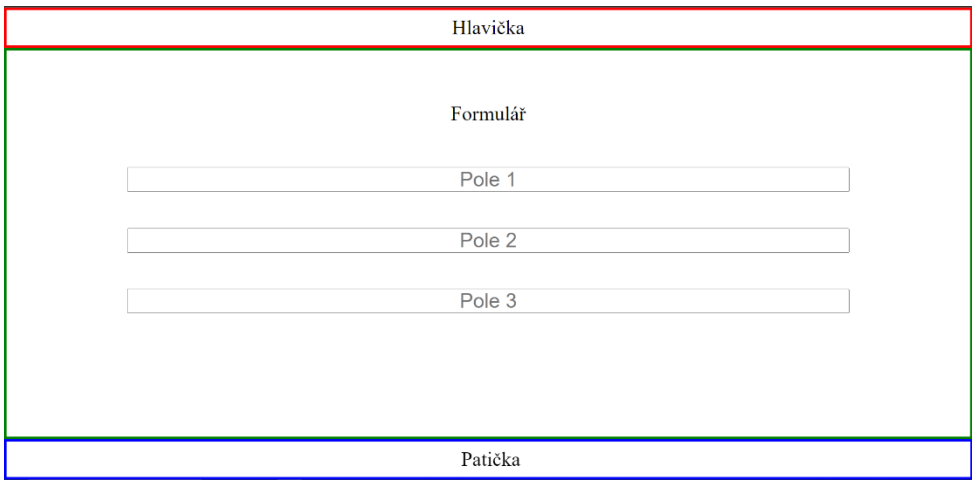
Aplikace se bude skládat z pěti hlavních částí: hlavička, patička, formulář, mobilní menu a plocha pro zobrazení odeslaných dat uživatele.

Formulář se bude skládat ze 7 polí různých typů například: textové pole (*text field*), výběrové pole (*select*) nebo přepínač (*radio buttun*). Hlavička bude obsahovat důležité ikony, informace a vyhledávací pole. Na zařízeních s malým displejem bude pod hlavičkou k dispozici možnost zobrazit mobilní menu. Patička bude obsahovat možnosti kontaktů v podobě

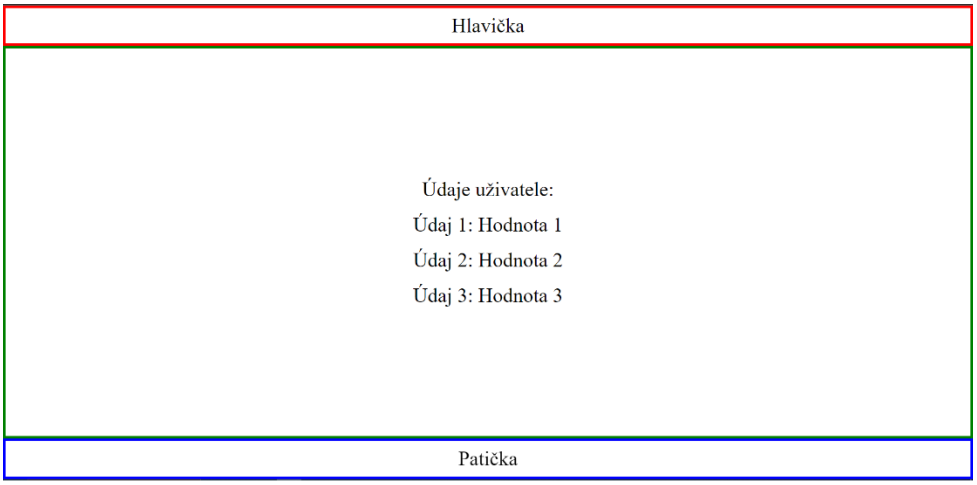
sociálních sítí. Plocha pro zobrazení odeslaných dat uživatele bude zobrazovat výpis informací o uživateli.

4.1.4 Tvorba jednoduchého vzoru aplikace a návrh grafické stránky aplikace

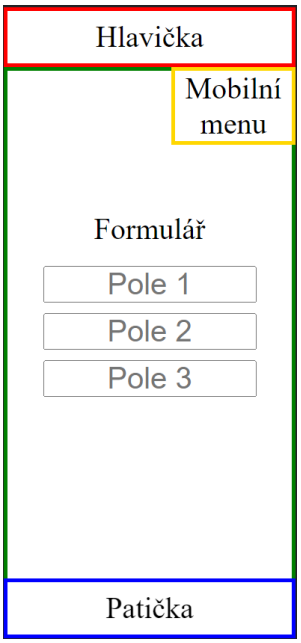
V tomto kroku jsem vytvořil několik základních zobrazení představujících strukturu a uspořádání jednotlivých částí aplikace. Ve vzorové aplikaci bude docházet ke dvěma důležitým změnám v zobrazení. Za prvé, v aplikaci bude zobrazen buď formulář pro vyplnění, nebo stránka s už odeslanými daty. Za druhé, pakliže bude uživatel používat malý displej, tak bude mít možnost si zobrazit mobilní menu s notifikacemi, které by se do hlavičky nevešly. Tyto stavy jsem zaznamenal do následujících obrázků.



Obrázek 10. Struktura aplikace s formulářem



Obrázek 11. Struktura aplikace s daty uživatele



Obrázek 12. Struktura aplikace s otevřeným mobilním menu

4.2 Vývoj demonstrační aplikace pomocí Angularu a Angular Material

K vývoji demonstračních aplikací používám vývojové prostředí *Visual Studio Code*.

Aplikaci jsem dříve rozdělil do 5 základních částí viz 4.1.3, pro každou z těchto částí jsem vytvořil komponentu příkazem `ng generate component components/nazev`. Všechny vytvořené komponenty je třeba v souboru *src/app/app.module.ts* importovat (v hlavičce) a deklarovat (v sekci *@NgModule/declarations*).

Komponenta v Angularu je rozdělena do několika souborů. Mezi nejdůležitější patří soubor s příponou *html*, kde je uložena šablona aplikace, dále soubor s příponou *ts,* v němž se nachází funkční logika komponenty a soubor s příponou *css*, kde se zapisují kaskádové styly.

Každá komponenta používá některé moduly knihovny AM, které je taky nutné do souboru *src/app/app.module.ts* importovat (v hlavičce a v sekci *@NgModule/imports*), rovněž zde importuji i *RouterModule*. Ten bude zajišťovat navigaci v aplikaci. Dalším důležitým modulem je *FontAwesomeModule*, ten je třeba nejdříve nainstalovat příkazem `ng add @fortawesome/fontawesome-svg-core` a následně jej stejným způsobem importovat. Tento modul umožňuje použití ikon, které nenabízí AM.

Veškeré vizuální upravování tagů a komponent je provedeno pomocí přidávání a upravování tříd kaskádových stylů.

```
4   import { AppComponent } from './app.component';
5
6   import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
7   import { MatIconModule } from '@angular/material/icon';
8   import { MatButtonModule } from '@angular/material/button';
9   import { MatToolbarModule } from '@angular/material/toolbar';
10  import { MatBadgeModule } from '@angular/material/badge';
11  import { MatInputModule } from '@angular/material/input';
12  import { MatCardModule } from '@angular/material/card';
13  import { FontAwesomeModule } from '@fortawesome/angular-fontawesome';
14  import { MatRadioModule } from '@angular/material/radio';
15  import { MatSelectModule } from '@angular/material/select';
16  import { MatCheckboxModule } from '@angular/material/checkbox';
17  import { MatListModule } from '@angular/material/list';
18  import { FormsModule } from '@angular/forms';
19  import { RouterModule } from '@angular/router';
20
21  import { HeaderComponent } from './components/header/header.component';
22  import { MobileMenuComponent } from './components/mobile-menu/mobile-menu.component';
23  import { FormComponent } from './components/form/form.component';
24  import { FormResultsComponent } from './components/form-results/form-results.component';
25  import { FooterComponent } from './components/footer/footer.component';
```

Obrázek 13. Importování potřebných komponent a modulů v souboru app.module.ts

```
28  @NgModule({
29    declarations: [
30      AppComponent,
31      HeaderComponent,
32      MobileMenuComponent,
33      FormComponent,
34      FormResultsComponent,
35      FooterComponent,
36    ],
37    imports: [
38      BrowserModule,
39      BrowserAnimationsModule,
40      MatIconModule,
41      MatButtonModule,
42      MatToolbarModule,
43      MatBadgeModule,
44      MatCardModule,
45      MatInputModule,
46      FontAwesomeModule,
47      MatRadioModule,
48      MatSelectModule,
49      MatCheckboxModule,
50      MatListModule,
51      FormsModule,
52      RouterModule.forRoot([
53        {
54          path: '', component: FormComponent
55        },
56        {
57          path: 'results', component: FormResultsComponent
58        }
59      ])
60    ],
```

Obrázek 14. Deklarace potřebných komponent a modulů v souboru app.module.ts

4.2.1 Hlavička

Šablona hlavičky se skládá z 3 hlavních částí. Na levé stravě se nachází název použité rozšiřující knihovny. Uprostřed je vyhledávací pole, které ovšem nemá funkční uplatnění, slouží jen jako vizuální prvek. A na pravé straně se nachází ikony s notifikacemi.

Obrázek 15. Hlavička aplikace v Angularu

Jako zastřešující tag hlavičky jsem vybral v dokumentaci AM komponentu *mat-toolbar*. Veškeré ikony jsou obaleny tagem *button* s parametrem *mat-icon-button*, což z nich dělá tlačítka AM. Protože AM nemá svoji vlastní vyhledávací komponentu, tak je zde vyhledávací pole tvořeno tagem *input* a ikonou lupy. Počet notifikací u ikon vyjadřuje atribut *matBadge*, barvu notifikací lze změnit atributem *matBadgeColor*, já jsem použil již předdefinovanou hodnotu *warn*.

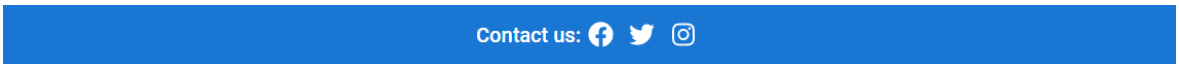
4.2.2 Patička

Šablona patičky obsahuje 3 odkazy na sociální sítě v podobě ikon. Sada ikon od AM neobsahuje ikony na některé sociální sítě, kvůli tomu jsem zde použil ikony *Font Awesome*, které se zapisují to HTML tagem *fa-icon* s parametrem *icon*, který rozhoduje o vzhledu ikony. Pro použití těchto ikon je třeba je importovat a uložit jejich hodnoty do proměnných v souboru s příponou *ts*. Tyto proměnné se předávají do šablony v parametru *icon*.

```
1  import { Component } from '@angular/core';
2  import { faFacebook } from '@fortawesome/free-brands-svg-icons';
3  import { faInstagram } from '@fortawesome/free-brands-svg-icons';
4  import { faTwitter } from '@fortawesome/free-brands-svg-icons';
5
6  @Component({
7    selector: 'app-footer',
8    templateUrl: './footer.component.html',
9    styleUrls: ['./footer.component.css']
10 })
11 export class FooterComponent {
12
13   constructor() { }
14
15   title = 'angular-app';
16   faFacebook = faFacebook;
17   faInstagram = faInstagram;
18   faTwitter = faTwitter;
```

Obrázek 16. Font Awesome ikony

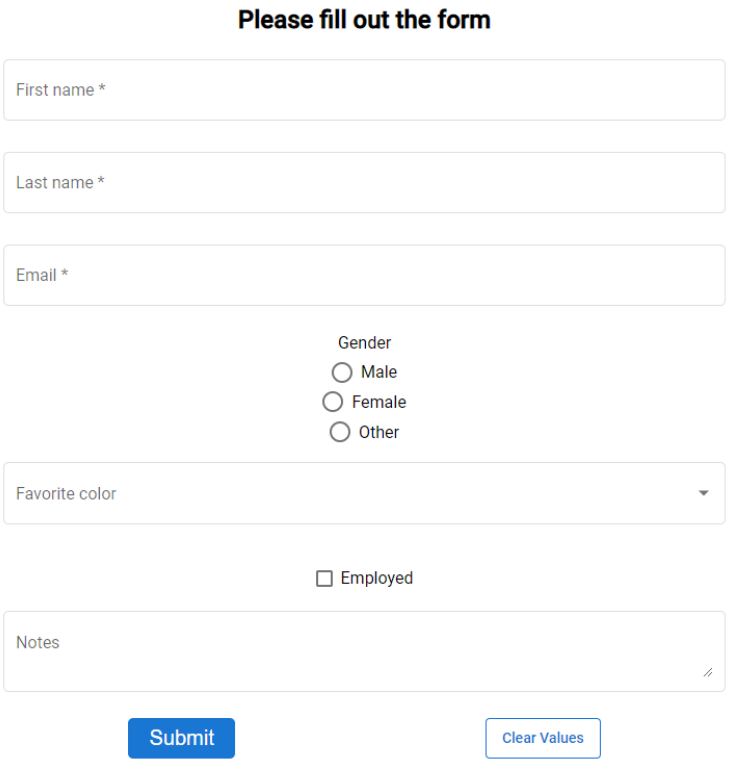
Pro ikonu Facebooku tag vypadá následovně `<fa-icon [icon]="faFacebook">`.



Obrázek 17. Patička aplikace v Angularu

4.2.3 Formulář

Formulář se skládá ze 7 polí (jméno, příjmení, email, pohlaví, oblíbená barva, stav zaměstnání a poznámky), které bude uživatel vyplňovat. Jedná se o 4 textová pole, z čehož jedno pole (pro poznámky) má výšku pro dva řádky, ostatní mají výšku jeden řádek. Dále jsou součástí formuláře výběrové pole (*select*), přepínací pole (*radio button*), zaškrtávací pole (*checkbox*) a dvě tlačítka pro odeslání dat a vyčištění formuláře. Pro tyto formulářové prvky má AM již předdefinované komponenty, které mají na začátku tagu zkratku *mat*, budu tedy například používat tagy *mat-form-field* a *mat-select*.



Obrázek 18. Formulář aplikace v Angularu

Pro snížení množství kódu v šabloně jsem definoval v souboru s příponou *ts* hodnoty variant (*options*) výběrového pole, jako seznam barev s hodnotou *value*. V šabloně lze poté vypsat všechny varianty výběrového pole pomocí direktivy **ngFor*, viz obrázek 19.

```
<mat-select name="favouriteColor">
  <mat-option *ngFor="let color of colors" [value]="color.value">
    {{color.value}}
  </mat-option>
</mat-select>
```

Obrázek 19. Direktiva Angularu ngFor

4.2.3.1 Odesílání a validace dat formuláře

Pro odeslání dat vyplněného formuláře je třeba provést několik kroků. Nejprve je třeba definovat cestu, kam se data budou posílat. Tato definice cest (*Routes*) se provádí v souboru *app.module.ts* v části *@NgModule/imports*, každá cesta v seznamu cest má 2 povinné parametry: cestu (*path*) a komponentu (*component*), která se bude na dané cestě zobrazovat viz obrázek 14.

Aby aplikace reagovala na změnu cesty, tak je třeba do šablony aplikace (*app.component.html*) přidat tag *router-outlet*, který se stará o vykreslování měnící se části aplikace. V mém případě se bude v aplikaci měnit formulář na plochu s odeslanými daty uživatele a opačně. Hlavička a patička budou viditelné vždy, proto budou také přímo uvedeny v šabloně aplikace.

```
1  <app-header></app-header>
2  <router-outlet></router-outlet>
3  <app-footer></app-footer>
```

Obrázek 20. Šablona aplikace s použitím routeru

Další kroky se týkají samotné komponenty formuláře. Prvním z nich je importování a přidání instance *Routeru* do konstruktoru v souboru *form.component.ts*. Ve stejném souboru si definuji výchozí stav nevyplněného formuláře pomocí objektu s názvem *result*, který obsahuje atributy jednotlivých políček formuláře.

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

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent {

  constructor(
    private router: Router,
  ) {
  }

  public result = {
    firstName: "",
    lastName: "",
    email: "",
    gender: "",
    favouriteColor: "",
    employed: false,
    notes: "",
  };

  colors = [
    {value: 'Red'},
    {value: 'Green'},
    {value: 'Blue'}
  ];
```

Obrázek 21. Definice proměnných formuláře v souboru form.component.ts

Dalším krokem je v šabloně u tagů všech vstupních polí přidání atributu *[(ngModel)]*, který je třeba položit rovno proměnné, pro kterou je pole určeno. Dále je třeba nastavit každému vstupnímu tagu ID jako hodnotu *ngModel* a v rámci jména, příjmení a emailu nastavit tato pole jako požadovaná (*required*), u emailu navíc i atribut email, který nedovolí odeslání formuláře, pokud uživatel nezadá platný email.

```html
<mat-form-field appearance="outline">
  <mat-label>Email</mat-label>
  <input
    matInput
    name="email"
    placeholder="Email"
    [(ngModel)]="result.email"
    #email="ngModel"
    required
    email
  >
  <mat-error [hidden]="email.valid">Email is required</mat-error>
</mat-form-field>
```

Obrázek 22. Vstupní pole emailu s potřebnými atributy

Tag *mat-error* bude ukazovat upozornění o špatném vyplnění pole, pakliže se tak stane.

Teď je třeba v šabloně přidat tagu formuláře atribut *ngSubmit*, kterému přiřadíme zatím neimplementovanou metodu *onSubmit* s třemi vstupními parametry, které budou hlídat, aby uživatel nemohl odeslat data formuláře, když nejsou vyplněná všechna požadovaná pole ve správném tvaru.

```
<form (ngSubmit)="onSubmit(firstName.valid, lastName.valid, email.valid)">
```

Obrázek 23. Tag formuláře s přiřazením atributu ngSubmit

Nyní už zbývá jen implementace metody *onSubmit*, která zkontroluje, jestli jsou požadovaná pole v pořádku a jestli jsou, tak všechna data formuláře pošle pomocí *Routeru* na komponentu s výsledky.

```
onSubmit(firstNameValid, lastNameValid, emailValid): void {
  if(firstNameValid && lastNameValid && emailValid) {
    this.router.navigateByUrl("/results", { state: this.result });
  }
}
```

Obrázek 24. Metoda onSubmit

4.2.3.2 Vyčištění formuláře

Metoda pro vyčištění formuláře je jednoduchá, stačí vrátit hodnoty všech vstupních polí formuláře to výchozího stavu.

```
onClear() {
  this.result.firstName = "";
  this.result.lastName = "";
  this.result.email = "";
  this.result.gender = "";
  this.result.favouriteColor = "";
  this.result.employed = false;
  this.result.notes = "";
}
```

Obrázek 25. Metoda onClear

4.2.4 Plocha pro zobrazení odeslaných dat

Když lze odesílat data formuláře, tak je vhodné je i někde prezentovat. Pro to slouží komponenta *form-results*, tato komponenta obsahuje pouze list s odeslanými údaji, které jsou v šabloně uvedeny pomocí interpolace, a odkaz (*routerLink*) pro návrat na formulář. U této komponenty je důležité hlavně to, že implementuje metodu *ngOnInit*, to znamená, že se

její kód provede ve chvíli, kdy Angular dokončí vytvoření komponenty. V metodě *ngOnInit* pouze přiřazuji proměnné *result* předaná data z formuláře pomocí atributu *history*.

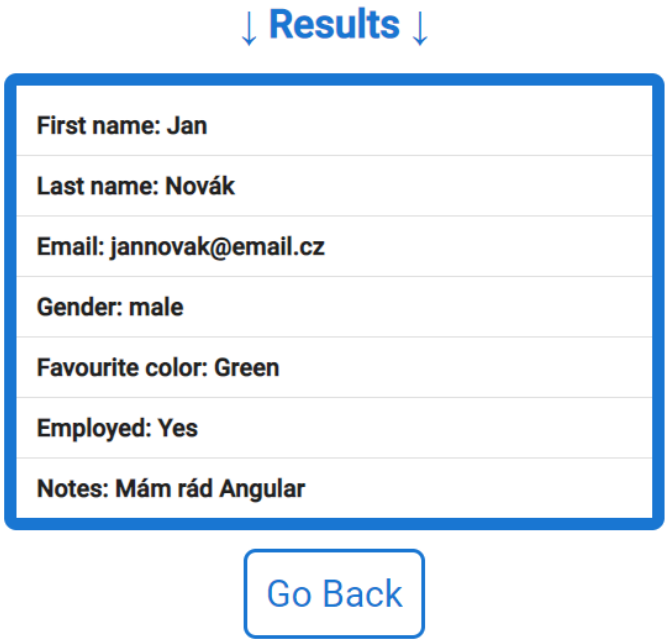
```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-form-results',
5    templateUrl: './form-results.component.html',
6    styleUrls: ['./form-results.component.css']
7  })
8  export class FormResultsComponent implements OnInit {
9
10   private result;
11
12   constructor() {
13   }
14
15   ngOnInit() {
16     this.result = history.state;
17   }
18
19   getFirstName() { return this.result.firstName; }
20   getLastName() { return this.result.lastName; }
21   getEmail() { return this.result.email; }
22   getGender() { return this.result.gender; }
23   getFavouriteColor() { return this.result.favouriteColor; }
24   getEmployed() { return this.result.employed; }
25   getNotes() { return this.result.notes; }
26
27 }
```

Obrázek 26. Zpracování dat z formuláře v souboru form-results.component.ts

Jelikož je proměnná *result* soukromá (*private*), tak k ní nelze přímo přistoupit v šabloně, proto jsem vytvořil pro každou dílčí proměnnou formuláře metodu začínající slovem *get*. Tyto metody slouží pro přístup k hodnotám daných proměnných v šabloně.

```
<mat-list class="list">
  <mat-list-item class="list-item-text">First name: {{ getFirstName() }}</mat-list-item>
  <mat-divider></mat-divider>
  <mat-list-item class="list-item-text">Last name: {{ getLastName() }}</mat-list-item>
  <mat-divider></mat-divider>
  <mat-list-item class="list-item-text">Email: {{ getEmail() }}</mat-list-item>
  <mat-divider></mat-divider>
  <mat-list-item class="list-item-text">Gender: {{ getGender() }}</mat-list-item>
  <mat-divider></mat-divider>
  <mat-list-item class="list-item-text">Favourite color: {{ getFavouriteColor() }}</mat-list-item>
  <mat-divider></mat-divider>
  <mat-list-item class="list-item-text">Employed: {{ getEmployed() ? "Yes" : "No"}}</mat-list-item>
  <mat-divider></mat-divider>
  <mat-list-item class="list-item-text">Notes: {{ getNotes() }}</mat-list-item>
</mat-list>
```

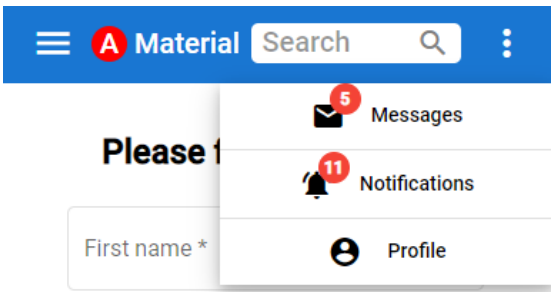
Obrázek 27. Přístup k datům formuláře v souboru form-results.component.html



Obrázek 28. Plocha pro zobrazení odeslaných dat

4.2.5 Mobilní menu

Na malých displejích by se nevlezly do hlavičky všechny potřebné dílčí komponenty, takže na těchto malých displejích je v pravém horním rohu zobrazena ikona tří teček, kterou lze otevřít mobilní menu s notifikacemi, které se na malý displej nevešly. Šablona se skládá ze tří tlačítkových ikon od AM, které jsou od sebe oddělené pomocí čáry *divider*.



Obrázek 29. Mobilní menu

4.2.5.1 Servis mobilního menu

Aby bylo možné otevírat a zavírat mobilní menu uvnitř různých komponent, tak je vhodné vytvořit servis (*service)*, který bude řídit viditelnost mobilního menu. Servis se vytvoří příkazem `ng generate service services/mobile-menu`. Pro řízení viditelnosti bude sloužit proměnná typu *Subject<boolean>*. Tato proměnná se nastaví jako *observable*, díky čemuž lze proměnnou aktualizovat ve funkci *emitChange*.

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class MobileMenuService {
  constructor() { }

  private mobileMenuVisibility = new Subject<boolean>();
  changeEmitted$ = this.mobileMenuVisibility.asObservable();
  emitChange(change: boolean) {
    this.mobileMenuVisibility.next(change);
  }
}
```

Obrázek 30. Servis mobilního menu

4.2.5.2 Implementace servisu do komponent

Pro implementaci servisu do komponent je třeba importovat servis, následně jej přidat do konstruktoru a vytvořit funkci pro volání metody servisu *emitChange*.

```
import { Component } from '@angular/core';
import { MobileMenuService } from 'src/app/services/mobile-menu.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent {

  constructor(private mobileMenuService: MobileMenuService) {}

  changeMobileMenuVisibility() {
    this.mobileMenuService.emitChange(true);
  }
}
```

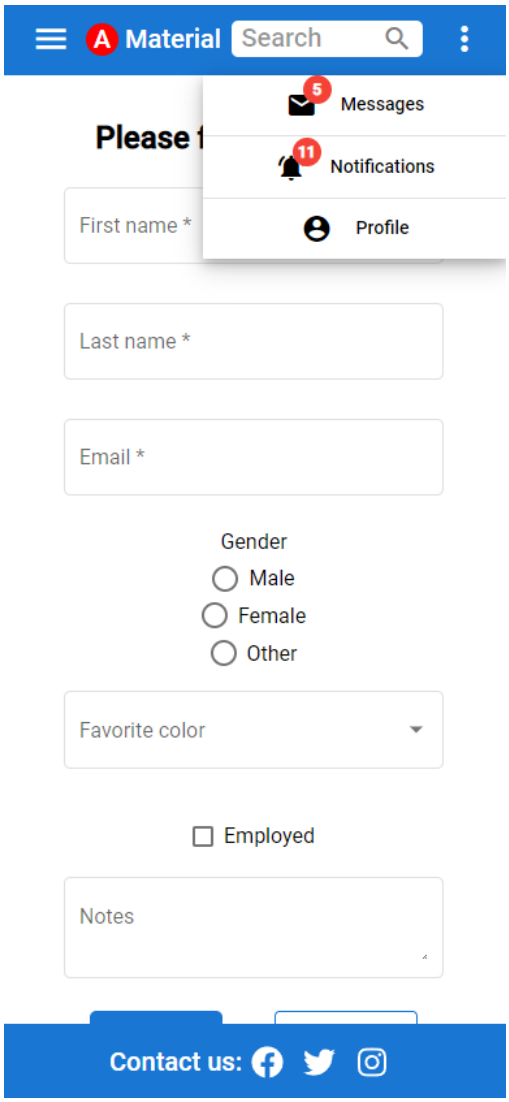
Obrázek 31. Implementace servisu mobilního menu

Hlavička je jediná komponenta, v níž lze otevírat mobilní menu, proto je v parametru funkce *emitChange* pravdivá hodnota, v ostatních komponentách je uvedena nepravdivá hodnota.

```
<button mat-icon-button class="more-btn" (click)="changeMobileMenuVisibility()">
  <mat-icon class="header-icon" >more_vert</mat-icon>
</button>
```

Obrázek 32. Přiřazení události pro otevření mobilního menu

Nyní už jsou všechny části demonstrační aplikace v Angularu hotové.



Obrázek 33. Aplikace vyvíjená pomocí Angularu a Angular Material

4.3 Vývoj demonstrační aplikace v Reactu a Material UI

Stejně jako u Angularu i v této variantě demonstrační aplikace jsem nejprve vytvořil komponentu pro každou základní část aplikace. Vytvoření komponent jsem provedl ručně ve vývojovém editoru vytvořením složky *components* ve složce *src*. Do složky *components* jsem následně umístil soubory komponent s příponou *jsx*.

Tento typ souboru se v moderních aplikacích používá při tvorbě uživatelského rozhraní. Jeho hlavními výhodami je možnost přiřazení HTML tagu javascriptové proměnné a zobrazování srozumitelnějších chybových hlášek [28].

V rámci přípravy stačí pouze nainstalovat *React Router* příkazem `npm install react-router-dom@6`. Pro usnadnění vývoje aplikací v Reactu je velmi výhodné nainstalovat (při použití prostředí *Visual Studio Code*) rozšíření *Simple React Snippets*, které usnadňuje tvorbu kódové struktury komponenty pomocí zkratek. V Reactu lze použít 2 základní typy komponent, a to třídní (*class*) a funkční (*function*). Za použití tohoto rozšíření lze vytvořit třídní komponentu pomocí zkratky *cc* a funkční komponentu pomocí zkratky *ffc*. Moderní aplikace používají spíše funkční komponenty, protože dovolují použití více React pomůcek (*React Hooks*), proto je použiji i já.

Obecně se komponenta v Reactu skládá z hlavičky s importy, šablony, která je obklopena klíčovým slovem *return*, a skriptové části, která se nachází v těle komponenty nad nebo pod šablonou.

Při vývoji je dále třeba myslet na to, že při použití předdefinovaných MUI komponent a ikon je nutné tyto komponenty importovat z knihovny *@mui/material* a ikony importovat z *@material-ui/icons*, viz kapitola 3.2.

V rámci vizuálních úprav MUI komponent většinou používám atribut *sx*, který dovoluje rychle a jednoduše stylovat MUI komponenty.

4.3.1 Hlavička

Struktura hlavičky je velmi podobná jako u aplikace v Angularu. V levé části je název rozšiřující knihovny Reactu, uprostřed se nachází vyhledávací pole, které je tvořeno komponentami *Paper*, *InputBase*, *IconButton* a *SearchIcon*. Na pravé straně jsou pak tlačítkové ikony s motivem různých notifikací.

Pro počet notifikací má MUI vlastní komponentu *Badge*, v níž pomocí atributu *badgeContent* lze nastavit hodnotu notifikací. Jednotlivé části hlavičky jsou obaleny komponentou *Box*. Všechny základní části jsou ještě obaleny komponentami *AppBar* a *ToolBar*.

Obrázek 34. Hlavička aplikace v Reactu

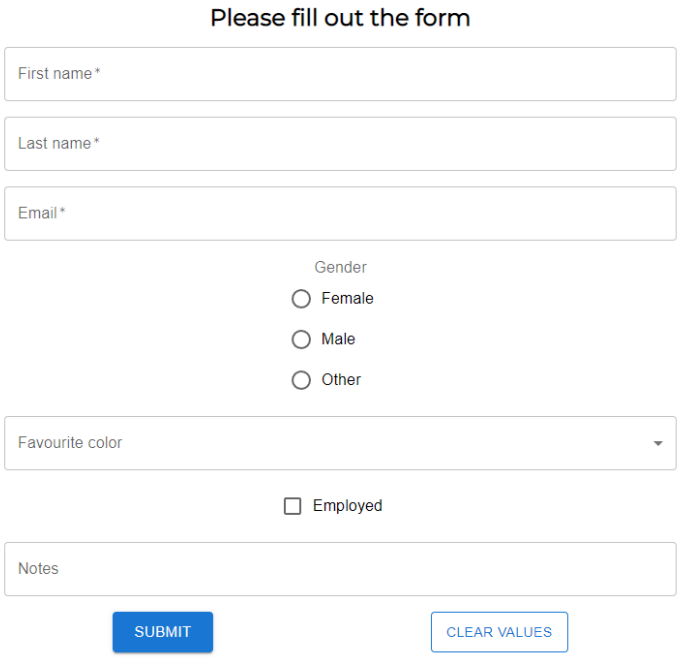
4.3.2 Patička

Patička se zde rovněž skládá ze 3 odkazů na sociální sítě. MUI má pro patičku předpřipravenou komponentu *BottomNavigation*, do níž jsem umístil *Box* s příslušnými ikonami.

Obrázek 35. Patička aplikace v Reactu

4.3.3 Formulář

Formulář je tvořen stejnými typy polí jako v demonstrační aplikaci Angularu (textové pole, přepínací pole, výběrové pole, zaškrtávací pole). Stejně jako u hlavičky jsou jednotlivé části formuláře obaleny komponentou *Box*. Přepínací pole jsou v MUI reprezentována komponentou *FormControlLabel*, kde pomocí atributu *control* lze upřesnit typ formulářového prvku, v tomto případě *Radio*. Stejně se zapisují i zaškrtávací pole, jen s rozdílem, že v atributu *control* bude hodnota *CheckBox*. Další mírnou změnou je zápis variant výběrového pole, který se provádí tagem *MenuItem*, namísto klasického tagu *option* v HTML.



Obrázek 36. Formulář aplikace v Reactu

4.3.3.1 Odesílání a validace dat formuláře

Při zpracování vstupu od uživatele je výhodné použít React pomůcku *useState*, pomocí které lze vytvořit proměnnou, přiřadit jí počáteční hodnotu a získat metodu pro změnu stavu vytvořené proměnné. Pomocí *useState* jsem vytvořil proměnné s jejich metodami pro všechny formulářové pole, a navíc i pro proměnnou, která detekuje, zda jsou data připravena k odeslání. Při odesílání dat je třeba data přesměrovat na jinou komponentu s výsledky, proto je

nutné použít pomůcku *useNavigate*. Obě zmíněné pomůcky je třeba v hlavičce komponenty importovat.

```
import React, { useState } from 'react';
import Box from '@mui/material/Box';
import FormControl from '@mui/material/FormControl';
import FormControlLabel from '@mui/material/FormControlLabel';
import FormLabel from '@mui/material/FormLabel';
import InputLabel from '@mui/material/InputLabel';
import TextField from '@mui/material/TextField';
import Radio from '@mui/material/Radio';
import RadioGroup from '@mui/material/RadioGroup';
import Checkbox from '@mui/material/Checkbox';
import Select from '@mui/material/Select';
import MenuItem from '@mui/material/MenuItem';
import Button from '@mui/material/Button';
import { useNavigate } from 'react-router-dom';

function Form(props) {
  const [firstName, setFirstName] = useState('');
  const [lastName, setLastName] = useState('');
  const [email, setEmail] = useState('');
  const [gender, setGender] = useState('');
  const [favouriteColor, setFavouriteColor] = useState('');
  const [employed, setEmployed] = useState(false);
  const [notes, setNotes] = useState('');
  const [redirect, setRedirect] = useState(false);

  const navigate = useNavigate();
```

Obrázek 37. Pomůcky useState a useNavigate v souboru Form.jsx

Dále je třeba věnovat pozornost dvěma událostem formulářových polí. Za prvé detekci změny formulářového pole a aktualizování proměnné, která je tomuto poli přiřazena (událost *onChange*). Za druhé samotnému odeslání dat, které odpovídají hodnotám, jež zadal uživatel (událost *onSubmit*).

```
<TextField
  name="firstName"
  label="First name"
  fullWidth
  value={firstName}
  onChange={ e => {setFirstName(e.target.value)}}
/>
```

Obrázek 38. Příklad implementace události onChange

U události *onSubmit* je vhodné si uvědomit, že při odeslání nepotřebujeme znovu načítat celou stránku, a proto je v obsluhovací metodě této události vhodné zavolat metodu *preventDefault*. Prvním krokem v implementaci odesílání dat je vytvoření obsluhovací metody pro událost *onSubmit*, tuto metodu jsem pojmenoval *handleSubmit*, ta bude aplikaci notifikovat, pomocí změny stavu proměnné *redirect*, že si uživatel přeje odeslat zapsaná

data. Metodu *handleSubmit* je dále třeba přiřadit jako obslužnou metodu události *onSubmit* v tagu *form*.

Jakmile se zavolá metoda *handleSubmit*, tak se změní proměnná *redirect*, kterou hlídá další React pomůcka *useEffect* (importování stejné jako u *useState*). Ta, když zjistí, že je proměnná *redirect* pravdivá, tak přesměruje zapsaná data na adresu s výsledky.

```
const handleSubmit = e => {
    e.preventDefault();
    setRedirect(true);
}

useEffect(() => {
  if(redirect) {
    navigate("/results", {state:{
      firstName: firstName,
      lastName: lastName,
      email: email,
      gender: gender,
      favouriteColor: favouriteColor,
      employed: employed,
      notes: notes
    }});
}

}, [
  redirect,
  firstName,
  lastName,
  email,
  gender,
  favouriteColor,
  employed,
  notes,
  navigate
]);
```

Obrázek 39. Odesílání a přesměrování dat v Reactu

Požadovaným polím (jméno, příjmení a email) se na závěr přidá atribut *required* a poli pro email ještě atribut *type*, který se položí roven hodnotě *email* a tím se zajistí lepší kontrola tohoto pole.

4.3.3.2 Vyčištění formuláře

Podobně jako u předchozí aplikace i v Reactu lze vyčistit formulář pomocí funkce, v níž bude docházet k uvedení všech polí do původního stavu.

```
const handleClear = () => {
    setFirstName('');
    setLastName('');
    setEmail('');
    setGender('');
    setFavouriteColor('');
    setEmployed(false);
    setNotes('');
};
```

Obrázek 40. Metoda handleClear

4.3.4 Plocha pro zobrazení odeslaných dat

Vizuální stránka této komponenty je stejná jako u aplikace v Angularu. Skládá se hlavně z MUI komponent *List*, *ListItem*, *ListItemText* a *Divider*. A je zde i odkaz (*Link*) pro návrat na formulář. Pro získání dat uživatele, které byly uloženy pomocí pomůcky *useNavigate* se používá pomůcka *useLocation*. Tu je nutné přiřadit proměnné a pomocí této proměnné lze poté přistoupit k atributu *state*, v němž jsou uloženy odeslaná data.

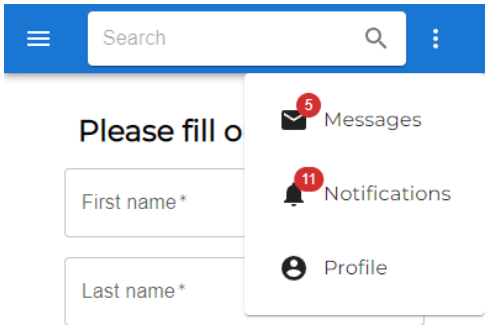
Zajímavé je, že komponentu *ListItemText* nelze upravovat pomocí atributu *sx*. Tato komponenta má vlastní atribut pro stylování a tím je *primaryTypographyProps*.

```
const location = useLocation();

<ListItem>
  <ListItemText primaryTypographyProps={{
    fontSize: '20px',
    fontWeight: "bold"
    }}
  >
    First name: {location.state.firstName}
  </ListItemText>
</ListItem>
```

Obrázek 41. Použití pomůcky useLocation

4.3.5 Mobilní menu

Mobilní menu se zde otevírá stejným způsobem jako v aplikaci u Angularu, a to ikonou se třemi tečkami v pravém horním rohu displeje. Mobilní menu je zde hlavně tvořeno MUI komponentami *Paper*, *MenuList*, *MenuItem* a *IconButton*.



Obrázek 42. Mobilní menu aplikace v Reactu

4.3.6 Zapouzdření aplikace

Všechny komponenty jsou uvedeny v hlavním souboru aplikace *App.js*, kde rovněž probíhá i směrování pomocí *React Routeru* a nachází se zde i logika pro otevírání a zavírání mobilního menu.

4.3.6.1 Směrování

Pro použití *React Routeru* je třeba importovat komponenty *BrowserRouter*, *Routes* a *Route* ze sekce *react-router-dom*. *BrowserRouter* obaluje všechny komponenty aplikace, uvnitř komponenty *Routes* se nachází všechny komponenty *Route*, které se budou lišit v závislosti na URL adrese. Komponenta *Route* má dva povinné atributy, a to cestu (*path*) a komponentu (*element*), která se bude na dané adrese zobrazovat. V aplikaci používám i atribut *exact*, který hlídá přesměrování na přesně danou adresu.

```
<BrowserRouter>
  <Header />
  <Routes>
    <Route path='/' exact element={<Form />} />
    <Route path='/results' exact element={<Formresults />} />
  </Routes>
  <Footer />
</BrowserRouter>
```

Obrázek 43. Směrování v Reactu

4.3.6.2 Obsluha mobilního menu

K otevírání a zavírání mobilního menu jsem použil již známou pomůcku *useState*, dále jsem vytvořil dvě metody (pro otevírání a zavírání menu), uvnitř kterých měním proměnnou vytvořenou pomocí *useState*, která říká, jestli má být mobilní menu viditelné nebo ne.

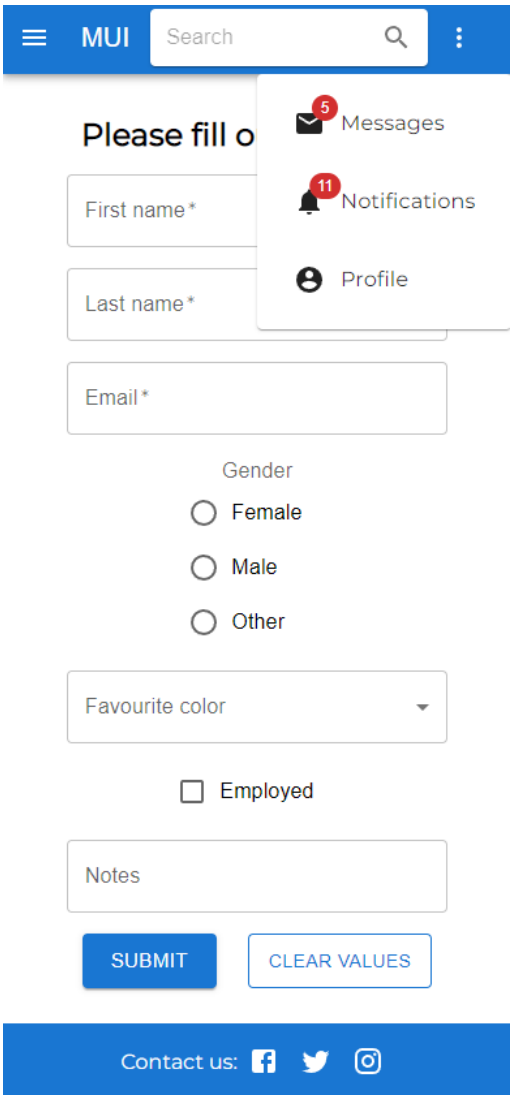
```
const [mobileMenuVisible, setMobileMenuVisible] = useState(false);

const showMobileMenu = () => {
  setMobileMenuVisible(true);
}

const closeMobileMenu = () => {
  if(mobileMenuVisible) {
    setMobileMenuVisible(false);
  }
}
```

Obrázek 44. Metody pro ovládání viditelnosti mobilního menu

Dále je nutné vytvořit dvě události, které budou obsluhovány vytvořenými metodami na obrázku 42. Jelikož mobilní menu lze otevřít pouze v hlavičce, tak událost pro otevření menu bude jen u komponenty hlavičky. Událost pro zavření mobilního menu bude definována i u ostatních komponent. Jakmile budou tyto události detekovány (kliknutím uživatele), tak se uvnitř dílčích komponent pomocí atributu komponent *props* zavolá událost v souboru *App.js*.

```
<BrowserRouter>
  <Header
    onShowMobileMenu={showMobileMenu}
    onCloseMobileMenu={closeMobileMenu}
  />
  {mobileMenuVisible && (<MobileMenu />)}
  <Routes>
    <Route path='/' exact element={<Form onCloseMobileMenu={closeMobileMenu} />} />
    <Route path='/results' exact element={<Formresults onCloseMobileMenu={closeMobileMenu}/>} />
  </Routes>
  <Footer onCloseMobileMenu={closeMobileMenu}/>
</BrowserRouter>
```

Obrázek 45. Přiřazení událostí pro obsluhu mobilního menu



Obrázek 46. Aplikave vyvíjená pomocí Reactu a Material UI

4.4 Vývoj demonstrační aplikace ve Vue a Vuetify

Na začátku si opět ručně vytvořím stejně jako u aplikace v Reactu složku pro komponenty a uvnitř vytvořím soubory pro všechny komponenty s příponou *vue*. Kvůli směrování je třeba nainstalovat Vue Router příkazem `vue add router`, poté se v projektu vytvoří dvě nové složky *router* a *views*.

Ve složce *router* se nachází soubor pro definování cest (*index.js*) a ve složce *views* jsou pohledy aplikace v závislosti na URL adrese. V demonstrační aplikaci se střídají dva pohledy, proto je třeba vytvořit ve složce *views* ještě jeden pohled *Results*. Šablona každého pohledu musí uvnitř obsahovat tag *v-app* v němž budou obsaženy všechny komponenty.

```
import { createRouter, createWebHashHistory } from 'vue-router'
import Home from '../views/Home.vue'
import Results from '../views/Results.vue'

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/results',
    name: 'results',
    component: Results,
    props: true
  }
]

const router = createRouter({
  history: createWebHashHistory(),
  routes
})

export default router
```

Obrázek 47. Definice směrovacích cest ve Vue v souboru index.js

Soubor komponenty ve Vue je rozdělen do 3 částí šablony (*template*), skriptů (*script*) a kaskádových stylů (*style*). Veškeré vizuální úpravy budou uvedeny v části *style*.

4.4.1 Hlavička

Hlavička má opět stejnou strukturu jako předešlé aplikace. Vyhledávací ikona je tvořena vstupním polem (*input*) a ikonou lupy. Větší části hlavičky jsou obaleny tagem *v-container*. Ikony s notifikacemi jsou tvořeny tagy *v-btn*, *v-icon* a *v-badge*.

Obrázek 48. Hlavička aplikace ve Vue

4.4.2 Patička

Patička i zde obsahuje tlačítkové ikony s odkazem na sociální sítě. V rámci úspory kódu je v šabloně použita direktiva *v-for*, pomocí níž jsou zobrazeny tlačítka. K použití této direktivy u tlačítek je třeba definovat ve skriptové části komponenty konkrétně v sekci *data*, seznam objektů reprezentující dané tlačítka. Instance tlačítka má dva parametry název ikony (*icon*) a odkaz na sociální síť (*link*).

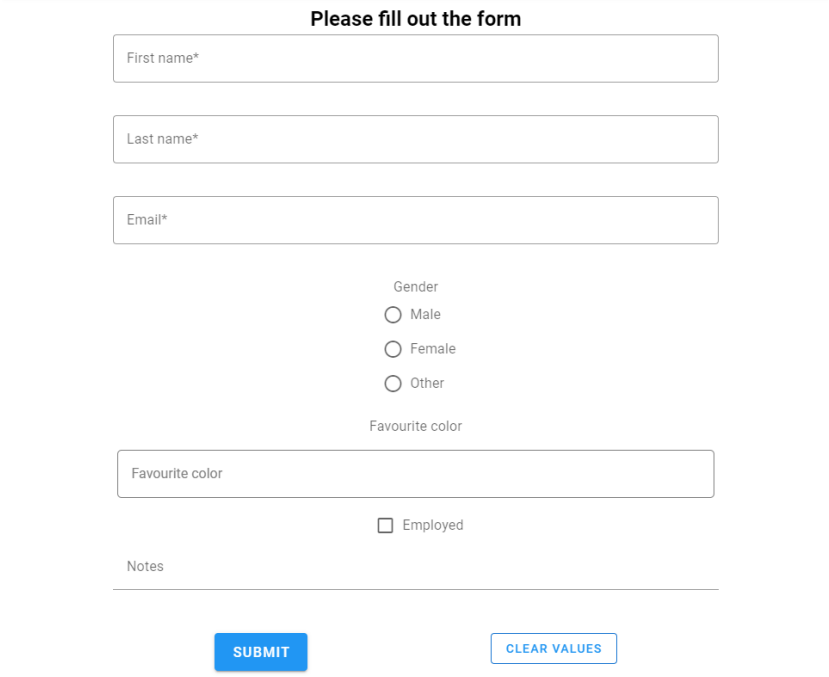
```
<v-btn
  v-for="button in buttons"
  :key="button"
  class="nav-icon"
  icon
  elevation="0"
  :href= button.link
>
  <v-icon size="24px">
    {{ button.icon }}
  </v-icon>
</v-btn>
```

Obrázek 49. Direktiva v-for

Obrázek 50. Patička aplikace ve Vue

4.4.3 Formulář

Formulář se skládá ze stejných typů polí jako předchozí aplikace. Většina polí je reprezentována komponentami Vuetify, například textové pole je reprezentováno tagem *v-text-field.* Jedinou výjimkou je výběrové pole, kde Vuetify verze 3.0.0 nenabízí svou komponentu pro tento typ pole, proto jsem použil variantu klasického HTML.



Obrázek 51. Formulář aplikace ve Vue

4.4.3.1 Odesílání a validace dat formuláře

Pro úspěšné odesílání dat je nejprve třeba v části *data* skriptové části vytvořit proměnné, které reprezentují všechna formulářová pole. Tyto proměnné je nutné nastavit na výchozí stav nevyplněného formuláře. V dalším kroku je důležité nastavit v šabloně každému formulářovému poli hodnotu definovanou v předešlém kroku pomocí direktivy *v-model*. Pro křestní jméno je direktiva následující *v-model* následující `v-model="firstName"`.

V sekci *data* skriptové části je vhodné vytvořit i proměnné, *firstNameError*, *lastNameError* a *emailError*, které v případě nevyplnění důležitých polí varují uživatele.

```
data: () => ({
  firstName: "",
  lastName: "",
  email: "",
  gender: "",
  favouriteColor: "",
  employed: false,
  notes: "",
  firstNameError: false,
  lastNameError: false,
  emailError: false,
}),
```

Obrázek 52. Definice důležitých proměnných formuláře

```
<v-label v-if="firstNameError" class="warning">First name is required</v-label>
<v-text-field
  label="First name*"
  color="info"
  variant="outlined"
  v-model="firstName"
></v-text-field>
```

Obrázek 53. Implementace direktiv v-if a v-model

Nyní je třeba vytvořit v sekci *export default* sekci *methods* a v ní metodu, která bude odesílat data vyplněného formuláře. Uvnitř této metody se bude kontrolovat vyplnění požadovaných polí. Vyplněná data se následně budou posílat pomocí *routeru* na adresu pro zobrazení výsledků.

```
submitForm() {
  if(this.firstName != "" &&
     this.lastName != "" &&
     this.email != "" &&
     /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(this.email)
    ) {
    this.$router.push({
      name: 'results',
      params: {
        firstName: this.firstName,
        lastName: this.lastName,
        email: this.email,
        gender: this.gender,
        favouriteColor: this.favouriteColor,
        employed: this.employed,
        notes: this.notes
      }
    });
  }
  else {
    if(this.firstName == "") this.firstNameError = true;
    if(this.lastName == "") this.lastNameError = true;
    if(this.email == "" ||
      /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(this.email) == false
    ) {
      this.emailError = true;
    }
  }
}
```

Obrázek 54. Metoda submitForm

4.4.3.2 Vyčištění formuláře

Funkce pro vyčištění formuláře funguje na stejném principu jako u předchozích aplikací, to znamená uvedení proměnných, které obsluhují formulářové pole, do původního stavu. Tuto metodu je nutné definovat ve vytvořené sekci *methods* skriptové části.

```
clearForm() {
  this.firstName = "";
  this.lastName = "";
  this.email = "";
  this.gender = "";
  this.favouriteColor = "";
  this.employed = false;
  this.notes = "";
}
```

Obrázek 55. Metoda clearForm

4.4.4 Plocha pro zobrazení odeslaných dat

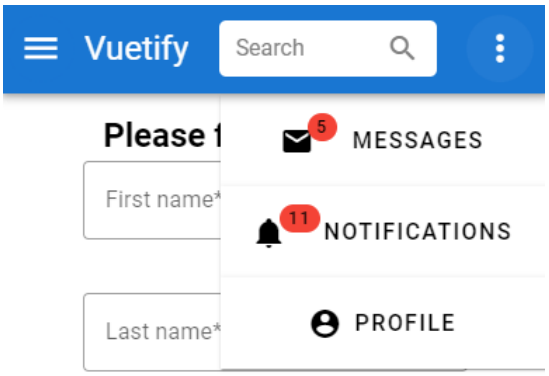
Vzhled této komponenty je stejný jako u předchozích aplikací. Tato komponenta je tvořena hlavně tagy *v-list*, *v-list-item*, *v-list-item-header* a *v-list-item-title*. Neměl by chybět odkaz pro možnost vrácení se na hlavní stránku (*router-link*). K datům se v šabloně přistupuje pomocí *routeru* a atributu *params*.

```
<v-list-item>
  <v-list-item-header>
    <v-list-item-title>First name: {{ this.$route.params.firstName }}</v-list-item-title>
  </v-list-item-header>
</v-list-item>
```

Obrázek 56. Získání přijatých dat

4.4.5 Mobilní menu

Mobilní menu je tvořeno tlačítky s ikonami různých upozornění.



Obrázek 57. Mobilní menu aplikace ve Vue

Podobně jako u Reactu je i zde třeba vytvořit proměnnou, která bude určovat viditelnost mobilního menu, tuto proměnnou je nezbytné vytvořit v každém pohledu (*view*). Ve všech pohledech aplikace je nutné vytvořit i dvě metody pro otevírání a zavírání mobilního menu.

```
data: () => ({
  mobileMenuVisible: false
}),
methods: {
  showMobileMenu() {
    this.mobileMenuVisible = !this.mobileMenuVisible;
  },

  closeMobileMenu() {
    if(this.mobileMenuVisible) {
      this.mobileMenuVisible = false;
    }
  }
},
```

Obrázek 58. Metody pro obsluhu viditelnosti mobilního menu ve Vue

Pro zavírání menu stačí už jen přidat vhodným komponentám (*Form*, *FormResults* a *Footer*) událost kliknutí, kterému se přiřadí metoda *closeMobileMenu*.

Protože lze otevírat mobilní menu jen v hlavičce pomocí ikony s třemi tečkami, tak je nutné vytvořit vlastní událost *showMobMenu*, kterou bude obsluhovat metoda *showMobileMenu*.

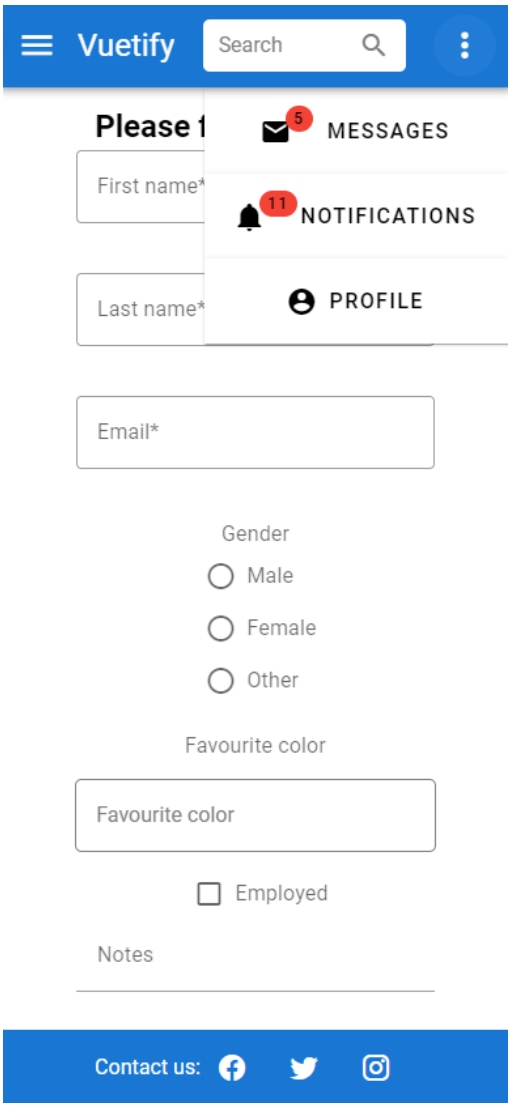
```
<v-app>
  <Header @showMobMenu="showMobileMenu"></Header>
  <MobileMenu v-if="mobileMenuVisible"></MobileMenu>
  <Form @click="closeMobileMenu"></Form>
  <Footer @click="closeMobileMenu"></Footer>
</v-app>
```

Obrázek 59. Implementace událostí řídící viditelnost mobilního menu v souboru Home.vue

V hlavičce u ikony tří teček je dále třeba přidat událost kliknutí, které se přiřadí rodičovská událost *showMobMenu* pomocí funkce *emit*. Záznam o notifikaci rodičovské události je třeba uvést ve skriptové části řádkem `emits: ["showMobMenu"]`.

```
<v-btn @click="$emit('showMobMenu')" icon class="nav-icon d-flex d-xs-flex d-md-none">
  <v-icon>mdi-dots-vertical</v-icon>
</v-btn>
```

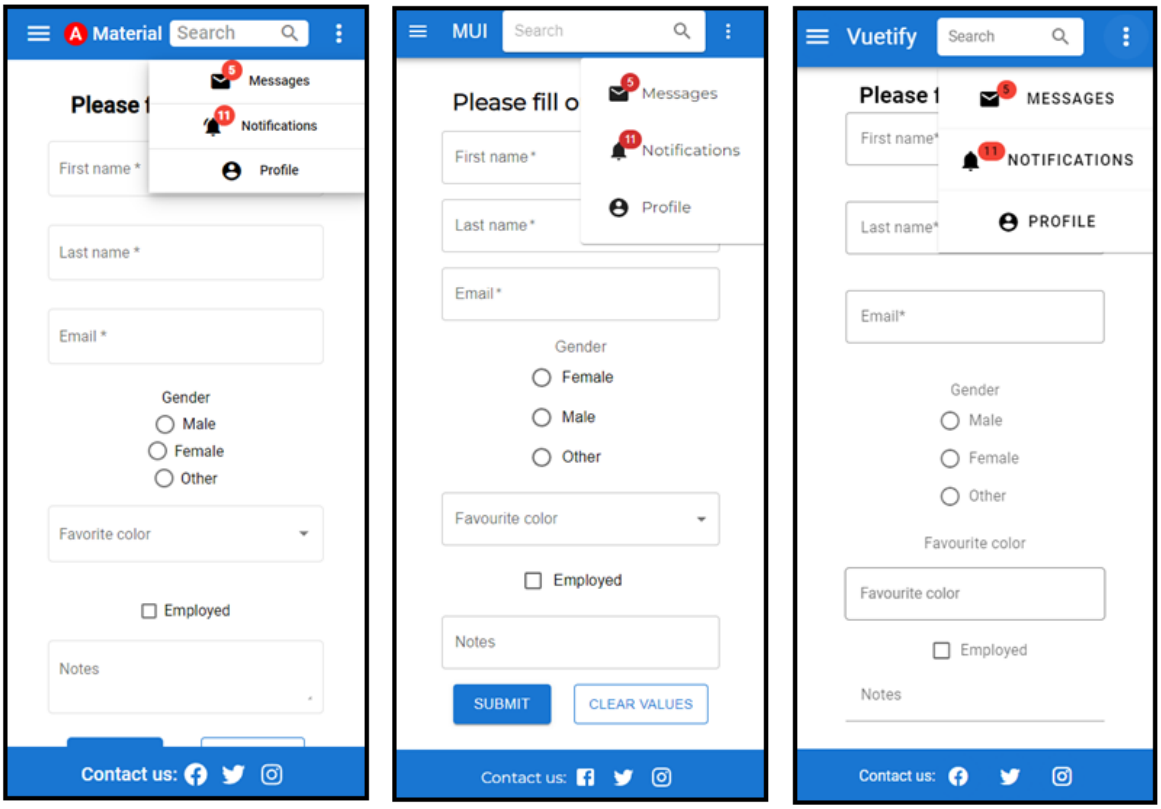
Obrázek 60. Tlačítko pro otevření mobilního menu



Obrázek 61. Aplikace vyvíjená pomocí Vue a Vuetify

5 POROVNÁNÍ VÝVOJE DEMONSTRAČNÍ APLIKACE V POUŽITÝCH NÁSTROJÍCH

Tato část se věnuje shrnutí vývoje demonstrační aplikace v jednotlivých frameworcích a jejich rozšiřujících knihovnách a následně porovnáním vývoje těchto nástrojů.



Obrázek 62. Tři varianty uživatelského rozhraní

5.1 Shrnutí vývoje demonstrační aplikace v Angularu a Angular Material

Angular je specifický tím, že v jednom souboru komponenty nemísí vývojové jazyky (HTML, CSS a JavaScript/TypeScript), ale pro každý má vlastní soubor, což může být pro mnoho lidí výhoda. Dalším znakem Angularu je soubor *app.module.ts*, kde lze vidět pohromadě veškeré vytvořené komponenty a moduly různých knihoven, které se v aplikaci používají.

Největším problémem během vývoje bylo hledání správného způsobu odesílání formulářových dat na plochu pro zobrazení výsledků pomocí *routeru*, to je způsobeno tím, že se Angular rychle vyvíjí a modernizuje, kvůli velkému množství verzí Angularu dochází k upravování určitých postupů a operací.

Co se týče knihovny AM, tak dokumentace je jednoduchá a přehledná. Pro začátečníky je velmi přínosné, že je u každé komponenty vidět mnoho různých podob a využití komponenty. Nedostatkem AM je sada *Material icons*, která neobsahuje ikony moderních sociálních (Twitter, Instagram). Kvůli tomu bylo třeba v patičce aplikace použít univerzálních ikon *Font Awesome*. Další nedostatkem je u tlačítek absence atributu *size*, kterým by vývojář mohl rychle nastavit velikost tlačítka, namísto toho je vývojář nucen tlačítko ručně stylovat.

5.2 Shrnutí vývoje demonstrační aplikace v Reactu a Material UI

Na rozdíl od Angularu se v Reactu skládá komponenta pouze z jednoho souboru, ve kterém lze upravovat šablonu, kaskádové styly a psát funkční logiku komponenty. Uvnitř každé komponenty je nutné v hlavičce importovat veškeré komponenty od MUI, to znamená několik stejných importů v několika komponentách.

Na žádný větší problém jsem při vývoji demonstrační aplikace nenarazil, a to hlavně díky sadě pomůcek *React Hooks*, které pomáhají řešit různé typy operací s daty a zároveň dělají kód čitelnějším a přehlednějším.

Knihovna MUI má velmi obsáhlou, ale přehlednou dokumentaci. Výhodou dokumentace je chytré vyhledávací pole, které vývojáři pomůže najít to, co hledá. Další výhodou je stejně jako u AM velké množství příkladů použití všech nabízených komponent. Za zmínku stojí i atribut *sx*, který lze použít jen u komponent MUI. Tento atribut umožňuje rychlé stylování MUI komponent.

5.3 Shrnutí vývoje demonstrační aplikace ve Vue a Vuetify

Ve Vue tvoří komponentu také jen jeden soubor, v porovnání s Reactem je kód Vue komponenty přehledněji strukturován, šablona je obalena tagem *template*, skriptová část tagem *script* a kaskádové styly tagem *style*. Vue se může chlubit vlastním grafickým nástrojem pro tvorbu nebo úpravu projektů, což ocení hlavně začátečníci. Pro některé vývojáře může být výhodou i přehledné rozvržení všech pohledů aplikace v samostatné složce.

Knihovna Vuetify má také přehlednou dokumentaci plnou příkladů využití jednotlivých komponent. Vuetify verze 3.0.0 je sice v testovací verzi, nicméně nedostatkem je zde absence funkční komponenty výběrového pole (*select*), které je důležitou součástí většiny formulářů. Aby se tedy formuláře skládaly ze stejných polí, tak jsem v této variantě použil klasické výběrové pole jazyka HTML.

5.4 Společné výhody všech rozšiřujících knihoven

Používání rozšiřujících knihoven dokáže velmi zefektivnit vývoj aplikace a ušetřit mnoho času. Mezi hlavní výhody patří předpřipravené komponenty, které by vývojář musel jinak zdlouhavě stylovat. Mezi takovéto komponenty patří:

- *Dialog*
- Hodnota počtu notifikací (*Badge*)
- Lišta aplikace (*App bar* nebo *Toolbar*)
- Několik typů formulářových polí (*text field*, *select*, *textarea*) v různých variantách (*Outlined*, *Filled* a *Standard*)
- *Parallax*
- *Progress bar*
- *Slider*
- Stránkování (*Pagination*)
- Vizuálně předpřipravená tlačítka

5.5 Porovnání frameworků a rozšiřujících knihoven

Pro zpřehlednění již zmíněných výhod a nevýhod frameworků a jejich rozšiřujících knihoven jsem vytvořil tabulku, kde je vše důležité z toho, co jsem během vývoje tří demonstračních aplikací zaznamenal. V každém řádku tabulky je zaznamenáno, jestli dané nástroje obsahují nebo podporují určitou věc.

Tabulka 1. Porovnání nástrojů použitých při vývoji demonstračních aplikací

	Angular, Angular Material	React, Material UI	Vue, Vuetify
Rozdělení komponenty frameworku do několika souborů podle jazyka	ANO	NE	NE
Grafické rozhraní pro tvorbu a úpravu projektu	NE	NE	ANO
Vlastní direktivy	ANO	NE	ANO
Atribut *size* pro tlačítka rozšiřujících knihoven	NE	ANO	ANO
Vyhledávací pole v dokumentaci rozšiřující knihovny	NE	ANO	NE
Komponenta výběrového pole rozšiřující knihovny	ANO	ANO	NE
Vlastní ikony sociálních sítí	NE	ANO	ANO
Vlastní způsob stylování	NE	ANO	NE

Dle tabulky 1 lze usoudit, že nejvíce užitečných nástrojů pro tvorbu demonstrační aplikace nabízí varianta React a Material UI. Nicméně je důležité říci, že vždy budou rozhodovat osobní preference vývojáře. Pokud například vývojář nadevše preferuje rozdělení komponenty do několika souborů, tak si bezpochyby k vývoji vybere Angular a Angular Material. Tato tabulka tedy slouží jako přehled, podle kterého se může vývojář rozhodnout, které nástroje využije.

Pro porovnání rychlosti načtení aplikace, velikosti stažených dat a dalších tzv. tvrdých dat, slouží tabulka 2. Uvedená data jsem získal pomocí nástrojů *Chrome DevTools*. V záložce *network* bylo provedeno 20 časových měření. Aby byla všechna měření podobná, tak bylo třeba zakázat ukládání dat do rychlé *cache* paměti.

Tabulka 2 Porovnání fyzických parametrů aplikací

	Angular, Angular Material	React, Material UI	Vue, Vuetify
Datová velikost projektu	384 MB	216 MB	141 MB
Průměrná doba načítání aplikace	956,3 ms	966,8 ms	1351,5 ms
Nejkratší doba načtení aplikace	833 ms	904 ms	1100 ms
Nejdelší doba načtení aplikace	1210 ms	1300 ms	2130 ms
Množství stažených dat při načtení aplikace	5,3 MB	5,6 MB	7,1 MB
Počet zpracovaných požadavků při načtení aplikace	14	11	18

Z tabulky 2 vyplývá, že čím více potřebných dat musí aplikace při načítání stáhnout, tím pomaleji její načítání trvá.

ZÁVĚR

Tato práce se zabývá porovnáním vývoje webového uživatelského rozhraní pomocí frameworků Angular, React a Vue a jejich rozšiřujících knihoven. V teoretické části se čtenář dočte, co to uživatelské rozhraní je, proč jej potřebujeme a jaké důležité principy bychom měli při jeho návrhu dodržovat. Dále je čtenář seznámen s moderními nástroji, které se používají pro vývoj webových aplikací. Práce hovoří o tom, jak dané frameworky a rozšiřující knihovny nainstalovat a jak vhodně využívat jejich předností.

V praktické části práce byl nejprve vytvořen návrh demonstrační aplikace a jejího uživatelského rozhraní podle doporučených fází dle teoretické části. Protože dnes patří ke každodennímu životu mobilní telefony, tak se návrh uživatelského rozhraní zaměřuje i na responzivitu aplikace. Po dokončení návrhu demonstrační aplikace se práce volně posouvá k vývoji této aplikace ve třech variantách, každá varianta byla vytvořena za pomoci jednoho z frameworků a jeho rozšiřující knihovny. Hlavní funkcionalitou demonstrační aplikace je zpracování dat uživatelem vyplněného formuláře. Povedlo se i implementovat základní validaci vybraných formulářových polí.

V závěru práce shrnuje přínosy a poukazuje na nedostatky použitých nástrojů během vývoje demonstračních aplikací. Výsledkem práce je dvojí porovnání všech tří variant demonstrační aplikace. První porovnání se zaměřuje na nabízené nástroje, které může vývojář při práci s daným frameworkem a rozšiřující knihovnou využít. Druhé porovnání se pak týká rychlosti spuštění aplikace a množství stažených dat při načtení. Tato porovnání nabízí čtenáři přehled, pomocí něhož se dokáže snadněji rozhodnout, které nástroje pro vývoj webových aplikací by mu vyhovovaly.

Když bych měl shrnout přínos práce, tak by to bylo seznámení se základními operacemi dat v jednotlivých frameworcích a poukázání na to, jak mohou rozšiřující knihovny zefektivnit vývoj aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] Most used web frameworks among developers worldwide, as of 2021. *Statista* [online]. 2021 [cit. 2022-05-02]. Dostupné z: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

[2] Uživatelské rozhraní. *TechLib* [online]. 2006 [cit. 2022-01-26]. Dostupné z: https://tech-lib.eu/definition/user_interface.html

[3] Návrh uživatelského rozhraní webové aplikace. *Katedra multimédií - Vysoká škola ekonomická v Praze* [online]. 2018 [cit. 2022-01-26]. Dostupné z: https://kme.vse.cz/wp-content/uploads/page/534/10.-N%C3%A1vrh-u%C5%BEivatelsk%C3%A9ho-rozhran%C3%AD-webov%C3%A9-aplikace.pdf

[4] Uživatelské prostředí. *L-production* [online]. 2022 [cit. 2022-01-27]. Dostupné z: http://www.lproduction.cz/uzivatelske-prostredi-96.htm

[5] Javascriptové frameworky. *Katedra multimédií - Vysoká škola ekonomická v Praze* [online]. 2018 [cit. 2022-01-31]. Dostupné z: https://kme.vse.cz/wp-content/uploads/page/534/9.-Javascriptov%C3%A9-frameworky.pdf

[6] Javascript. *Jan Štráfelda* [online]. 2015 [cit. 2022-01-31]. Dostupné z: https://www.strafelda.cz/javascript

[7] Introduction to the Angular Docs. *Angular* [online]. ©2010-2022 [cit. 2022-01-31]. Dostupné z: https://angular.io/docs

[8] SESHADRI, Shyam. *Angular: up and running : learning Angular, step by step*. Sebastopol, CA: O'Reilly Media, [2018]. ISBN 1491999837.

[9] Introduction to Angular concepts. *Angular* [online]. ©2010-2022 [cit. 2022-02-01]. Dostupné z: https://angular.io/guide/architecture

[10] How to Install Angular on Windows: A Guide to Angular CLI, Node.js, and Build Tools. *FreeCodeCamp* [online]. 2019 [cit. 2022-02-02]. Dostupné z: https://www.freecodecamp.org/news/how-to-install-angular-on-windows-a-guide-to-angular-cli-node-js-and-build-tools/

[11] EditorConfig Specification. *EditorConfig* [online]. ©2019--2020 [cit. 2022-02-23]. Dostupné z: https://editorconfig-specification.readthedocs.io/

[12] Gitignore Explained: What is Gitignore and How to Add it to Your Repo. *FreeCodeCamp* [online]. 2020 [cit. 2022-02-02]. Dostupné z: https://www.freecodecamp.org/news/gitignore-what-is-it-and-how-to-add-to-repo/

[13] What is README.md File?. *GeeksforGeeks* [online]. 2021 [cit. 2022-02-02]. Dostupné z: https://www.geeksforgeeks.org/what-is-readme-md-file/

[14] Built-in directives. *Angular* [online]. ©2010-2022 [cit. 2022-02-04]. Dostupné z: https://angular.io/guide/built-in-directives

[15] Difference between ReactJS and React Native. *Javatpoint* [online]. © 2011-2021 [cit. 2022-02-05]. Dostupné z: https://www.javatpoint.com/reactjs-vs-reactnative

[16] Understanding Component-Based Architecture. *Medium* [online]. 2016 [cit. 2022-02-07]. Dostupné z: https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238

[17] Component-Based Architecture. *Tutorials Point* [online]. 2021 [cit. 2022-02-07]. Dostupné z: https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm

[18] Create-react-app files/folders structure explained. *Medium* [online]. 2020 [cit. 2022-02-08]. Dostupné z: https://medium.com/@abesingh1/create-react-app-files-folders-structure-explained-df24770f8562

[19] BANKS, Alex a Eve PORCELLO. *Learning react: functional web development with react and redux*. Sebastopol ;: O'Reilly, 2017. ISBN 978-1-491-95462-1.

[20] What is Vue.js?. *Vue.js* [online]. © 2014-2022 [cit. 2022-02-09]. Dostupné z: https://v2.vuejs.org/v2/guide/#What-is-Vue-js

[21] Quick Start. *Vue.js* [online]. © 2014-2022 [cit. 2022-02-09]. Dostupné z: https://vuejs.org/guide/quick-start.html

[22] A quick introduction to Vue.js. *FreeCodeCamp* [online]. 2018 [cit. 2022-02-11]. Dostupné z: https://www.freecodecamp.org/news/a-quick-introduction-to-vue-js-72937ee8880d/

[23] Config Files. *Babel* [online]. 2018 [cit. 2022-02-11]. Dostupné z: https://babeljs.io/docs/en/config-files

[24] MACRAE, Callum. *Vue.js: up and running : building accessible and performant web apps*. Sebastopol: O'Reilly, 2018. ISBN 978-1-491-99724-6.

[25] Icons. *Material UI* [online]. © 2022 [cit. 2022-02-28]. Dostupné z: https://mui.com/components/icons/#main-content

[26] Introduction. *Vuetify* [online]. 2022 [cit. 2022-02-28]. Dostupné z: https://vuetifyjs.com/en/introduction/why-vuetify/#what-is-vuetify3f

[27] Introduction. *Material Design* [online]. 2022 [cit. 2022-02-28]. Dostupné z: https://material.io/design/introduction#components

[28] Introducing JSX. *React* [online]. © 2022 [cit. 2022-05-03]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
CLI	Command Line Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
GUI	Graphical User Interface
HTML	HyperText Markup Language
MB	Megabyte
MDI	Material Design Icons
MS	Millisecond
UCD	User Centered Design
UI	User Interface
URL	Uniform Resource Locator
UXD	User Experience Design

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD DISK

Přiložené CD k bakalářské práci obsahuje:

- Bakalářská práce ve formátu PDF
- Zdrojové kódy aplikací
- Seznam zaznamenaných časových měření