VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vývoj komponent pro framework Angular

Development of Components for the Angular Framework

Jan Slobodník

Bakalářská práce

Vedoucí práce: Ing. Jan Janoušek

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/079084
vsboee220261C8

Zadání bakalářské práce

Student: **Jan Slobodník**

Studijní program: B0613A140014 Informatika

Téma: Vývoj komponent pro framework Angular
Development of Components for the Angular Framework

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit sadu komponent pro framework Angular, zaměřených na práci s mapou. Půjde například o vyhledávání, plánování trasy, vizualizaci výškového profilu, správu vrstev, ovládání mapy, import tras a bodů zájmů v různých formátech a případně další.

Hlavní body zadání:
1. Seznámení s frameworky pro tvorbu webových aplikací, frameworkem Angular a problematikou vývoje komponent pro tento framework.
2. Návrh a implementace jednotlivých komponent.
3. Vytvoření ukázkové aplikace demonstrují možnosti jednotlivých komponent.

Seznam doporučené odborné literatury:

[1] FAIN, Yakov a Anton MOISEEV. Angular Development with TypeScript. 2. Manning Publications, 2018. ISBN 9781617295348.
[2] SAVKIN, Victor a Jeff CROSS. Essential Angular. Packt Publishing Ltd, 2017. ISBN 9781788291040.
[3] SAVKIN, Victor a Jeff CROSS. Angular Router. Packt Publishing, 2017. ISBN 9781787287150.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Janoušek**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 07.11.2022 12:25:53

Abstrakt

Cílem práce je vytvoření skupiny komponent pro framework Angular, které se zaměřují na práci s mapou. Komponenty budou umožňovat práci s geografickými daty, které budou využity při vizualizaci výškového profilu, plánování tras či správě vrstev. Tyto komponenty budou následně použity ve vzorové aplikaci testující jejich funkcionalitu. Součástí práce je také seznámení s frameworkem Angular a jeho porovnání s ostatními frameworky pro vývoj jednostránkových webových aplikací.

Klíčová slova

Angular; mapy; JavaScript; TypeScript; framework; geodata;

Abstract

The aim of this thesis is to create a group of components for the Angular framework that focus on working with the map. The components will allow working with geographic data, which will be used for elevation profile vizualization, planning routes or managing map layers. These components will be used in a sample application testing their functionality. This thesis also includes an introduction to the Angular framework and its comparison with other frameworks for developing single-page web applications.

Keywords

Angular; maps; JavaScript; TypeScript; framework; geodata;

Poděkování

Rád bych zde poděkoval vedoucímu práce Ing. Janu Janouškovi za cenné rady a ochotu při tvorbě bakalářské práce.

Obsah

Seznam použitých zkratek a symbolů

HTML	–	Hypertext Markup Language
DOM	–	Document Object Model
SPA	–	Single-page application
API	–	Application Programming Interface
XML	–	Extensible Markup Language
JSON	–	JavaScript Object Notation
KML	–	Keyhole Markup Language
HTTP	–	Hypertext Transfer Protocol
BLOB	–	Binary Large OBject
CSS	–	Cascading Style Sheets

Seznam obrázků

Kapitola 1

Úvod

Základem většiny frameworků pro vývoj jednostránkových aplikací jsou prvky, reprezentující části uživatelského rozhraní, zvané komponenty. Výjimkou není ani framework Angular, jehož komponenty jsou vyvíjeny tak, aby byly znovupoužitelné a dostupné pro ostatní vývojáře v komunitě tohoto frameworku.

Cílem této práce je vytvořit řadu knihoven pro framework Angular, obsahujících komponenty, které se zaměřují převážně na práci s mapou a geografickými daty. Tyto komponenty budou řešit například plánování tras, zobrazení výškového profilu či správu vrstev mapy. Pro demonstraci použitelnosti komponent bude vytvořena aplikace, která ukazuje k jakým účelům mohou být tyto komponenty použity a jak mohou vzájemně interagovat.

Součástí práce je také seznámení s dalšími frameworky pro vývoj webových aplikací a detailnější pochopení vývoje komponent pomocí frameworku Angular. Práce je rozdělena na 4 části. První se zabývá představením a srovnáním některých populárních frameworků pro vývoj webových aplikací. Druhá kapitola popisuje detailněji základní principy a výhody vývoje knihoven a jejich komponent ve frameworku Angular. V třetí kapitole jsou obsaženy informace o postupech použitých při návrhu a implementaci konkrétních knihoven. Poslední část popisuje vytvoření vzorové aplikace a použití jednotlivých komponent.

Kapitola 2

Frameworky pro tvorbu webových aplikací

Framework pro tvorbu webových aplikací je kolekce knihoven, pravidel, nástrojů či doporučených postupů, sloužící k usnadnění vývoje webových aplikací. V dnešní době jsou k dispozici desítky takových frameworků, které se mohou lišit v zaměření, funkcionalitě či použité technologii. Velice populární jsou takzvané SPA (Single-page application) [1] frameworky, umožňující vývoj právě jednostránkových aplikací. Jedny z nejpoužívanějších SPA frameworků jsou React.js, Angular či Vue.js. Tyto frameworky vycházejí z jazyka JavaScript a budou následně popsány a porovnány v závislosti na jejich výhodách a nevýhodách. Protože jsou tyto frameworky založeny na JavaScriptu, bude taktéž popsán framework Blazor, který je na rozdíl od předešlých frameworků založen na technologii .NET a programovacím jazyce C#.

2.1 Angular

Angular je framework pro vývoj webových aplikací, vyvinutý společností Google, jehož cílem je převážně umožnit vytvářet jednostránkové aplikace, které rychle reagují na uživatelské události. Součástí Angularu je kolekce knihoven, obsahující velké množství funkcí, které řeší například navigaci, správu stavu, řízení formulářů či HTTP komunikaci. Angular obsahuje také mnoho již připravených komponent, které může vývojář použít a ušetřit čas při tvorbě aplikace. Výhodou je také rozsáhlá dokumentace a podpora velké komunity uživatelů, které umožňují novým vývojářům získat všechny potřebné informace o vývoji aplikací [2].

Angular je oproti ostatním frameworkům založen na jazyce TypeScript. TypeScript je nadstavba jazyka JavaScript a rozšiřuje jej o několik funkcionalit. Mezi rozdíly patří například statická typová kontrola, která umožňuje detekovat chyby v kódu v době překladu, tvorba tříd a rozhraní, či generické programování. Kód TypeScriptu je při kompilaci nejprve zkompilován do JavaScriptu, z čehož vyplývá, že kódy JavaScriptu jsou zároveň validními kódy TypeScriptu [3].

Jednou z nevýhod Angularu může být jeho větší složitost. Především začínající vývojáři mohou oproti frameworkům React a Vue.js považovat Angular za složitý a to kvůli jeho komplexnosti

a rozsáhlému množství témat, které obsahuje. Protože je Angular velice rozsáhlý a obsahuje mnoho funkcí, má větší velikost a tedy i větší nároky na hardware, což může ovlivnit výkon aplikace a rychlost načítání webové stránky. Na rozdíl od React a Vue.js Angular nepoužívá virtuální DOM, ale ten skutečný, který přímo aktualizuje [4].

2.2 Vue.js

Vue.js je JavaScriptový framework pro tvorbu uživatelských rozhraní a jednostránkových aplikací. Framework Vue.js je oproti Angularu jednoduší, snadněji použitelný a přívětivější k naučení. Pro začínajícího vývojáře stačí mít základní znalosti HTML, CSS a JavaScriptu a není třeba znát jiné technologie jako například TypeScript u Angularu nebo JSX u Reactu. Vue.js má malou velikost, což má vliv na rychlost instalace, výkon aplikace a rychlejší načítání obsahu webové stránky. Stejně jako React využívá Vue.js virtuální DOM [5].

Vue.js také nabízí několik knihoven pro animace, které obsahují například komponenty pro aplikování přechodů u HTML elementů. Pro použití přechodů existují ve Vue.js komponenty **Transition** a **TransitionGroup**. Komponenta Transition slouží k přidání efektů přechodu u elementů, které byly přidány či odebrány z DOM. Ukázka použití komponenty Transition i s příslušnými CSS třídami je na příkladu kódu 2.1 [6].

```
<button @click="show = !show">Show</button>
<Transition>
  <p v-if="show">hello</p>
</Transition>
...
//css styly komponenty Transition
.v-enter-active,
.v-leave-active {
  transition: opacity 0.5s ease;
}
.v-enter-from,
.v-leave-to {
  opacity: 0;
}
```

Listing 2.1: Příklad použití komponenty Transition [6]

K Vue.js existuje kvalitní dokumentace a podpora komunity, které ale nejsou tak rozsáhlé jako u frameworků Angular či React.js. Velká část komunity Vue.js je zastoupena v Číně, proto některé řešení či návody nemusí být dostupné v angličtině. V porovnání s ostatními frameworky obsahuje

Vue.js menší množství funkcí, nástrojů a knihoven, což může být problém při vývoji složitějších a rozsáhlejších aplikací [5].

Stejně jako ostatní zmíněné frameworky je Vue.js založeno na komponentách. Každá komponenta obsahuje šablonu HTML, styly CSS a script, ve kterém je definována práce z daty a chování komponenty. Data komponenty lze zobrazit v HTML kódu šablony, která se značí tagem `<template>`. Příklad jednoduché komponenty lze vidět na ukázce kódu 2.2.

```
<script>
    export default{
        data() {
            return {
                count: 0
            }
        },
        methods:
            CountUp(){
                this.count++;
            }
    }
</script>

<template>
    <button @click="count++"> Count is : {{count}} </button>
    <button @click="CountUp()"> Count is : {{count}} </button>
</template>
<style>
    button {
        font-size: 17px;
    }
</style>
```

Listing 2.2: Vue.js komponenta [6]

Data komponenty jsou zde reprezentována pomocí funkce *data()*, která vrací objekt obsahující jednotlivé vlastnosti, jakožto proměnné komponenty. Tyto proměnné lze použít pro zobrazení dat v šabloně komponenty nebo pro práci a úpravu v metodách komponenty. Pro zobrazení dat v šabloně slouží, stejně jako u frameworku Angular, data binding. Hodnotu proměnné lze v šabloně zobrazit pomocí složených závorek "**{{}}**" (viz. 2.2).

Vue.js také rozšiřuje HTML elementy o takzvané direktivy. Jednou z nejpoužívanějších je direktiva **v-bind**, která se používá k dynamickému přiřazení hodnoty proměnné k HTML atributu.

Stejným způsobem slouží *v-bind* také pro předávání dat z rodičovské komponenty do komponenty potomka. Zde se přiřazují hodnoty k tzv.*props*, které slouží jako vstupní data komponenty. V praxi se místo klíčového slova *v-bind* využívá ekvivalentní dvojtečka "**:**". Příklad použití direktivy *v-bind* je na ukázce kódu 2.3.

```
<ChildComponent :title="post.title"/>
<ChildComponent :title="post.title + 'by' + post.author.name"/>
<ChildComponent :likes="42"/>
<ChildComponent :posted="true"/>
<a v-bind:href="url"></a>
<img :width="200"></img>
```

Listing 2.3: Použití direktivy v-bind [6]

Jako další typ bindigu ve Vue.js jsou reakce na uživatelské události v HTML elementech, které jsou zde řešeny opět pomocí jedné z direktiv a to **v-on**, kdy k události je přiřazena metoda komponenty. Direktivu *v-on* lze nahradit v kódu ekvivalentním znakem "**@**" a příklad jejího použití je na ukázce kódu 2.2. Pro dvousměrný data binding se používá direktiva *v-model*, kterou lze využít například při řízení formulářů.

V rámci **props**, jakožto vstupních dat komponenty, lze vytvořit více proměnných, které mohou být nastaveny při volání komponenty a následně použity v šabloně či metodách. Props mohou být definovány buď jako pole, obsahující pouze názvy proměnných, nebo také jako objekt, kde lze nastavit proměnné datový typ a provést validaci. Příklad použití *props* je na ukázce kódu 2.4 a 2.5.

```
export default{
    props: ['name']
    print() {
        console.log(this.name)
    }
}
```

Listing 2.4: Použití props pomocí pole

```
export default{
    props: {
        name: String,
        age: Number
    }
    print() {
        console.log(this.name,this.age)
    }
}
```

Listing 2.5: Použití props pomocí objektu

2.3 React

React je front-end JavaScriptová knihovna pro tvorbu uživatelského rozhraní, která se používá jako základ při vývoji jednostránkových webových aplikací. Je založena na principu, kdy jsou navrženy pohledy pro různé stavy aplikace a při změně dat se zobrazí požadované části uživatelského rozhraní. Vyznačuje se svou rychlou odezvou na změny dat a uživatelské události [7].

React je v současnosti asi nejpopulárnějším frameworkem pro tvorbu webových aplikací a to z důvodu jeho jednoduchosti a výkonu. Aplikace tvořené pomocí React mají velice dobrý výkon díky použití virtuálního DOM. Virtuální DOM je technika, která umožňuje při změně stavu aplikace neaktualizovat přímo DOM, ale pouze jeho kopii virtuální DOM. Při změně dojde k porovnání mezi předchozím nezměněným a aktualizovaným virtuálním DOM a React provede potřebné změny pouze v těch částech skutečného DOM, které byly opravdu změněny, aniž by musela být překreslena celá stránka. Aktualizace virtuálního DOM je mnohem efektivnější a rychlejší, protože manipulace s DOM je nákladná operace a může mít vliv na výkon aplikace [8]. Díky svým vlastnostem má React rozsáhlou komunitu, což je výhoda pro vývojáře, kteří při problému mohou řešení hledat na fórech, používaných tisíce vývojáři. Mají díky tomu k dispozici rozsáhlé množství návodů, tutoriálů a také knihoven či nástrojů.

Mezi nevýhody React.js patří jeho časté aktualizace. Ty sice přináší nové funkce a vylepšení, avšak pro vývojáře může být náročné se pravidelně učit nové způsoby psaní kódu a přizpůsobovat se změnám. S tímto přichází také problém s dokumentací, která z důvodu častých aktualizací není tak obsáhlá a kvalitní jako například u Angularu. Protože React je knihovna, která slouží pouze k tvorbě uživatelského rozhraní, vzniká zde potřeba použití dalších knihoven a nástrojů, které budou řešit potřebnou funkcionalitu [9].

Základ aplikace Reactu je opět rozdělení do jednotlivých částí zvaných komponenty. Tyto komponenty jsou znovupoužitelné prvky uživatelského rozhraní a mohou být přidány do struktury DOM pomocí knihovny React DOM. Komponenty v Reactu mohou být vytvořeny pomocí funkcí nebo tříd. Komponenta vytvořená pomocí třídy obsahuje metodu *render()* vracející obsah v podobě HTML

```
class Greeter extends React.Component {
    state ={ name: "John" };
    render() {
      return <h1>{this.props.greeting} {this.state.name}</h1>
    }
  }

ReactDOM.render(<Greeter greeting="Hello World!"/>, document.getElementById('
    myReactApp'));
```

Listing 2.6: Použití komponenty vytvořené třídou

kódu, který se má po zavolání vykreslit. Jako vstupní data komponenty, tvořené třídou i funkcí, slouží tzv. **props**, které lze nastavit při vkládání komponenty. Tyto *props* slouží v komponentě pouze pro čtení a neměly by proto být měněny. Kromě *props* mohou v komponentě existovat proměnné označující se jako **state** neboli stav. Stav je ve třídě reprezentován jako objekt, jehož vlastnosti mohou být, na rozdíl od *props*, v komponentě změněny. V poslední verzi knihovny React již není doporučeno používat komponenty tvořené třídou, místo toho React doporučuje v kódu používat komponenty tvořené funkcí. Příklad komponenty vytvořené třídou je na ukázce kódu 2.6 [10].

Komponenty tvořené pomocí funkcí přijímají *props* jako parametr funkce a vrací stejný výsledek jako metoda *render()* v komponentě tvořené třídou. Rozdíl je zde také v definici a použití stavu komponenty. Příklad komponenty vytvořené funkcí je na ukázce kódu 2.7.

```
    export default function Welcome(props){

        const [age, setAge] = useState(0);
        function IncrementAge() {
            setAge(age + 1);
        }
        return <div>
        <button onClick={IncrementAge}>Increment age</button>
        <h1>Hello, {props.name} ({age})</h1></div>;
    }
```

Listing 2.7: Použití komponenty vytvořené funkcí [11]

Komponenty Reactu často využívají rozšíření jazyka JavaScript **JSX (JavaScript XML)**, které umožňuje psát HTML kód přímo do JavaScriptu a usnadňuje tím vytváření komponent. JSX lze využít právě jako návratová hodnota metody *render()* či komponenty tvořené funkcí [12]. Příklad použití JSX je vidět na ukázkách kódu 2.6 a 2.7.

2.4 Blazor

Blazor je framework pro vývoj webových aplikací, který na rozdíl od předešlých frameworků umožňuje vytvářet webové aplikace pomocí jazyka C#. Je vyvíjen společností Microsoft a má proto výhodu v použití funkcí a knihoven .NET. Pro vývojáře, kteří již mají zkušenosti s nástroji .NET a jazykem C# je toto velkou výhodou, protože mohou tvořit aplikace s plnou funkcionalitou, aniž by se museli učit jiný jazyk či použít jiný framework. Díky podpory Microsoftu má Blazor také kvalitní a rozsáhlou dokumentaci [13].

Blazor je založen na vytváření komponent, jakožto prvků uživatelského rozhraní, a nabízí několik možností jejich hostování jako například Blazor Server či Blazor WebAssembly. V případě Blazor server běží aplikace na straně serveru. Pro vývoj jednostránkových aplikací (SPA) je používán Blazor WebAssembly, který spouští Blazor aplikace na straně klienta v prohlížeči pomocí technologie .NET runtime založené na WebAssembly. V tomto modelu hostování je aplikace Blazoru, včetně jejích závislostí, spolu s .NET runtime stažena do prohlížeče a poté spuštěna. Blazor WebAssembly vyžaduje v prohlížeči podporu WebAssembly a nemusí být tedy kompatibilní s některými staršími prohlížeči [14].

Framework Blazor je poměrně nový a zatím méně známý. Nemá tedy tak rozsáhlou komunitu vývojářů a také kolekci knihoven jako například frameworky React.js či Angular. Může být proto obtížnější najít řešení některých specifických problémů. Jak bylo dříve zmíněno, Blazor vyžaduje znalost jazyka C# a ekosystému .NET, což může být překážkou pro vývojáře, kteří s těmito technologiemi doposud nepracovali.

Komponenty v Blazoru jsou tvořeny jako třídy jazyka C# a pro tvorbu uživatelského rozhraní se opět využívá jazyk HTML. Třídy komponenty jsou obvykle psány v souboru s příponou .razor, který umožňuje spojení HTML a C# kódu. Příklad Blazor komponenty lze vidět na ukázce kódu 2.8.

```
<div class="card" style="width:22rem">
        <h3 class="card-title">@Title</h3>
        <button @onclick="OnYes">Yes!</button>
</div>
@code {
    [Parameter]
    public string? Title { get; set; }
    private void OnYes()
    {
        Console.WriteLine("Write to the console in C#! 'Yes' button selected.");
    }
}
```

Listing 2.8: Blazor komponenta [13]

Součástí komponenty je tedy HTML kód, parametry komponenty a funkce, které většinou řeší reakce na události uživatelského rozhraní. Komponentu lze opět vložit do jiné komponenty, přičemž je zároveň možné nastavit hodnoty parametrů dané komponenty (viz. 2.9).

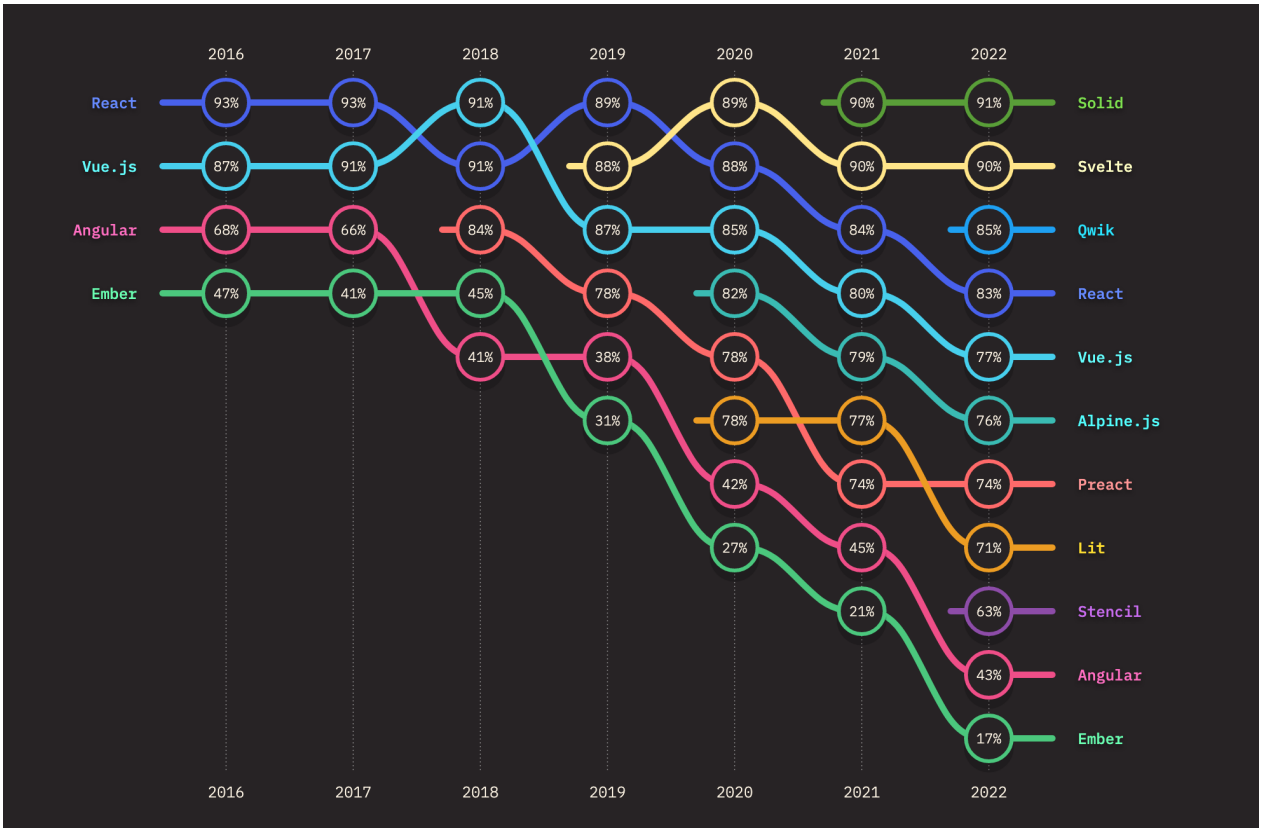
```
<p>
    Welcome to your new app.
</p>
<Dialog Title="Learn More">
    Do you want to lear more about Blazor?
</Dialog>
```

Listing 2.9: Vložení Blazor komponenty [13]

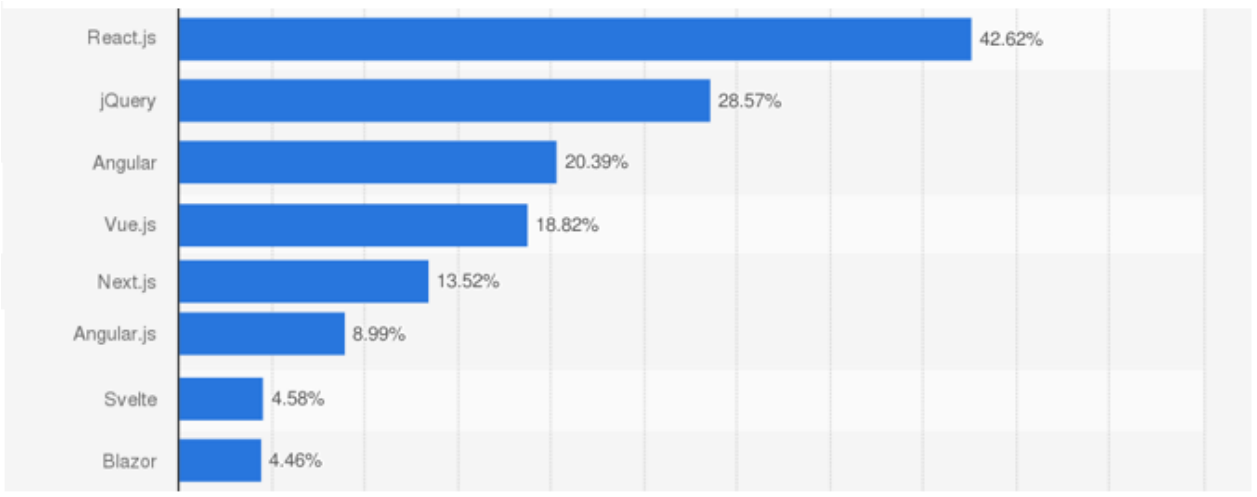
2.5 Výběr a porovnání frameworků

Cílem použití frameworků je zjednodušit a zefektivnit vývoj webových aplikací. Při výběru frameworku je důležité stanovit, pro jaké účely bude využit, protože různé frameworky mohou být vhodné na různé typy aplikací a nelze obecně říct, který framework je nejlepší. Angular může být dobrou volbou pro vývoj mobilních či rozsáhlejších aplikací. React.js a Vue.js se vyznačují svou rychlostí a mohou být vhodné, pokud vývojář vyžaduje po aplikaci vysoký výkon. Pro vývojáře, používající jazyk C# a nástroje .NET, bude pravděpodobně nejjednodušší volbou framework Blazor.

Na obrázku 2.1 lze vidět statistiku spokojenosti programátorů s některými JavaScriptovými front-end frameworky v průběhu let. Graf ukazuje procento programátorů, kteří již s daným frameworkem pracovali a použili by jej znovu. Dále obrázek 2.2 zobrazuje statistiku průzkumu používanosti jednotlivých webových frameworků mezi programátory v roce 2022.



Obrázek 2.1: Statistika spokojenosti při použití frameworků [15]



Obrázek 2.2: Použití frameworků v roce 2022 [16]

Kapitola 3

Vývoj Angular knihoven

Ve frameworku Angular existuje velké množství knihoven, které lze používat při vývoji aplikací, a které obsahují často používané komponenty, služby či direktivy. Tyto knihovny nemohou fungovat samy, ale musí být přidány do aplikace a poté použity. Vývojáři mohou vytvářet také vlastní knihovny, které pak použijí ve své aplikaci nebo je mohou sdílet pro ostatní vývojáře v komunitě Angular. V této kapitole je popsán vývoj knihoven a jejich komponent [17].

3.1 Angular Workspace

Pro vytváření, správu či spouštění Angular aplikací v příkazové řádce existuje nástroj Angular CLI. Angular CLI umožňuje vytvářet knihovny, komponenty, direktivy či služby, a zároveň generovat jejich strukturu a konfiguraci [18]. Tento nástroj slouží také k vytvoření tzv. **Angular Workspace**, díky kterému může být několik projektů uloženo v jednom adresářovém stromě a sdílet nastavení, konfiguraci a závislosti. Příkaz `ng new nazev-workspace` vytvoří nový Angular workspace a zároveň v něm vygeneruje následující adresářovou strukturu včetně výchozího projektu se stejným jménem [19].

- node_modules/ – adresář, který obsahuje balíčky potřebné pro běh aplikací, dostupný pro všechny projekty ve workspace.
- src/
- angular.json – soubor obsahující konfiguraci pro Angular CLI ke kompilaci a spouštění projektů ve workspace.
- package.json [20] – soubor obsahující seznam balíčků a závislostí, které jsou potřebné pro běh aplikací. Je využíván všemi projekty v daném workspace. Při vývoji aplikací jsou zde přidávány potřebné balíčky.

- README.md – soubor obsahující úvodní informace k výchozímu projektu.
- tsconfig.app.json – soubor obsahující konfiguraci pro TypeScript kompilaci, specifický k výchozímu projektu.
- tsconfig.json – soubor obsahující konfiguraci pro TypeScript kompilaci, používaný všemi projekty ve workspace.
- tsconfig.spec.json – soubor obsahující konfiguraci pro TypeScript, který specifikuje nastavení pro TypeScript kompilátor při kompilaci testů aplikace.

V adresáři **src** jsou umístěny zdrojové kódy výchozího projektu včetně podadresáře **app**. Tento podadresář obsahuje soubory výchozí komponenty, která je při spuštění aplikace zobrazena. Tato komponenta se chová jako kořenová a jiné komponenty, které mají být zobrazeny, do ní musí být vloženy. Struktura výchozího projektu je následující:

- app/
  - app.components.css – soubor obsahující CSS styly pro kořenovou komponentu.
  - app.component.html – Soubor obsahující HTML šablonu kořenové komponenty.
  - app.component.ts – soubor definující hlavní kořenovou komponentu aplikace.
  - app.module.ts – hlavní modul projektu, který definuje závislosti, komponenty a služby.
- index.html – soubor obsahující hlavní šablonu aplikace. Tato HTML stránka je zobrazena po spuštění aplikace a je zde automaticky vložena kořenová komponenta.
- main.ts – hlavní TypeScript soubor aplikace.
- styles.css – soubor obsahující globální CSS styly pro celou aplikaci.

3.2 Vytvoření Angular knihovny

Vytvoření nové Angular knihovny lze provést příkazem z Angular CLI `ng generate library nazev-knihovny`, kde jako aktuální adresář musí být zvolený Angular workspace. Po vytvoření knihovny vznikne v adresáři **projects** nový adresář s názvem knihovny, obsahující konfigurační soubory, výchozí modul a zdrojový kód hlavní komponenty.

3.3 Angular modul

V Angularu lze organizovat funkcionalitu souvisejících komponent, direktiv či služeb do bloků zvaných **moduly**, kdy každý projekt či knihovna má minimálně jeden. Tyto moduly mohou být importovány v jiných modulech a umožňují tak použití komponent knihovny v ostatních komponentách

či aplikaci. Moduly jsou vytvářeny pomocí třídy s dekorátorem **@NgModul**, obsahující následující parametry [21]:

- declarations – parametr obsahující pole komponent či direktiv, náležící danému modulu.
- imports – parametr definuje seznam jiných modulů, které mohou být použity v rámci daného modulu.
- exports – parametr obsahuje seznam komponent či direktiv, náležících danému modulu, které mohou být použity v jiných modulech.
- bootstrap – seznam kořenových komponent, které jsou inicializovány při spuštění aplikace.
- providers – seznam služeb použitelných v rámci modulu.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ MyComponent ],
  exports:      [ MyComponent ],
  bootstrap:    [ MyComponent ]
})
export class MyModule { }
```

Listing 3.1: Soubor Angular modul

3.4 Angular component

Knihovny v Angularu obsahují při jejich vytvoření jednu výchozí komponentu. V knihovnách však může být vytvořeno několik komponent, které se vzájemně používají. Vytvoření nové komponenty lze pomocí příkazu `ng generate component nazev-komponenty`, kdy po zadání příkazu se vygenerují následující soubory:

- my-component.component.css
- my-component.component.html
- my-component.component.spec.ts
- my-component.component.ts

Soubor *my-component.component.ts* obsahuje definici samotné komponenty a jeho součástí jsou importy knihoven, třída a dekorátor **@Component** označující třídu za komponentu (viz. 3.2). Dekorátor *@Component* má parametry, které definují nastavení komponenty. Parametr *selector* určuje pod jakým názvem bude komponenta vkládána do šablony jiné komponenty. Parametry *templateUrl* a *styleUrls* odkazují na soubory obsahující HTML šablonu a soubory s CSS styly komponenty (viz. 3.2). Šablona či styly CSS mohou být v komponentě, místo odkazu, nastaveny také přímo pomocí parametrů *template* a *styles* (viz. 3.3).

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-component',
  templateUrl: './my-component.component.html',
  styleUrls: ['./my-component.component.css']
})
export class MyComponentComponent {
}
```

Listing 3.2: Soubor Angular komponenty my-component.component.ts [22]

```
template: '<p>my-component works!</p>',
styles: ['p{ font-size:18px; }']
```

Listing 3.3: Nastavení šablony a stylů

Třída komponenty obsahuje metody a properties. Metody definují chování komponenty a jsou často využívány jako reakce na uživatelské události v HTML elementech. Properties reprezentují data komponenty a mohou být zobrazeny v HTML šabloně či upraveny a použity v metodách. Příklad properties a metod třídy je na ukázce kódu 3.4.

```
export class MyComponentComponent {
  name:string="Jan";
  active:boolean=false;

  SetActive(){
    this.active=true;
  }
}
```

Listing 3.4: Příklad properties a metod

3.4.1 Sdílení dat mezi komponentami

Ve chvíli, kdy je komponenta vložena do šablony jiné komponenty, se z jedné z nich stává komponenta potomka a z druhé komponenta rodiče. Pro komunikaci mezi nimi je nutné, aby bylo možné do komponenty potomka data zaslat a tato komponenta mohla data také vrátit. Pro vstup a výstup z komponenty se používají *properties* komponenty, před které je přidán dekorátor **@Input** a **@Output**.

Property, před kterou je umístěn dekorátor *@Input*, je označena jako vstupní a může mít různé datové typy jako *string*, *number*, *boolean* či *objekt*. Tato *property* může být nastavena rodičovskou komponentou při přidání komponenty do její šablony. Příklad předávání dat z rodičovské komponenty do komponenty potomka je na ukázkách 3.5 a 3.6.

Dekorátor *@Output*, který je přidán před název *property*, ji označí jako výstupní a umožňuje přenos dat do rodičovské komponenty. Tato *property* musí být instancí třídy EventEmitter, která slouží k vytváření a vysílání vlastních událostí. Při vytvoření instance třídy EventEmitter je zadán jako parametr datový typ dat, která budou při vysílání události přenášena. Pro zaslání dat rodičovské komponentě je na výstupní *property* zavolána funkce *emit* (viz. 3.5).

```
@Component({
  selector: 'my-component',
  template: `<input #myInput type="text" (keyup)="OnInputText(myInput.value)"/> `,
  styleUrls: ['./my-component.component.css']
})
export class MyComponentComponent {
  @Output() message=new EventEmitter<string>();
  @Input() textToShow:string="";
  OnInputText(messageText:string){
    this.message.emit(messageText);
  }
}
```

Listing 3.5: Komponenta potomka

Pro naslouchání vyslané události je potřeba v komponentě rodiče vytvořit metodu, která je přiřazena k výstupní property potomka. Příklad použití lze vidět na ukázce kódu 3.6, kde je metoda *returnMessage()* přiřazena k výstupní property a proměnná *$event* obsahuje data zaslaná z komponenty potomka.

```
@Component({
  selector: 'parent-component',
  template: `
  <my-component (message)="returnMessage($event)"></my-component>
```

```
  <my-component textToShow="Welcome!"></my-component>
  <my-component [textToShow]="actualText"></my-component>
    `,
  styleUrls: ['./parent-component.component.css']
})
export class ParentComponentComponent {
  messages:string[]=[];
  returnMessage(returnedMessage:string){
    this.messages.push(returnedMessage);
  }
}
```

Listing 3.6: Komponenta rodiče

3.4.2 Data binding

Jednou z nejdůležitějších vlastností SPA frameworků, včetně frameworku Angular, je tzv. data binding. Data binding se používá k propojení modelu komponenty (daty) s jejím pohledem (uživatelské rozhraní). Data modelu mohou být propojena s HTML elementem, komponentou potomka či direktivou rozšiřující HTML element. Toto propojení slouží pro detekci změn, což znamená, že změna v modelu se promítne do pohledu či naopak. V Angularu existuje několik typů data bindingu, které lze rozdělit na binding jednosměrný a dvousměrný.

3.4.2.1 Jednosměrný data binding

Jednosměrný binding umožňuje synchronizaci modelu s pohledem (uživatelské rozhraní) pouze s jednosměrným tokem dat. To znamená, že data změněná v modelu se detekují a zobrazí v pohledu, ale změny provedené v pohledu se v modelu již neprojeví. Jednosměrný binding může být však také v opačném směru, kdy na události vyvolané v pohledu reagují příslušné metody.

Jedním z typů jednosměrné vazby je **Interpolace**, která umožňuje zobrazit hodnotu property komponenty v její šabloně. Název property se pro zobrazení umístí do těla HTML elementu ve složených závorkách "{{}}" (viz. 3.7) a při změně property v modelu se změna projeví i v šabloně.

Dalším typem bindingu je **property binding**, který slouží k přiřazení hodnoty k vlastnosti HTML elementu či ke sdílení dat mezi komponentami. Umožňuje tedy dynamicky měnit hodnoty vlastností HTML elementů. Property binding se v šabloně označuje pomocí hranatých závorek "[ ]" a obsahuje dvě části, tedy název vlastnosti elementu a výraz, který se použije pro nastavení hodnoty. Příklad použití lze vidět na ukázce 3.7, kde je dynamicky měněna adresa obrázku v závislosti na property komponenty. Tento typ bindingu také umožňuje provádět matematické operace, řetězení či podmíněné výrazy.

Binding, kdy pohled naopak ovlivňuje model komponenty a tedy model umožňuje reagovat na uživatelské události HTML elementů v šabloně se nazývá **event binding**. Při události vyvolané operacemi s HTML elementy lze nastavit, jaká metoda komponenty se zavolá. Pro toto přiřazení slouží kulaté závorky "()", uvnitř kterých je název události. Použití lze vidět na ukázce kódu 3.7, kde se po kliknutí na tlačítko zavolá metoda komponenty *buttonClick()*. Tímto způsobem lze ovládat události jako například změna hodnoty textového pole či stisk klávesy.

```
@Component({
  selector: 'my-component',
  template: `
    <h1>{{message}}</h1>
    <img  [src]="person.img" >
    <button [disabled]="counter<10" (click)="buttonClick()">Přidat</button>
    <img [style.width.px]="20+imgSize">
    `,
  styleUrls: ['./my-component.component.css']
})
export class MyComponentComponent {
    message:string="Welcome";
    person:{name:string,img:string}={name:"Jan",img:"urlpath"}
    counter:number=0;
    imgSize:number=200;
    buttonClick(){
        this.counter++;
    }
}
```

Listing 3.7: Data binding

3.4.2.2 Dvousměrný binding

Dvousměrný binding v Angularu umožňuje synchronizovat data mezi dvěma komponentami nebo mezi modelem a pohledem komponenty v obou směrech. Tedy pokud se změní hodnota property komponenty, změna se projeví i v uživatelském rozhraní a naopak. Dvousměrný binding se v Angularu značí speciální syntaxí "[()]", kombinující event binding a property binding.

Pomocí dvousměrného bindingu lze synchronizovat komunikace mezi dvěma komponentami. Pro správné fungování je potřeba upravit komponentu potomka tak, aby obsahovala vstupní property s dekorátorem **@Input** a výstupní property typu EventEmitter s dekorátorem **@Output**. Propojení těchto dvou vlastností je v Angularu vyřešeno tak, že pokud má vstupní vlastnost název **proper-**

tyname, musí být výstupní vlastnost pojmenovaná **propertynameChange**. Toto pojmenování je pro správné fungování nezbytné. Použití lze vidět na ukázkách komponenty potomka 3.8 a komponenty rodiče 3.9. Po změně hodnoty vstupní property *counter* se pro detekci změny v rodičovské komponentě musí na výstupní property *counterChange* zavolat metoda **emit** s hodnotou změněné vstupní property. V tu chvíli rodičovská komponenta automaticky detekuje změnu a přepíše hodnotu vlastní přiřazené property *parentCounter*.

```
@Component({ ...
  template: `<button (click)="buttonClick()">Přidat</button>`, ...
})
export class MyComponentComponent {
  @Input() counter:number=0;
  @Output() counterChange=new EventEmitter<number>();

  buttonClick(){
    this.counter++;
    this.counterChange.emit(this.counter);
  }
}
```

Listing 3.8: Komponenta potomka při dvousměrném bindingu

```
@Component({ ...
  template: `
   <my-component [(counter)]="this.parentCounter"></my-component>
   <p>{{parentCounter}}</p>`, ...
})
export class ParentComponent {
  parentCounter:number=0;
}
```

Listing 3.9: Komponenta rodiče při dvousměrném bindingu

Dvousměrný binding může sloužit také pro řízení formulářů. Zde se pro obousměrný tok dat používá direktiva NgModel.

3.4.3 Direktivy

Direktivy jsou jednou z funkcí frameworku Angular sloužící k rozšíření chování HTML elementů. Umožňují přidávat a manipulovat s funkcionalitou HTML elementů nad rámec běžného HTML. V Angularu jsou 3 typy direktiv:

- Direktivy komponentové
- Direktivy atributové
- Direktivy strukturální

3.4.3.1 Atributové direktivy

Atributové direktivy mění chování a vzhled HTML elementů přidáváním atributů. V Angularu je k použití několik již vestavěných direktiv, ale direktivy může uživatel definovat i vlastní. Zde budou popsány nejpoužívanější vestavěné direktivy.

NgModel je direktiva, která byla již zmíněna v kapitole o dvousměrném bindingu. Používá se pro přidání dvousměrného bindingu do HTML elementu a slouží k řízení formulářů. Pro použití této direktivy je potřeba importovat Angular modul **FormsModule**, který slouží pro práci s formuláři a obsahuje několik direktiv pro vytváření či validaci formulářů. K propojení formulářového HTML elementu s property komponenty je direktiva *ngModel* umístěna do závorek pro dvousměrný data binding ("[()]"), ke kterým je přiřazena property. V ukázce kódu 3.10 je ve vstupním HTML elementu zobrazena hodnota property a lze jí v něm zároveň i změnit.

```
@Component({ ...
  template: `<input type="text" [(ngModel)]="name">
             <p>{{name}}</p>`, ...
})
export class MyComponentComponent {
  name:string="Jméno";
}
```

Listing 3.10: Dvousměrný binding použitím direktivy NgModel

Direktiva **NgClass** umožňuje dynamicky přidávat či odebírat třídy CSS v HTML elementech. Hlavní výhodou této direktivy je přiřazení třídy na základě podmíněného výrazu či nastavení více CSS tříd jednomu elementu najednou. Třída může být přiřazena elementu na základě hodnoty property komponenty datového typu `boolean` či výrazu vracející `true` nebo `false` (viz. 3.11).

Přiřazení více tříd jednomu elementu lze pomocí objektu obsahující názvy tříd a výrazy vracející true nebo false. Direktiva NgClass je použita na ukázce 3.11, kde se na základě hodnot property `isHighlighted` a `isError` nastaví příslušné třídy.

```
@Component({ ...
  template: `
    <p [ngClass]="{'highlight':isHighlighted,'error':isError}">{{message}}</p>
    <p [ngClass]="isHighlighted? 'highlight' : ''">{{message}}</p>
    <p [ngClass]="(counter>=limit)? 'error' : ''">{{counter}}</p>
    `, ...
})
export class MyComponentComponent {
  counter:number=1;
  limit:number=100;
  message:string="Welcome";
  isHighlighted:boolean=true;
  isError:boolean=false;
}
```

Listing 3.11: Použití direktivy NgClass

Pro nastavení stylů HTML elementu slouží také direktiva **NgStyle**, která funguje na stejném principu jako direktiva NgClass, ovšem namísto tříd se zde přiřazují jednotlivé CSS vlastnosti. Na ukázce 3.12 lze vidět použití direktivy NgStyle, kde barva textu elementu bude měněna na základě podmíněných výrazů.

```
@Component({
  ...
  template: `
    <p [ngStyle]="{ 'background-color': isHighlighted ? 'yellow' : 'white','color
       ':isError? 'red' : 'black' }">{{message}}</p>
    `, ...
})
export class MyComponentComponent {
  isHighlighted:boolean=true;
  isError:boolean=false;
  message:string="Welcome";
}
```

Listing 3.12: Použití direktivy NgStyle

3.4.3.2 Strukturální direktivy

Tento druh direktiv umožňuje manipulovat s DOM strukturou na základě dat v aplikaci. Pomocí hodnot properties komponenty lze přidávat, mazat či měnit HTML elementy.

Direktiva **NgIf** umožňuje vykreslovat HTML elementy na základě podmíněného výrazu. Pokud je tento výraz pravdivý, element se vykreslí, pokud ne, tento element bude odebrán z DOM struktury a nebude tedy zobrazen. V ukázce použití 3.13 je element <p> vykreslován na základě hodnoty property *showMessage*.

K zobrazení seznamu či pole dat komponenty v HTML šabloně slouží direktiva **NgFor**, která umožňuje iterovat přes property komponenty obsahující nějakou kolekci dat a vytvořit seznam HTML elementů pro jednotlivé prvky. Použití direktivy NgFor je vidět na ukázce 3.13, kde je direktiva vložena do elementu, který se bude opakovat na základě property obsahující pole prvků.

```
@Component({
  ...
  template: `
  <div *ngFor="let customer of customersList;let i=index;">
    {{i+1}}. Name: {{customer.name}} Age: {{customer.age}}
  </div>
  <p *ngIf="showMessage" >{{message}}</p>`,
  ...
})
export class MyComponentComponent {
    showMessage:boolean=true;
    message:string="Welcome";
    customersList=[{name:"Jan",age:22},
                   {name:"Martin",age:23},
                   {name:"Jiří",age:24},]
}
```

Listing 3.13: Použití direktivy NgFor

Direktiva **NgSwitch** slouží k vykreslení části HTML kódu na základě hodnoty proměnné či výrazu. Pomocí této direktivy lze nastavit několik možných výsledků (elementů či skupin elementů), kdy pouze jeden z nich bude přidán do DOM a vykreslen.

```
...
  <div [ngSwitch]="employee.salaryLevel">
    <div *ngSwitchCase="1" class="top-salary" > Top salary employee: {{employee.
        name}} </div>
```

```
    <div *ngSwitchCase="2" class="mid-salary" > Medium salary employee: {{employee
        .name}} </div>
    <div *ngSwitchCase="3" class="low-salary" > Low salary employee: {{employee.
        name}} </div>
    <div *ngSwitchDefault class="no-data" > Employee: {{employee.name}} </div>
  </div>
 ...
```

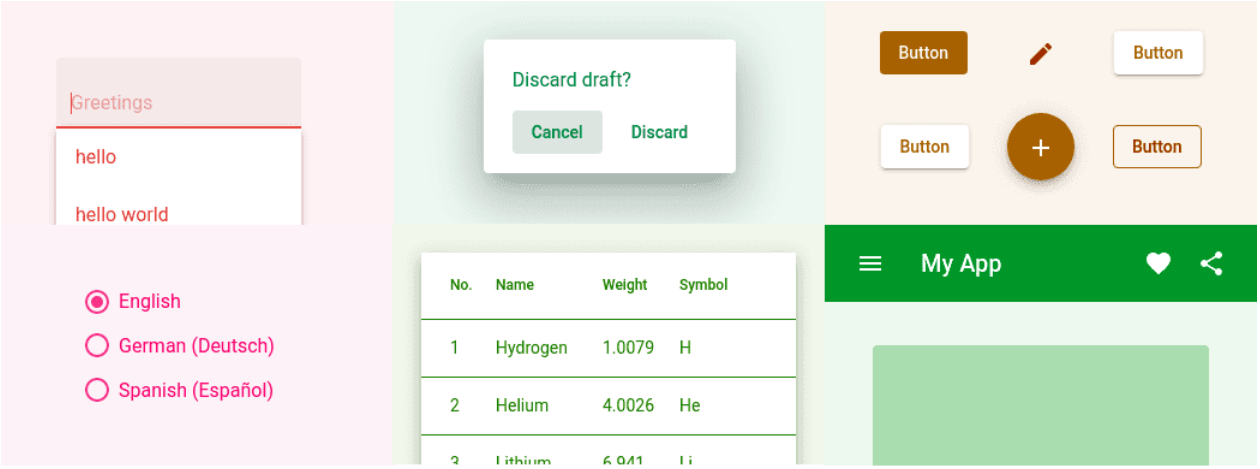
Listing 3.14: Použití direktivy NgSwitch

3.5 Angular Material

Výhodou při tvorbě komponent ve frameworku Angular je možnost použití již existujících knihoven, obsahující komponenty či direktivy, které usnadňují práci při řešení funkcionality a úpravě vzhledu uživatelského rozhraní. Pro tyto účely existuje knihovna **Angular Material** [23], která obsahuje velkou řadu komponent a direktiv reprezentující prvky uživatelského rozhraní jako jsou tlačítka, formuláře, dialogy, ikony a další. Tyto komponenty a direktivy rozšiřují klasické HTML prvky o vzhled a funkcionalitu. Příklad použití komponent knihovny Angular Material lze vidět na ukázce kódu 3.15.

```
<mat-toolbar-row>
    <mat-form-field>
        <input matInput placeholder="Vyhledat soubory" [(ngModel)]="searchText">
        <button *ngIf="searchText" matSuffix mat-icon-button>
            <mat-icon style="color:black;">close</mat-icon>
        </button>
    </mat-form-field>
</mat-toolbar-row>
```

Listing 3.15: Ukázka použití komponent Angular Material



Obrázek 3.1: Ukázka komponent Angular Material [23]

Součástí této knihovny je také sada nástrojů **CDK** (Component Dev Kit) poskytující řadu funkcionalit a utilit, které lze použít při tvorbě vlastních komponent či jako rozšíření již stávajících. Nástroje CDK slouží pouze k přidání funkcionality HTML elementům a nerozšiřují je tedy o vzhled jako v případě komponent Angular Material. Některé z nástrojů CDK jsou například:

- Drag and Drop – poskytuje direktivy a funkce k přidání Drag and Drop funkcionality HTML elementům.
- Menu – poskytuje direktivy k vytváření různých druhů menu či kontextových nabídek s možností konfigurace jako například nastavení pozice či orientace menu.
- Scrolling – nabízí několik komponent a direktiv, které slouží pro práci s posuvníkem stránky a především také komponentu, která zajišťuje, že při zobrazení velkého množství dat budou vykresleny pouze prvky, které se v jednu chvíli vejdou na stránku a ostatní data budou vykreslovány v reakci na posun stránky.

Kapitola 4

Návrh a implementace vzorových komponent

V rámci této práce bylo vytvořeno několik vzorových komponent demonstrujících způsob tvorby knihoven a komponent v rámci frameworku Angular. Tyto komponenty slouží jako alternativa k již existujícím komponentám třetích stran. Komponenty byly vytvořeny tak, aby byly obecně dostupné a funkční v různých aplikacích a využívající libovolné API.

4.1 Vizualizace výškového profilu

Angular knihovna, řešící vizualizaci výškového profilu pro geografická data trasy, vznikla jako alternativa k podobným řešením této problematiky, které lze vidět na známých webových či mobilních aplikacích jako jsou Mapy.cz [24], Google mapy [25] a další. Vizualizace výškového profilu pomocí této knihovny se od těchto rešení v některých aspektech liší a zároveň přidává některé nové funkcionality.

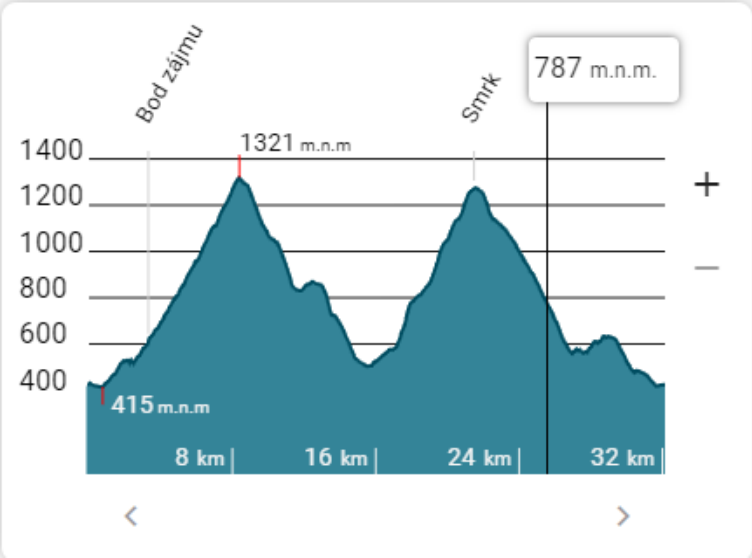
Pro použití vizualizace výškového profilu v různých aplikacích byly v této knihovně vytvořeny dvě Angular komponenty. Hlavní komponenta s názvem *elevation-profile* ve svém těle obsahuje komponentu *elevation-canvas*, která řeší samotné vykreslení výškového profilu na základě geografických dat, a její rodičovská komponenta implementuje ovládání pomocí tlačítek.

Aby byla hlavní komponenta obecně použitelná, musí být při jejím použití zadány vstupní *properties*. Tyto *properties* určují její vzhled, velikost a především data, která budou zpracovávána a zobrazena. Pro reprezentaci geografických dat existuje několik formátů. Komponenta tato data ukládá v podobě pole objektů uživatelsky definovaného typu s názvem *ElevationData* a pro správné fungování komponenty je potřeba data různých formátů převést právě na tento formát. Typ *ElevationData* má následující vlastnosti:

- elevation – vlastnost definující nadmořskou výšku aktuálního bodu v metrech.

- importantpoint – nepovinná vlastnost obsahující název bodu zájmu (město, památka či atrakce) v aktuálním bodě.
- id – vlastnost, která bodu přiřazuje vygenerovaný identifikátor.
- lat a lng – vlastnosti určující zeměpisné souřadnice: zeměpisná šířka a zeměpisná délka.

Pro vykreslení výškového profilu byl použit HTML element *canvas* [26], který slouží pro vykreslování geometrických tvarů, textu či obrázků na virtuální plátno pomocí JavaScriptu. Na plátno jsou vykreslovány tvary pomocí objektu JavaScriptu *CanvasRenderingContext2D* [27], který obsahuje metody jako *lineTo()*, *moveTo()*, *fill()*, *stroke()*, *fillText()* pro pohyb na plátně, vykreslení čar, vykreslení výplně plochy či vypsání textu. Výsledek komponenty, který lze vidět na obrázku 4.2, byl vykreslen pomocí několika metod, kde každá vykresluje jednotlivé části.



Obrázek 4.1: Ukázka komponenty Elevation profile

Nejdůležitější část je vykreslení samotného výškového profilu, čehož bylo docíleno vykreslováním čar pomocí funkce *lineTo()* na základě vstupních dat. Pro zlepšení vzhledu byl prostor pod výslednou křivkou vybarven pomocí funkce *fill()*, která umožňuje vybarvit obrazce či uzavřený prostor tvořený čarami.

Pro zlepšení orientace ve výškovém profilu lze vidět několik popisků, určující nadmořskou výšku či vzdálenost od začátku trasy. Popisky spolu s přímkami, určující nadmořskou výšku, jsou zobrazeny v několika úrovních. Počet těchto úrovní a intervaly mezi nimi jsou vypočteny na základě rozdílu mezi maximální a minimální nadmořskou výškou. Velikost intervalu byla určena pomocí výpočtů na ukázce 4.1, kde proměnná *count* určuje počet úrovní nadmořské výšky.

```
    diff = maxElevation - minElevation;
    tmp = diff / count;
    numCount = Math.round(tmp).toString().length;
    exp = Math.pow(10, numCount);
    interval= Math.ceil(tmp / exp * 5) / 5 * exp;
```

Listing 4.1: Výpočet intervalu mezi úrovněmi nadmořské výšky

Jsou zde také 4 popisky určující vzdálenost od začátku trasy. Jednotlivé vzdálenosti jsou zaokrouhleny na přehledná čísla a jsou tedy od sebe vzdáleny v téměř stejných intervalech. Zobrazeny jsou i hodnoty maximální a minimální nadmořské výšky, které mohou být v závislosti na nastavení vstupu komponenty skryty. Pokud vstupní data obsahují nějaké body zájmu, budou ve výsledku také vykresleny.

Po vykreslení výškového profilu může uživatel s komponentou interagovat pomocí myši či tlačítek komponenty. Při pohybu myši na výškovém profilu je zobrazena nadmořská výška v bodě odpovídajícímu pozici myši a zároveň jsou, jako výstup komponenty, vráceny informace o tomto bodě pro interakci s rodičovskou komponentou. S výškovým profilem lze také manipulovat přiblížením či posunem na konkrétní části trasy. Při posunu a přiblížení se vstupní geografická data zkrátí na pouhou část a jedině ta je na canvas v jednu chvíli vykreslena. Přiblížení lze provést buď pohybem kolečkem myši nebo tlačítky. Pro posun může uživatel opět využít tlačítka nebo také operaci Drag and Drop. Nové pole geografických dat, určující pouze část trasy, je vytvořeno v závislosti na hodnotě přiblížení a posunu, což lze vidět na ukázce 4.2, kde *sizeStat* a *positionStat* jsou právě tyto hodnoty.

```
newData: ElevationData[] = []
lastPoint=this.elevationData[this.elevationData.length - 1].distanceFromStart
startDistance = Math.floor(lastpoint * (this.sizeStat + this.positionStat));
endDistance = Math.floor(lastpoint * (1 - (this.sizeStat - this.positionStat)));
startIndex = this.elevationData.findIndex(x => x.distanceFromStart>startDistance);
endIndex = this.elevationData.findIndex(x => x.distanceFromStart>endDistance);

for (let i = startIndex; i < endIndex; i++) {
  newData.push(this.elevationData[i]);
}
```

Listing 4.2: Výpočet přiblížení a posunu

4.2 Plánování trasy

Knihovna s názvem **route-planning** byla vytvořena za účelem poskytnout nástroj pro jednoduché a uživatelsky přívětivé plánování tras. Součástí této knihovny jsou dvě komponenty. Tou hlavní je komponenta s názvem *route-planning*, která obsahuje seznam bodů trasy a nastavení v podobě typu přepravy (auto, autobus, kolo atd.). Jednotlivé body trasy jsou zadávány pomocí komponenty pro výběr místa *place-input*. Aby bylo komponenty možno použít obecně, mají jako vstup funkci libovolného API vracející seznam míst, které se zobrazí jako našeptávání při zadávání jednotlivých bodů trasy.

Komponenta pro výběr místa obsahuje jeden vstupní element, do kterého je zadáván bod trasy a který při zadávání zobrazuje našeptávání seznamu míst. Pro zobrazení nabídky seznamu míst a automatické vyplňování vstupního elementu byla použita komponenta **MatAutocomplete** z knihovny Angular Material [23]. Uživatel může kromě výběru z nabízených možností zadat na vstup také zeměpisné souřadnice míst v různých formátech. Pro zpracování souřadnic byl použit balíček npm **geo-coordinates-parser** [28], který zpracovává souřadnice v různých formátech a vrací hodnotu zeměpisné šířky a zeměpisné délky. Příklad formátů zeměpisných souřadnic:

- N 49°50.07872', E 18°16.92265'
- 49°50'4.723"N, 18°16'55.359"E
- 49.8346453N, 18.2820442E

Výstupem této komponenty je objekt reprezentující bod na trase a obsahující následující vlastnosti:

- name - Název místa
- lat - Zeměpisná šířka místa
- lng - Zeměpisná délka místa

Uživatel pracující s komponentou může při plánování trasy přidávat či odebírat jednotlivá místa. Místa lze mezi sebou také řadit či prohodit pomocí operace v uživatelském rozhraní zvané "Drag and Drop". Je tedy umožněno "přetáhnout" jedno místo na pozici druhého a tím změnit pořadí. Pro řešení funkcionality Drag and Drop byla použita sada nástrojů **Angular Material CDK** [29]. Konkrétně pro tuto potřebu CDK nabízí knihovnu **Drag and Drop**. Tato knihovna obsahuje direktivy, které jsou přidány do HTML elementů, a které umožňují jejich přesouvání. Pro vytvoření seznamu elementů, které lze mezi sebou přesouvat, byla z knihovny Drag and Drop použita direktiva *cdkDropList*. Součástí elementu obsahující direktivu *cdkDropList* je událost *cdkDropListDropped*, která je vyvolána po dokončení Drag and Drop operace mezi prvky seznamu. V metodě reagující na tuto událost lze poté zavolat metodu knihovny CDK Drag and Drop *moveItemInArray()*, která

aktualizuje seznam míst v komponentě tak, aby odpovídal novému seřazení seznamu v uživatelském rozhraní. Příklad použití knihovny Drag and Drop je na ukázce 4.3.

```
<div cdkDropList class="example-list" (cdkDropListDropped)="drop($event)">
  <div *ngFor="let point of travelInfo.points;index as i" cdkDrag>
   <div class="example-handle" cdkDragHandle> </div>
   <div class="example-custom-placeholder" *cdkDragPlaceholder></div>
   <place-input [searchPlaces]="searchPlaces" (pointResult)="result($event,point)"
      (deletePoint)="deletePoint($event,i)">
   </place-input>
  </div>
</div>
```

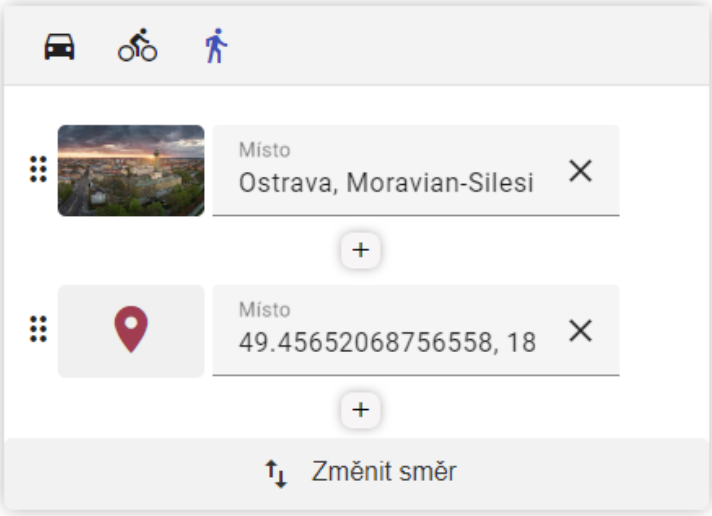
Listing 4.3: Použití knihovny CDK/Drag and Drop

4.2.1 Mapbox Search API

Komponenta *route-planning* umožňuje napojení na různé zdroje dat. Pro demonstraci funkčnosti této komponenty bylo ve vzorové aplikaci provedeno napojení na zdroj dat v podobě služby Mapbox Search API [30]. Pomocí Mapbox Search API lze vyhledávat místa na základě kritérií jako jsou názvy měst, názvy ulic či poštovní směrovací čísla. Výsledek hledání lze získat v podobě seznamu míst obsahující vlastnosti jako název, popis, adresu či zeměpisné souřadnice. Pro získání seznamu míst byl vytvořen dotaz na danou službu pomocí HTTP požadavku s parametrem vyhledávaného textu a jako výsledek byl vrácen seznam v JSON formátu.

```
async findMapbox(text:string) {
    const query =await fetch(
     `https://api.mapbox.com/geocoding/v5/mapbox.places/${text}.json?proximity=ip&
        types=place%2Cpostcode%2Caddress&access_token=pk.
        eyJ1IjoiaG9uemFzbG9ib2RuaWsiLCJhIjoiY2xjeHNsbGtoMDR2dDNwcGQ3NWtqbHNsdiJ9.
        nP2W6e90aoIwZyblOq-TXg`,
      { method: 'GET' }
    );
    const json = await query.json();
  }
```

Listing 4.4: Použití Mapbox Search API



Obrázek 4.2: Ukázka komponenty Route planning

4.3 Správce souborů

Knihovna s názvem **file-manager** vznikla za účelem vytvořit uživatelské rozhraní zobrazující soubory a adresáře, které umožňuje uživateli provádět s nimi typické operace. Knihovna obsahuje komponentu *file-manager*, která byla navržena tak, aby umožňovala připojit libovolné API, které dokáže vrátit data z nějakého úložiště a obsahuje funkce typické pro správu souborů jako přidávání, úprava či mazání souborů. Toto API je do komponenty posláno jako vstupní objekt obsahující metody, které jsou v komponentě využívány.

Třída, jejíž instance je posílána do komponenty, musí obsahovat následující metody:

- `renameFile(fileId: string, newTitle: string)` – asynchronní metoda, která provede přejmenování souboru s identifikátorem *fileId* na nový název *newTitle*.
- `moveFileToFolder(fileId:string,folderId:string)` – asynchronní metoda, která provede přesun souboru či adresáře s identifikátorem *fileId* do adresáře s identifikátorem *folderId*.
- `CreateFolder(folderName: string, parentID?: string)` – asynchronní metoda, která vytváří nový adresář se jménem *folderName* uvnitř adresáře s identifikátorem *parentID*.
- `fileDoubleClick(file: FileData)` – asynchronní metoda, která reaguje na uživatelskou událost dvojklik u souboru.
- `deleteFile(file: FileData)` – asynchronní metoda, která provede odstranění souboru z úložiště.

- `listFiles(sortbyKey: string, sortDirection: boolean, folderId: string):FileData[]` – asynchronní metoda vracející seřazený seznam souborů ze specifikovaného adresáře, jejíž návratová hodnota musí být pole typu *FileData*.

- `search(text: string, folderId?: string):FileData[]` – asynchronní metoda vracející seznam souborů, které jsou vybrány na základě hodnoty pro vyhledávání *text*. Pokud je zadán parametr *folderId*, budou soubory vyhledávány pouze v adresáři s tímto identifikátorem.

- `Download(file: FileData)` – asynchronní metoda, která provede stažení daného souboru do zařízení uživatele.

- `upload(file: FileData)` – asynchronní metoda, která provede nahrání souboru, jehož obsah je zde zadán ve formátu BLOB, do úložiště.

- `getSortByItems():{name:string,key:string}` – metoda vracející seznam objektů, obsahujících klíč a název vlastnosti souborů, pomocí kterých lze soubory v komponentě seřadit.

Komponenta ukládá a pracuje se soubory, které jsou reprezentovány pomocí typu *FileData*. Tento typ se často vyskytuje v metodách připojeného API, buď jako parametr nebo jako návratová hodnota. Objekt typu FileData má následující vlastnosti:

- id – vlastnost typu string definující identifikátor souboru.

- name – vlastnost typu string obsahující název souboru.

- fileType – vlastnost typu string určující typ souboru. (obrázek,dokument,složka atd.)

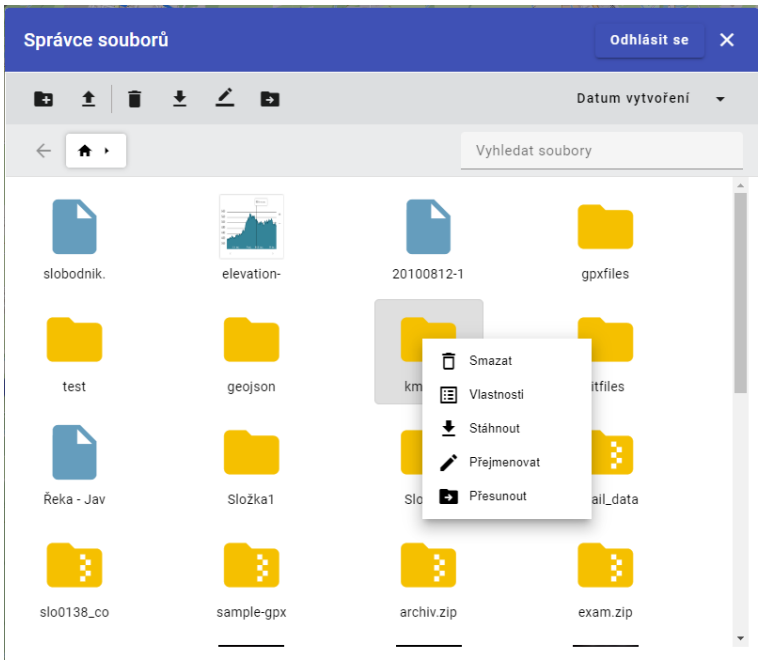
- img – nepovinná vlastnost obsahující adresu obrázku, který bude v komponentě pro daný soubor zobrazen.

- normalFolder – vlastnost typu boolean, která rozlišuje zda je soubor adresářem.

- zipFolder – vlastnost typu boolean, která rozlišuje zda je soubor ZIP archiv.

- blobContent nepovinná vlastnost typu Blob, která obsahuje obsah souboru ve formátu BLOB (Binary Large OBject).

- parentFolder – vlastnost typu string obsahující identifikátor adresáře, ve kterém je soubor umístěn.



Obrázek 4.3: Ukázka komponenty File manager

4.3.1 Google Drive API

Komponenta *file-manager* byla ve vzorové aplikaci napojena na službu Google Drive API [31], která umožňuje přistupovat k úložišti dat Google Disk (*angl. Google Drive*). Funkce Google Drive API umožňují pracovat se soubory uloženými na Google Disku pomocí zasílání HTTP požadavků. API bylo v aplikaci použito pomocí Google knihovny GAPI. Google Drive API obsahuje několik funkcí pro práci se soubory na Google Disku jako například *create*, *list*, *delete* či *update*. Příklad použití Google Drive API lze vidět na ukázce kódu 4.5, kde je zaslán požadavek na API a jako odpověď je vrácen JSON objekt obsahující seznam souborů.

```
let response;
try {
    response = await gapi.client.drive.files.list({
      fields: 'files(id, name,parents, mimeType, modifiedTime,size)'}
    );
} catch (err: any) {
    return;
}
var files = response.result.files;
```

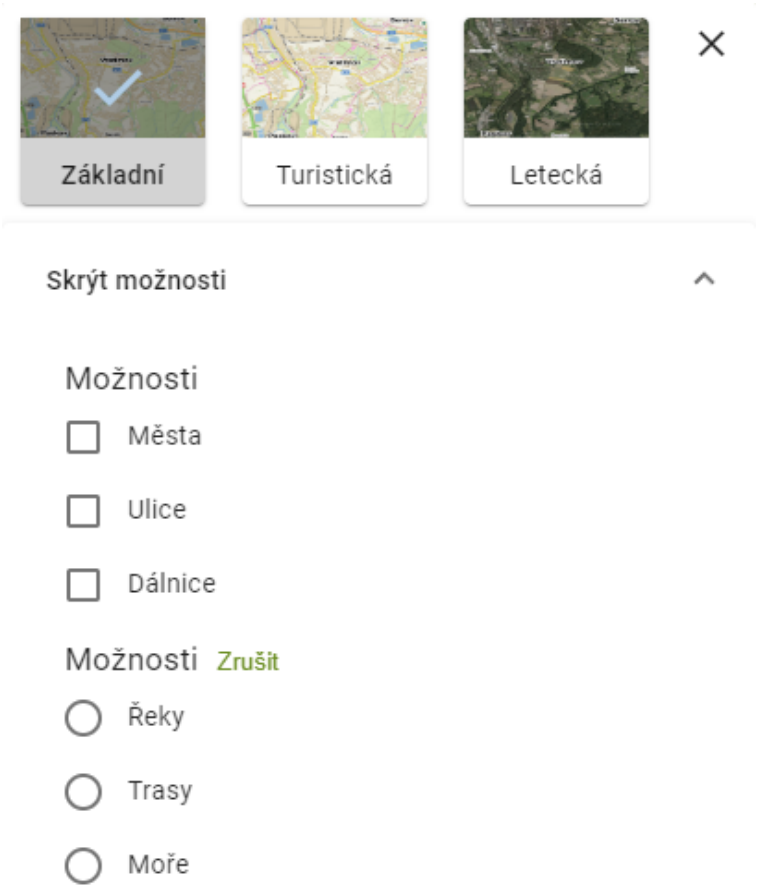
Listing 4.5: Použití Google Drive API

4.4 Správa vrstev

Knihovna **map-layers** vznikla za účelem poskytnout komponentu, jejíž výsledkem je přehledné uživatelské rozhraní pro výběr vrstev map. Tato komponenta byla vytvořena tak, aby mohla být obecně použitelná pro všechny potřeby uživatele. Vstupem komponenty je objekt obsahující informace o tom, jaké vrstvy lze zvolit a jakým způsobem mají být prezentovány. Výstupem jsou pak informace o uživatelem zvolených vrstvách.

Objekt, který je posílán do komponenty jako vstup, obsahuje následující vlastnosti:

- `mapType : {key:string, name:string, imgPath:string}[]` – vlastnost definující seznam typů mapy kde jeden z nich může být uživatelem, který používá komponentu, zvolen. Každý typ mapy je definovaný názvem, adresou obrázku, který bude v komponentě zobrazen, a klíčem, který bude z komponenty při výstupu vrácen.
- `optionGroups :{ options: { key: string, name: string}[], type: string, groupName : string}[]` – vlastnost, která umožňuje rozdělit výběr vrstev mapy do skupin, kde každá skupina obsahuje dále vlastnosti:
  - `options` – seznam vrstev určenými klíčem a názvem.
  - `type` - vlastnost definující zda může být vybrána jedna nebo více vrstev z této skupiny.
  - `groupName` – název skupiny vrstev.



Obrázek 4.4: Ukázka komponenty Map layers

4.5 Import tras v různých formátech geografických dat

Geografická data mohou být uložena v souborech několika různých formátů jako například GPX, GeoJSON, KML a další. Vzorová aplikace a komponenty, ze kterých se skládá, využívají geografická data především ve formátu GeoJSON. Aby bylo možné do aplikace importovat také data jiných formátů, vznikla knihovna **Geodataconverter**, která obsahuje funkce pro převod těchto formátů. V této kapitole budou popsány jednotlivé formáty a převod mezi nimi.

4.5.1 GeoJSON

GeoJSON [32] je formát pro ukládání geografických dat v JSON formátu. Umožňuje reprezentovat různé typy geografických prvků jako jsou linie, body či polygony a také jejich vlastnosti. GeoJSON soubor se skládá z tzv. *features*, kde každá reprezentuje geografický prvek a obsahuje vlastnosti jako název, popis, typ a především souřadnice. Příklad souboru GeoJSON je na ukázce 4.6.

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {
                "name": "Můj bod",
                "description": "Tohle je popis mého bodu."
            },
            "geometry": {
                "type": "Point",
                "coordinates": [14.4216941, 50.0878114 ]
            }
        },
        {
            "type": "Feature",
            "properties": {
                "name": "Můj polygon",
                "description": "Tohle je popis mého polygonu."
            },
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [ 14.4214646, 50.0878813 ],[ 14.4213339, 50.0877896 ],[
                            14.4214646, 50.0878813 ]
                    ]
                ]
            }
        }
    ]
}
```

Listing 4.6: Soubor GeoJSON

4.5.2 GPX

GPX [33] je formát, který slouží k ukládání geografických dat pro trasy a obsahuje především informace o zeměpisné šířce, zeměpisné délce či nadmořské výšce. GPX soubor je typ XML dokumentu, obsahující elementy, které určují trasy, body, názvy či čas. Příklad obsahu jednoduchého GPX souboru lze vidět na ukázce 4.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx version="1.1" creator="My GPS Device">
  <metadata>
    <name>My GPX Track</name>
    <desc>Track of my outdoor activity</desc>
  </metadata>
  <trk>
    <name>My Track</name>
    <trkseg>
      <trkpt lat="37.12345" lon="-122.98765">
        <ele>500</ele>
        <time>2023-03-25T09:30:00Z</time>
      </trkpt>
      <trkpt lat="37.12346" lon="-122.98766">
        <ele>505</ele>
        <time>2023-03-25T09:31:00Z</time>
      </trkpt>
      <trkpt lat="37.12347" lon="-122.98767">
        <ele>510</ele>
        <time>2023-03-25T09:32:00Z</time>
      </trkpt>
    </trkseg>
  </trk>
</gpx>
```

Listing 4.7: Soubor GPX

4.5.3 Garmin FIT

Garmin FIT [34] je formát souborů používaný pro ukládání dat, které jsou generovány zařízeními společnosti Garmin, určených převážně pro sportovní aktivity, jako například hodinky. Obsahuje informace o poloze, trasách, čase a také fyzické aktivitě jako rychlost, srdeční tep či spalování

kalorií. Tyto soubory mohou být sdíleny mezi dalšími zařízeními a zobrazovány v mobilním telefonu či počítači. Data jsou v souboru ukládány binárně.

4.5.4 KML

KML (Keyhole Markup Language) [35] je formát pro ukládání geografických dat. KML soubory jsou XML dokumenty, které obsahují elementy pro popis různých geografických prvků. V souboru jsou jednotlivé prvky označovány elementem *Placemarks*, kde každý takový element může obsahovat různé typy geometrických prvků jako:

- <Point> - bod
- <LineString> - linie
- <Polygon> - polygon
- <MultiGeometry> - element kombinující několik různých geometrických prvků

Tyto geometrické prvky jsou tvořeny seznamem souřadnic v podobě zeměpisné šířky, zeměpisné délky a nadmořské výšky. Příklad KML souboru je na ukázce 4.8.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Placemarks</name>
    <Placemark>
      <name>Place 1</name>
      <description>This is placemark 1.</description>
      <Point>
        <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <name> Place 2</name>
      <description>This is placemark 2.</description>
       <Polygon>
           <outerBoundaryIs>
             <LinearRing>
               <coordinates> -112.3372510731295,36.14888505105317,1784
                 -112.3356128688403,36.14781540589019,1784
                 -112.3372510731295,36.14888505105317,1784
               </coordinates>
```

```
            </LinearRing>
          </outerBoundaryIs>
        </Polygon>
    </Placemark>
  </Document>
</kml>
```

Listing 4.8: Soubor KML

4.5.5 Převod formátů GPX, KML a GeoJSON

Formáty geografických dat GPX a KML jsou oba typy XML dokumentu. Pro převod těchto formátů na GeoJSON bylo použito rozhraní DOMParser [36], které umožňuje převést XML soubor v textové podobě na tzv. DOM Document, pomocí kterého lze modifikovat a číst jeho části. Při získávání hodnot z GPX souboru byl postupně skládán JavaScript objekt, který je následně pomocí metod JavaScript objektu JSON převeden na validní JSON formát. Příklad převodu části GPX souboru pomocí DOMParser na GeoJSON je na ukázce kódu 4.9.

```
var xmlDoc = parser.parseFromString(gpx, 'text/xml');
var jsonObject: { type: string, features: any[], info: string } = {
    "type": "FeatureCollection",
    "features": [],
    "info": "gpx to geoJSON converted!"
}
var points = xmlDoc.getElementsByTagName('wpt');
for (let i = 0; i < points.length; i++) {
    var feature:{ type: string, geometry:{ type: string, coordinates: number[]}}={
        "type": "Feature",
        "geometry": {"type": "Point","coordinates": [] }
    };
    var latitude = parseFloat(points[i].getAttribute('lat'));
    var longitude = parseFloat(points[i].getAttribute('lon'));
    feature.geometry.coordinates.push(longitude);
    feature.geometry.coordinates.push(latitude);
    jsonObject.features.push(feature);
    var geojson=JSON.parse(JSON.stringify(jsonObject));
}
```

Listing 4.9: Převod GPX na GeoJSON

Převod formátu GeoJSON na GPX byl realizován pomocí knihovny GPX builder, která slouží k vytváření GPX souborů.

4.5.6 Převod formátů Garmin FIT a GeoJSON

K převodu souboru ve formátu FIT byl použit balíček *fit-file-parser*, který umožňuje převádět binární data FIT souboru na objekt, který poté lze převést na JSON. Z tohoto JSON objektu byly následně čteny data a postupně přidávány do výsledného objektu ve formátu GeoJSON. Příklad použití knihovny *fit-file-parser* je na ukázce 4.10.

```
var fitParser = new FitParser({
      force: true,
      speedUnit: 'km/h',
      lengthUnit: 'km',
      temperatureUnit: 'celsius',
      elapsedRecordField: true,
      mode: 'both',
    });

fitParser.parse(fit_file, function (error: any, data: any) {
    if (error) {
        console.log(error);
    }else {
        var json = JSON.parse(JSON.stringify(data));
    }
});
```

Listing 4.10: Použití balíčku fit-file-parser

Kapitola 5

Vzorová aplikace

Pro demonstraci funkcionality jednotlivých knihoven byla vytvořena aplikace, která využívá komponenty těchto knihoven a zároveň ukazuje možnost, jak mohou společně tvořit jeden celek zaměřený na práci s mapou. V aplikaci jsou použity všechny komponenty dříve popsané v kapitole 4, a tedy vizualizace výškového profilu, plánování tras, správce vrstev, správce souborů a pro převod formátů také třída knihovny GeoDataConverter.

Protože většina vytvořených komponent je zaměřena na práci s mapou, byla do aplikace přidána mapa pomocí JavaScript knihovny **Leaflet** [37]. Tato knihovna umožňuje nejen přidání samotné mapy, ale také vykreslování geometrických tvarů a ikon, vytváření vrstev či nastavení vzhledu mapy.

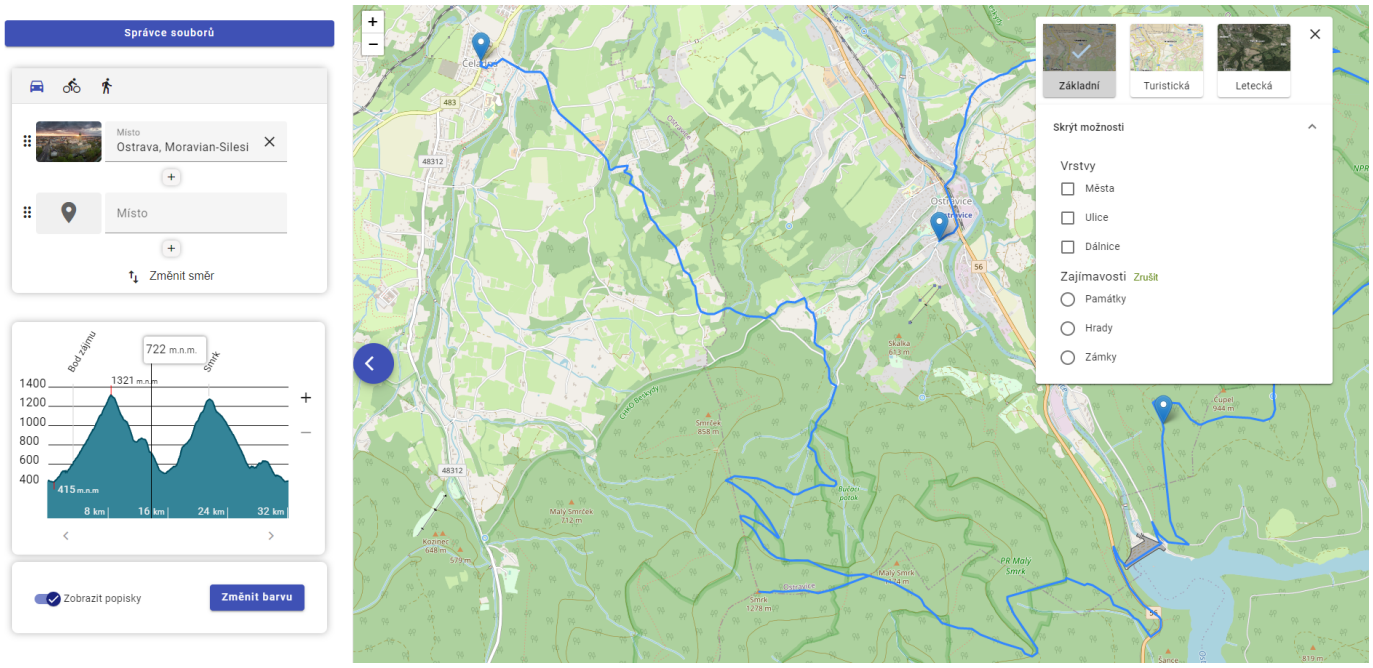
Vzhled a vrstvy mapy jsou v aplikaci měněny na základě výstupu komponenty pro správu vrstev.

Pomocí komponenty pro správu souborů lze do aplikace nahrát soubor obsahující geografická data ve formátech zmíněných v kapitole 4.5. Vybraný soubor je nejprve převeden na formát GeoJSON a poté jsou jeho data předána pro vykreslení výškového profilu a funkci knihovny Leaflet, která umožňuje přímé vykreslení tras a bodů ze souboru GeoJSON do mapy. Po vykreslení trasy ze souboru je při interakci uživatele s komponentou pro vizualizaci výškového profilu, zobrazením nadmořské výšky na pozici myši, vrácena z komponenty informace o daném bodě. Součástí této informace jsou hodnoty zeměpisné délky a šířky, díky kterým je do mapy vykreslována ikona na odpovídající pozici. Uživatel aplikace tedy může pozorovat stoupání či klesání v jednotlivých částech trasy na mapě.

Pro vykreslení trasy, naplánované pomocí komponenty pro plánování tras, do mapy byla použita služba **Mapbox Directions API** [38]. Tato služba umožňuje vypočítat celou trasu ze dvou či více zadaných bodů a vrátit ji ve formátu GeoJSON. Jako vstup pro tuto službu může být také nastavení typu přepravy, které je spolu s body trasy součástí výstupu komponenty pro plánování tras. Trasa ve formátu GeoJSON je získána zasláním HTTP požadavku s příslušnými parametry na dané API. Příklad požadavku na získání trasy je na ukázce 5.1, kde proměnná `coordinates` obsahuje souřadnice zadaných bodů trasy a proměnná `transport` definuje typ přepravy.

```
var points: string = ``;
    for (let i = 0; i < coordinates.length; i++) {
        points += coordinates[i][1].toString();
        points += ",";
        points += coordinates[i][0].toString();
    }
const query = await fetch(
      ` https://api.mapbox.com/directions/v5/mapbox/${transport}/${points}?
        alternatives=true&geometries=geojson&language=en&overview=full&steps=true
        &access_token=pk.
        eyJ1IjoiaG9uemFzbG9ib2RuaWsiLCJhIjoiY2xjeHNsbGtoMDR2dDNwcGQ3NWtqbHNsdiJ9.
        nP2W6e90aoIwZyblOq-TXg`,
      { method: 'GET' }
    );
const json = await query.json();
```

Listing 5.1: Soubor GeoJSON



Obrázek 5.1: Ukázka vzorové aplikace

Kapitola 6

Závěr

V rámci této práce vzniklo pět knihoven pro framework Angular a vzorová aplikace demonstrující jejich použití. Vzniklé knihovny, které slouží k práci s mapou a geografickými daty, obsahují komponenty pro plánování trasy, vykreslení výškového profilu a správu vrstev. Pro práci s různými formáty geografických dat, jako GPX, GeoJSON, KML či Garmin FIT, byla vytvořena knihovna, která obsahuje funkce pro jejich vzájemný převod. Komponenta poslední vytvořené knihovny slouží jako správce souborů, který umožňuje zobrazovat adresáře a jejich soubory z libovolného zdroje dat. Tato komponenta nesouvisí přímo s práci s mapou, ale ve vzorové aplikaci byla použita pro výběr a načtení souborů, obsahující geografická data.

Vytvořené komponenty mohou být použity při vývoji nových Angular aplikací či komponent ostatními vývojáři. Výsledná vzorová aplikace může dále sloužit jako alternativa k již existujícím webovým stránkám, které se zabývají elektronickými mapami.

Jako rozšíření aplikace a komponent, ze kterých se skládá, by mohla být implementována geolokace určující aktuální polohu uživatele, která by byla následně zobrazena na mapě či přidána jako bod trasy v komponentě pro plánování tras.

Při tvorbě komponent a aplikace ve frameworku Angular jsem nabyl nové vědomosti o možnostech vývoje webových aplikací.

Literatura

1. *Single-page application* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-04-15]. Dostupné z: https://en.wikipedia.org/wiki/Single-page_application.
2. *What is Angular?* [online]. Mountain View (Kalifornie): Google, c2010-2023 [cit. 2023-04-03]. Dostupné z: https://angular.io/guide/what-is-angular.
3. *TypeScript* [online]. Redmond: Microsoft, c2012-2023 [cit. 2023-04-03]. Dostupné z: https://www.typescriptlang.org/.
4. *The Good and the Bad of Angular Development* [online]. Foster City (Kalifornie): AltexSoft, 2022 [cit. 2023-04-03]. Dostupné z: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/.
5. *The Good and the Bad of Vue.js Framework Programming* [online]. Foster City (Kalifornie): AltexSoft, 2022 [cit. 2023-04-03]. Dostupné z: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/.
6. *Vue.js* [online]. Evan You, c2014-2023 [cit. 2023-04-08]. Dostupné z: https://vuejs.org/.
7. *React (software)* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-04-03]. Dostupné z: https://en.wikipedia.org/wiki/React_(software).
8. *React: The Virtual DOM* [online]. New York City: Codecademy, c2023 [cit. 2023-04-03]. Dostupné z: https://www.codecademy.com/article/react-virtual-dom.
9. BARANOWSKI, Wojciech. *React JS: Advantages and Disadvantages* [online]. Katowice (Polsko): Massive Pixel Creation, 2021 [cit. 2023-04-03]. Dostupné z: https://massivepixel.io/blog/react-advantages-disadvantages/.
10. *Component* [online]. MetaOpenSource, c2023 [cit. 2023-04-04]. Dostupné z: https://react.dev/reference/react/Component.
11. *React* [online]. MetaOpenSource, c2023 [cit. 2023-04-03]. Dostupné z: https://react.dev/.
12. *Writing Markup with JSX* [online]. MetaOpenSource, c2023 [cit. 2023-04-03]. Dostupné z: https://react.dev/learn/writing-markup-with-jsx.

13. *ASP.NET Core Blazor* [online]. Redmond: Microsoft, 2022 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/?view=aspnetcore-7.0.

14. *ASP.NET Core Blazor hosting models* [online]. Microsoft, c2023 [cit. 2023-04-15]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0.

15. *Front-end frameworks* [online]. Sacha Greif, 2022 [cit. 2023-04-16]. Dostupné z: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/.

16. *Most used web frameworks among developers worldwide, as of 2022* [online]. Statista, 2022 [cit. 2023-04-16]. Dostupné z: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/.

17. FAIN, Yakov; MOISEEV, Anton. *Angular development with TypeScript*. Second edition. Shelter Island: Manning, [2019]. ISBN 978-161-7295-348.

18. *CLI Overview and Command Reference* [online]. Mountain View (Kalifornie): Google, c2010-2023 [cit. 2023-04-04]. Dostupné z: https://angular.io/cli.

19. *Workspace and project file structure* [online]. Mountain View (Kalifornie): Google, c2010-2023 [cit. 2023-04-04]. Dostupné z: https://angular.io/guide/file-structure%5C#workspace-and-project-file-structure.

20. *Package.json* [online]. [cit. 2023-04-04]. Dostupné z: https://docs.npmjs.com/cli/v9/configuring-npm/package-json.

21. *NgModules* [online]. Mountain View (Kalifornie): Google, c2010-2023 [cit. 2023-04-04]. Dostupné z: https://angular.io/guide/ngmodules.

22. *Angular components overview* [online]. Mountain View (Kalifornie): Google, c2010-2023 [cit. 2023-04-04]. Dostupné z: https://angular.io/guide/component-overview.

23. *Angular Material* [online]. Mountain Vew (Kalifornie): Google, c2010-2023 [cit. 2023-04-02]. Dostupné z: https://material.angular.io/.

24. *Mapy.cz* [online]. Česká republika: Seznam.cz, 1998 [cit. 2023-04-02]. Dostupné z: https://mapy.cz.

25. *Mapy Google* [online]. Google, 2005 [cit. 2023-04-02]. Dostupné z: https://maps.google.com/.

26. *HTML Canvas* [online]. Mozilla Foundation, 2005 [cit. 2023-04-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.

27. *CanvasRenderingContext2D* [online]. Mozilla Foundation, 2005 [cit. 2023-04-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D.

28. *Geo Coordinates Parser* [online]. Npm, 2023 [cit. 2023-04-08]. Dostupné z: https://www.npmjs.com/package/geo-coordinates-parser.

29. *Angular Material CDK* [online]. Google, c2010-2023 [cit. 2023-04-08]. Dostupné z: https://material.angular.io/cdk.

30. *Mapbox Search API* [online]. Mapbox [cit. 2023-04-08]. Dostupné z: https://docs.mapbox.com/api/search/search/.

31. *Google Drive API* [online]. Google, 2023 [cit. 2023-04-08]. Dostupné z: https://developers.google.com/drive/api.

32. *GeoJSON* [online]. 2016. [cit. 2023-04-08]. Dostupné z: https://geojson.org/.

33. *GPX: the GPS Exchange Format* [online]. Dan Foster [cit. 2023-04-08]. Dostupné z: https://www.topografix.com/gpx.asp.

34. *Flexible and Interoperable Data Transfer (FIT) Protocol* [online]. Garmin LTD, c2023 [cit. 2023-04-08]. Dostupné z: https://developer.garmin.com/fit/protocol/.

35. *KML Documentation Introduction* [online]. Google [cit. 2023-04-08]. Dostupné z: https://developers.google.com/kml/documentation.

36. *DOMParser* [online]. Mozilla Foundation, c1998–2023 [cit. 2023-04-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/DOMParser.

37. *Leaflet* [online]. Volodymyr Agafonkin, c2010-2023 [cit. 2023-04-04]. Dostupné z: https://leafletjs.com/.

38. *Directions API* [online]. Mapbox [cit. 2023-04-04]. Dostupné z: https://docs.mapbox.com/api/navigation/directions/.