FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Aplikace pro podporu otevírání dat o dřevinách pomocí otevřené formální normy
Student:	Ladislav Prix
Vedoucí:	RNDr. Jakub Klímek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem bakalářské práce je podpořit otevírání dat o dřevinách tvorbou otevřené formální normy (OFN) [1] a webové aplikace tuto normu využívající.
Aplikace bude zpracovávat data splňující OFN na základě jejich katalogizace v datovém katalogu jako např. NKOD [2]. Kroky k vypracování:

- Analyzujte data o dřevinách, která poskytují obce.
- Vytvořte pro ně ve spolupráci s vedoucím OFN – konceptuální schéma, datové schéma, příklady a dokumentaci.
- Podmnožinu dostupných dat transformujte dle normy.
- Navrhněte, implementujte, zdokumentujte a otestujte webovou aplikaci, která bude data zobrazovat formou seznamu a mapy a umožní vyhledat nejbližší dřeviny a dřeviny v dané vzdálenosti od určeného místa.
- Ukažte, že pouze přidáním dalších dat dle OFN do katalogu se tato data objeví v aplikaci.
- Aplikaci publikujte jako open-source na GitHubu.

Seznam odborné literatury

[1] Otevřené formální normy, https://ofn.gov.cz
[2] Národní katalog otevřených dat, https://data.gov.cz

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 1. října 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Aplikace pro podporu otevírání dat o dřevinách pomocí otevřené formální normy

Ladislav Prix

Katedra softwarového inženýrství
Vedoucí práce: RNDr. Jakub Klímek, Ph.D.

4. června 2020

Poděkování

Rád bych poděkoval RNDr. Jakubu Klímkovi, Ph.D. za vedení práce, konzultace a věcné připomínky. Dále bych chtěl poděkovat svojí rodině za neustálou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. června 2020 ……

Odkaz na tuto práci

Prix, Ladislav. *Aplikace pro podporu otevírání dat o dřevinách pomocí otevřené formální normy.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Otevřená formální norma je pojem definovaný v § 3 odst. 9 zákona č. 106/1999 Sb. o svobodném přístupu k informacím. Jedná se o datovou specifikaci, která zaručí, že otevřená data splňující tyto požadavky budou mít jednotnou formu a budou navzájem kombinovatelná. V současnosti ale neexistuje mnoho aplikací, které by data splňující tyto otevřené formální normy používaly. To pak demotivuje poskytovatele dat, aby svá data podle otevřených formálních norem publikovali.

Cílem této práce je ukázat poskytovatelům dat výhody poskytování dat podle otevřené formální normy. Toho je dosaženo tvorbou otevřené formální normy pro dřeviny a současnou implementací aplikace, která data o dřevinách podle otevřené formální normy zpracovává a zobrazuje v seznamu a v mapě. Aplikace data, která splňují tuto normu, periodicky stahuje a ihned zobrazuje. Přímo tedy poskytovatelům dat ukazuje výhody poskytování dle otevřené formální normy, tedy že pokud tuto otevřenou formální normu využijí, aplikace třetí strany je mohou ihned začít využívat.

Práce se tedy zabývá analýzou existujících dat a tvorbou otevřené formální normy. Dále se zabývá celým procesem tvorby webové aplikace. Tedy analýzou, návrhem architektury a výběrem technologií, implementací, dokumentací a následně testováním aplikace.

V závěru se práce zabývá hodnocením dosažených výsledků.

Klíčová slova otevřená data, otevřená formální norma, dřeviny, JavaScript, TypeScript, React, C#, ASP.NET Core, Docker

Abstract

An open formal norm is a term defined in § 3 paragraph 9 of Act No. 106/1999 Coll. on free access to information. It is a data specification that guarantees that open data meeting these requirements will have a common form and will be combinable with each other. However, there are currently not many applications that use data that meets these open formal norms. This demotivates data providers to publish their own data according to open formal norm.

The aim of this thesis is to show data providers the benefits of providing data according to an open formal norm. This is achieved by creating an open formal norm for woody plants and by implementing an application that processes and displays woody plants data according to the open formal norm in a list and in a map. The application periodically downloads and displays data that meets this norm. Thus, it directly shows data providers the benefits of providing data according to an open formal norm, ie that if they use this open formal norm, third-party applications can start using them immediately.

The thesis therefore deals with the analysis of existing data and the creation of an open formal norm. It also deals with the whole process of creating a web application. That is, analysis, design of architecture and selection of technologies, implementation, documentation and subsequent testing of the application.

In conclusion, the thesis deals with the evaluation of the achieved results.

Keywords open data, open formal norm, woody plants, JavaScript, TypeScript, React, C#, ASP.NET Core, Docker

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Publikace a využití otevřených dat je jeden ze současných trendů v oblasti práce s daty. Tento trend se výrazně projevuje ve státní správě. [1]

Státní správa poskytuje data podle principů otevřených dat. Otevřená data v České republice jsou termín ukotvený v legislativě, který mimo jiné vyžaduje registraci všech otevřených dat v národním katalogu otevřených dat, který je součástí portálu otevřených dat České republiky [2]. Otevřená data jsou definována v § 3 odst. 11 zákona č. 106/1999 Sb. o svobodném přístupu k informacím[1]. Otevřená data jsou data publikovaná v otevřeném a strojově čitelném formátu. Následné využití těchto dat není nikterak omezeno a jsou evidována v národním katalogu otevřených dat [3].

Národní katalog otevřených dat, dále již jen NKOD, je informační systém pod správou Ministerstva vnitra, který slouží k evidování informací zveřejňovaných jakožto otevřená data [3]. NKOD je definovaný v § 4c zákona č. 106/1999 Sb. o svobodném přístupu k informacím[2] a lze ho nalézt na portále otevřených dat[3].

Otevřená formální norma, dále již jen OFN, je definována v § 3 odst. 9 zákona č. 106/1999 Sb. o svobodném přístupu k informacím[4]. Jedná se o specifikaci požadavků, které zaručí, že data splňující tyto požadavky budou mít jednotnou formu a budou navzájem kombinovatelná [3].

V současné době neexistuje mnoho aplikací, které by využívaly data splňující OFN. Poskytovatelé dat tedy nevidí výhody používání OFN a nemají motivaci OFN používat. Dále existují aplikace pracující s daty, ale nemají jednotný formát pro data a místo toho využívají svůj vlastní proprietární formát. Z tohoto důvodu je pro každou kombinaci dat z různých aplikací potřeba vytvořit transformační skript. Zatímco při využití OFN stačí již jen existující

[1] https://www.zakonyprolidi.cz/cs/1999-106#p3-11
[2] https://www.zakonyprolidi.cz/cs/1999-106#p4c
[3] https://data.gov.cz/
[4] https://www.zakonyprolidi.cz/cs/1999-106#p3-9

data přetransformovat, aby splňovala OFN. Počet potřebných transformačních skriptů se tedy radikálně sníží.

Cíle práce

Cílem práce je vytvořit datový formát a jeho dokumentaci ve formě OFN pro data o dřevinách poskytovaná obcemi. Poté implementovat, zdokumentovat a otestovat webovou aplikaci, která bude data zobrazovat formou seznamu a mapy. Dále bude aplikace umožňovat data o dřevinách vyhledávat a řadit. Aplikace bude dostupná jakožto open-source na GitHubu. Již existující data je potřeba transformovat do definovaného formátu.

Webová aplikace bude poskytovatelům dat ukazovat výhody poskytování dat podle OFN. Bude ukazovat, že stačí pouhé přidání dat splňujících OFN do NKODu, aby se tato data následně objevila v aplikaci. Toto bude sloužit jako přímá motivace poskytovatelům dat, aby tuto OFN použili místo svého proprietárního formátu.

Struktura práce

Kapitola 1 se zabývá analýzou existujících dat o dřevinách a následným návrhem OFN. Dále stručně popisuje vlastnosti vzniklé OFN.

Kapitola 2 se zabývá analýzou požadavků webové aplikace zpracovávající data splňující vzniklou OFN. Na to následně navazuje kapitola 3, která popisuje návrh, architekturu a výběr vhodných technologií webové aplikace.

V kapitole 4 práce popisuje implementaci webové aplikace. Kapitola popisuje vybraná vývojová prostředí, systém správy verzí a následně samotný vývoj. V kapitole 5 je popsána transformace dat aplikace a demo verze aplikace. Následně na to se kapitola 6 zabývá dokumentací této vzniklé aplikace.

Kapitola 7 obsahuje formy testování aplikace, jejich výsledky a případné dopady na budoucí vývoj aplikace.

Kapitola 1

Tvorba otevřené formální normy

Tato kapitola se zabývá analýzou dat a tvorbou OFN pro dřeviny. Již existující OFN se nacházejí na portále otevřených dat[5]. Definice těchto norem se nacházejí ve veřejném repozitáři `opendata-mvcr/otevrene-formalni-normy`[6] na GitHubu.

1.1 Analýza existujících dat

NKOD obsahuje již existující datové sady. Ke dni 14. 5. 2020 katalog vracel na dotazy *stromy* a *dřeviny* dohromady jedenáct výsledků. Většina těchto datových sad obsahuje jen velmi malé množství dat nebo mají data jen velmi málo vlastností, například jen název a identifikátor.

Ústřední seznam ochrany přírody v sekci s památnými stromy[7] obsahuje velké množství záznamů, které obsahují velké množství vlastností. Tyto záznamy je možno exportovat. Nejvíce vlastností, které byly přidány do OFN, pochází z této datové sady.

1.2 Přidání otevřené formální normy

OFN pro dřeviny byla konzultována s Ministerstvem vnitra. Byl pro ni vytvořen konceptuální model a specifikace, ve které jsou definovány jednotlivé vlastnosti. Dále byly vytvořeny příklady ve formátu JSON[8] a JSON schéma[9].

OFN byla přidána do repozitáře norem jako draft v rámci pull requestu **187**[10]. Pull request byl vytvořen z autorova forku repozitáře do originálního

[5] https://ofn.gov.cz
[6] https://github.com/opendata-mvcr/otevrene-formalni-normy
[7] https://drusop.nature.cz/ost/chrobjekty/pstromy/
[8] https://www.json.org/
[9] https://json-schema.org/
[10] https://github.com/opendata-mvcr/otevrene-formalni-normy/pull/187

repozitáře OFN. OFN pro dřeviny lze nyní nalézt mezi ostatními OFN[11].

1.3 Vlastnosti

Vlastnosti byly vybrány na základě analýzy existujících dat. Dále byly přidány vlastnosti podle autorova úsudku a podle připomínek vedoucího práce.

Pro potřeby OFN byly do repozitáře číselníků `opendata-mvcr/ciselniky`[12] na GitHubu přidány dva číselníky. Tyto číselníky jsou publikovány Ministerstvem vnitra v NKODu. Jedná se o číselníky *Kategorie územní ochrany*[13] a *Typy dřevin*[14].

Seznam vlastností:

- **Název** – Název dřeviny ve formátu OFN *Text*[15].
- **Obrázek** – Odkaz na obrázek ve formátu OFN *URL*[16].
- **Poznámka** – Poznámka ke dřevině ve formátu OFN *Text*.
- **Druh dřeviny** – Druh dřeviny ve formátu OFN *Text*.
- **Umístění** – Umístění ve formátu OFN *Umístění*.[17]
- **Kategorie ochrany** – Hodnota z číselníku *Kategorie územní ochrany*.
- **Typ dřeviny** – Hodnota z číselníku *Typy dřevin*.
- **Členské dřeviny** – Pole vnořených dřevin.

Ukázku dat je k vidění na obrázku 1.1.

[11] https://ofn.gov.cz/dřeviny
[12] https://github.com/opendata-mvcr/ciselniky
[13] https://data.gov.cz/datová-sada?iri=https://data.gov.cz/zdroj/datové-sady/MV/706529437/6c8acee4f7be45e5ef90c7d92414dfba
[14] https://data.gov.cz/datová-sada?iri=https://data.gov.cz/zdroj/datové-sady/MV/706529437/f0515f072e9806d2ff5ec3832239a95a
[15] https://ofn.gov.cz/základní-datové-typy/draft/#text
[16] https://ofn.gov.cz/základní-datové-typy/draft/#url
[17] https://ofn.gov.cz/umístění/draft/

```
[
  {
    "název": {
      "cs": "Chráněná krajinná oblast Jeseníky"
    },
    "umístění": {
      "geometrie": {
        "type": "Point",
        "coordinates": [
          50.04361,
          17.21028
        ]
      }
    },
    "členské_dřeviny": [
      {
        "název": {
          "cs": "Klen nad Českou Vsí"
        },
        "poznámka": {
          "cs": "Na svažité louce"
        },
        "umístění": {
          "geometrie": {
            "type": "Point",
            "coordinates": [
              50.25166931594392,
              17.22599651254377
            ]
          },
          "adresa": {
            "název_vúsc": "Olomoucký",
            "název_okresu": "Jeseník",
            "název_obce": "Jeseník"
          }
        }
      }
    ]
  }
]
```


Obrázek 1.1: Ukázka dat splňující OFN ve formátu JSON

Kapitola 2

Analýza webové aplikace

Tato kapitola se zabývá analýzou webové aplikace. Analyzuje požadavky a zkoumá případy užití aplikace.

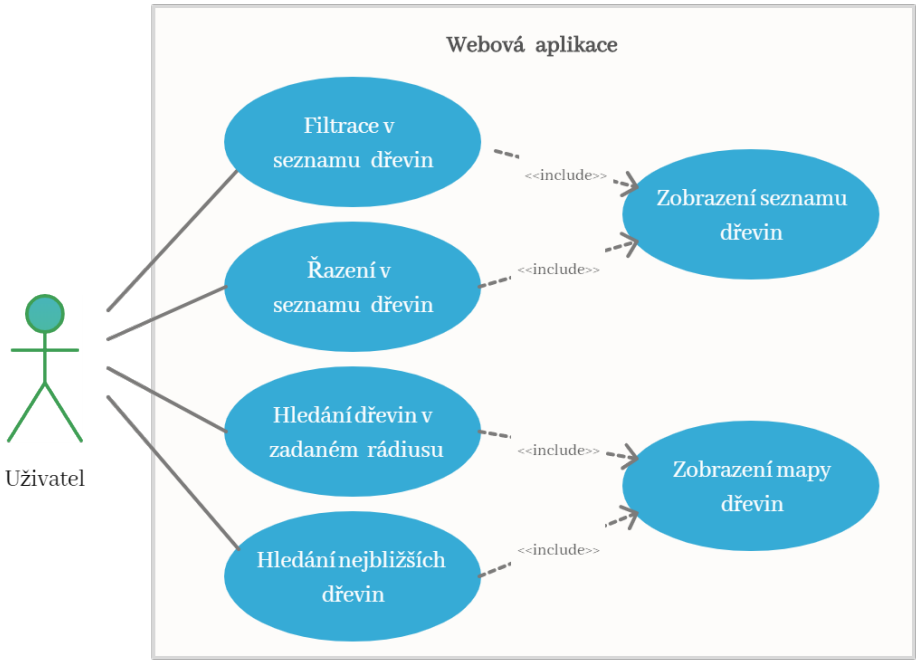
2.1 Požadavky

V této sekci jsou rozebírány funkční a nefunkční požadavky webové aplikace.

2.1.1 Funkční požadavky

Funkční požadavky této aplikace vycházejí ze zadání a jsou to:

- **Zobrazení seznamu dřevin (R1)** – Aplikace bude poskytovat informace o dřevinách formou seznamu.
- **Filtrace v seznamu dřevin (R2)** – Uživatel bude mít možnost dřeviny v seznamu filtrovat dle vybraných polí.
- **Řazení v seznamu dřevin (R3)** – Uživatel bude mít možnost dřeviny v seznamu řadit dle vybraných polí.
- **Zobrazení dřevin v mapě (R4)** – Aplikace bude poskytovat mapu, ve které bude možno dřeviny zobrazovat.
- **Zjištění polohy (R5)** – Aplikace bude zjišťovat polohu uživatele a zobrazovat ji na mapě. Tato poloha bude sloužit jako výchozí bod k vyhledávání v mapě.
- **Vyhledávání dřevin v zadané vzdálenosti (R6)** – Uživatel bude moci určit vzdálenost, ve které budou dřeviny hledány.
- **Vyhledání nejbližších dřevin od zadané polohy (R7)** – Uživatel bude moci zadat počet dřevin, které má aplikace najít.



Obrázek 2.1: Případy užití

- **Propojení s katalogem otevřených dat (R8)** – Aplikace si bude periodicky stahovat data o dřevinách splňující OFN a ty následně zobrazovat.

2.1.2 Nefunkční požadavky

Nefunkční požadavky této aplikace taktéž vycházejí ze zadání a jedná se o:

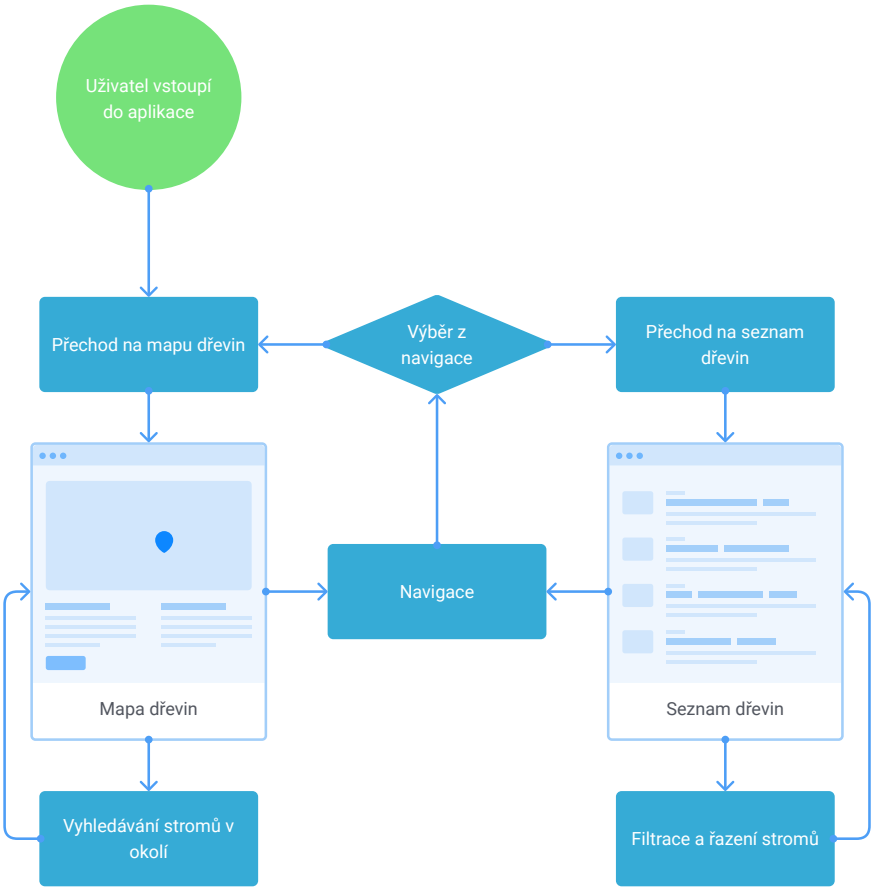
- **Jednoduché nasazení** – Aplikaci bude možno snadno a rychle nasadit.

2.2 Případy užití

Model případů užití se nachází na obrázku 2.1 a jednotlivé případy jsou detailněji rozepsány. Dále na obrázku 2.2 je zobrazen průchod uživatele aplikací.

- **Zobrazení seznamu dřevin**
  1. Uživatel spustí webovou aplikaci.
  2. Uživatel si zobrazí seznam dřevin.

- **Filtrace v seznamu dřevin**
  1. Uživatel si zobrazí seznam dřevin.
  2. Uživatel zadá filtr.
  3. Aplikace mu zobrazí filtrovaná data.
- **Řazení v seznamu dřevin**
  1. Uživatel si zobrazí seznam dřevin.
  2. Uživatel použije řazení.
  3. Aplikace mu zobrazí seřazená data.
- **Zobrazení mapy dřevin**
  1. Uživatel spustí webovou aplikaci.
  2. Uživatel si zobrazí mapu dřevin.
- **Hledání dřevin v zadaném rozsahu**
  1. Uživatel si zobrazí mapu dřevin.
  2. Zadá pozici v mapě a rádius, ve kterém chce hledat.
  3. Aplikace mu zobrazí dřeviny v zadaném rozsahu.
- **Hledání nejbližších dřevin**
  1. Uživatel si zobrazí mapu dřevin.
  2. Zadá pozici v mapě a počet dřevin, které chce hledat.
  3. Aplikace mu zobrazí dřeviny, které se nacházejí nejblíže k pozici, kterou uživatel zadal.



Obrázek 2.2: Uživatelský průchod aplikací

Kapitola 3

Návrh webové aplikace

Tato kapitola se zabývá architekturou aplikace a výběrem vhodných technologií.

3.1 Architektura aplikace

Mezi nejznámější typy architektur patří architektury monolitická, vícevrstvá a architektura mikroslužeb [4].

3.1.1 Monolitická architektura

V monolitické architektuře se veškerý zdrojový kód nachází v jednom projektu. Tento přístup nečiní problém u menších aplikací, ale může být problém u větších aplikací, kde se zdrojový kód aplikace může snadno stát nepřehledný a neudržovatelný.

3.1.2 Vícevrstvá architektura

Vícevrstvá architektura, na rozdíl od monolitické architektury, má oddělené různé vrstvy aplikace. Každá vrstva je zodpovědná za určitou část logiky aplikace. Jedním možným členěním je například na vrstvy:

- **Prezentační vrstva** – Je zodpovědná za zobrazení dat uživateli.
- **Byznys vrstva** – Je zodpovědná za vykonávání byznys logiky aplikace.
- **Datová vrstva** – Je zodpovědná za uchování a získávání dat.

Výhodou tohoto přístupu je možnost rozdělit vrstvy aplikace mezi různé projekty, které lze následně editovat a případně i nasazovat nezávisle na sobě.

3.1.3 Architektura mikroslužeb

Architektura mikroslužeb se zaměřuje především na rozdělení aplikace na menší služby podle funkcionalit, které aplikace zastává a následně na komunikaci mezi těmito službami. Tento přístup usnadňuje editaci a nasazení jednotlivých služeb nezávisle na sobě. Dále tento přístup usnadňuje škálování aplikace, jelikož se mohou samostatně škálovat jen ty služby, které jsou nejvíce zatížené.

3.1.4 Výběr architektury

Pro aplikaci byla zvolena vícevrstvá architektura. Aplikace je tedy rozdělena na serverovou a klientskou část, kde klientská část obstarává prezentační vrstvu a ostatní vrstvy jsou obsažené v serverové části aplikace. Klient bude komunikovat se serverem pomocí HTTP aplikačního rozhraní. Tento způsob umožňuje nezávislý vývoj klientské a serverové části, který může probíhat i v různých jazycích a jediné, v čem se obě části musejí shodnout, je aplikační rozhraní.

Aplikace je menšího rozsahu, byla by tedy dostačující monolitická architektura. Vícevrstvá architektura byla zvolena pro zvýšení přehlednosti zdrojového kódu. Z důvodu rozsahu aplikace není možno hovořit ani o architektuře mikroslužeb, jelikož aplikace nemá dostatek funkcionalit, ze kterých by mohly vzniknout samostatné služby. Jelikož se ale vícevrstvá architektura a architektura mikroslužeb zcela nevylučují, je možné při dalším rozšiřování aplikace využívat principů mikroslužeb.

3.1.5 Serverová část

Serverová část aplikace poskytuje HTTP aplikační rozhraní, pomocí kterého s ním komunikuje klientská část. Server si data o dřevinách periodicky stahuje a následně ukládá do databáze. Toto jsou dvě na sobě nezávislé funkcionality, z toho důvodu je serverová část aplikace rozdělena na dvě služby:

- První služba bude sloužit ke stahování dat a následnému ukládání do databáze.
- Druhá služba bude poskytovat HTTP aplikační rozhraní pro klienta a bude umožňovat filtraci, řazení a získávání dat.

Výhodou tohoto přístupu je především oddělení nezávislých částí kódu, a díky tomu zvýšení čitelnosti kódu. Dále tento přístup zvýší odolnost aplikace vůči výpadku, jelikož služba pro stahování dat není pro chod aplikace potřebná.

Technologie

Aplikace nemá žádné nefunkční požadavky na technologie, mohla tedy být použita jakákoliv technologie, která poskytuje možnosti pro tvorbu HTTP aplikačního rozhraní, například Node.js, Python Flask nebo PHP Nette.

Pro serverovou část aplikace byl zvolen jazyk C# na platformě .NET Core a framework pro tvorbu webových aplikací ASP.NET Core. Kombinace C# a ASP.NET Core byla zvolena především kvůli zkušenostem autora s tímto jazykem a frameworkem. Popis jazyku C# a frameworku ASP.NET Core se nachází v sekcích 3.2.2 až 3.2.4.

Databáze

Data o dřevinách obsahují několik vnořených podstruktur, například umístění, které obsahuje souřadnice ve formátu GeoJSON[18]. Pro takto strukturovaná data je vhodnější místo tradiční SQL databáze zvolit NoSQL databázi s volným schématem, nejlépe dokumentovou, ve které půjde mít data v jedné struktuře a nebude potřeba je na základě dotazů spojovat, jak by to bylo potřeba v případě SQL databáze. Dále lze očekávat stále rostoucí počet dat a použití NoSQL databáze zjednoduší případné škálování.

Mezi databáze, které by byly vhodné pro tuto aplikaci, patří například CouchDB, MongoDB nebo DynamoDB.

Pro tuto aplikaci byla zvolena MongoDB a to především kvůli autorově zkušenostem s touto databází a prostorovému indexu, který MongoDB poskytuje. Tento index bude potřeba pro dotazy v mapě. Databáze MongoDB a prostorový index jsou detailněji popsány v sekci 3.2.5.

3.1.6 Klientská část

Klientská část aplikace reaguje na uživatelský vstup, komunikuje se serverem a generuje uživatelské rozhraní.

Technologie

Pro klientskou část aplikace je potřeba technologie, která umožňuje komunikaci s HTTP aplikačním rozhraním serveru. V současné době jsou v této oblasti velmi populární JavaSciptové frameworky a knihovny jako například Angular[19], React[20] nebo Vue.js[21]. Tento trend je vidět například v průzkumu od Stack Overflow za rok 2019 [5].

[18] https://geojson.org/
[19] https://angular.io/
[20] https://reactjs.org/
[21] https://vuejs.org/

Pro tuto aplikaci byla zvolena knihovna React a to především z důvodu autorova zájmu o prohloubení znalostí v této oblasti. Popis jazyku JavaScript a knihovny React se nachází v sekcích 3.2.6 až 3.2.8.

Jeden z požadavků na aplikaci je zobrazení dřevin v mapě. Bylo tedy potřeba vybrat technologii, pomocí které se bude mapa a následně data v mapě vykreslovat. Při volbě technologií pro mapu je možno zvolit placené služby, jako například Google Maps JavaScript API nebo Mapbox. Tyto služby nabízejí výchozí počet požadavků, který je zdarma, ale následně je potřeba za požadavky platit. Například Mapbox nabízí padesát tisíc vykreslení mapy za měsíc zdarma. Tento údaj platil v době psaní práce, konkrétně k datu 17. 5. 2020.

Jelikož tato aplikace je open-source, je vhodné zvolit také nějaké open-source a neplacené řešení. Mezi taková řešení patří například knihovna Leaflet, popsána v sekci 3.2.9.

3.1.7 Platforma pro nasazení

Vzhledem k vybraným technologiím je pro běh serverové části potřeba .NET Core běhové prostředí, které je multiplatformní. Je tedy potřeba mít na prostředí, kam se aplikace nasazuje, nainstalované běhové prostředí. Klientskou část je potřeba umístit na webový server. Pro zjednodušení těchto požadavků je možno využít kontejnerizace aplikace. Jedna z platforem, které toto umožňují, je Docker.

Docker poskytuje jednotnou platformu pro vývoj i pro nasazení. Redukují se tím problémy s tím, že aplikace na vývojovém prostředí funguje a na produkčním již ne. Dále zjednodušuje nasazení aplikací. Pro spuštění kontejnerů stačí pouze nainstalovaný a spuštěný Docker. Z těchto důvodů byla pro vývoj a vydání veškerých služeb aplikace zvolena technologie Docker. Docker je detailněji popsán v sekci 3.2.10.

3.2 Technologie

V této sekci se nachází popis technologií, na které se práce odkazuje v předchozí sekci s architekturou.

3.2.1 Přehled

Stručný přehled vybraných technologií:

- **Serverová část** – Jazyk C#, platforma .NET Core, framework pro tvorbu webových aplikací APS.NET Core
- **Klienská část** – Jazyk JavaScipt a jeho nadstavba TypeScript, knihovna pro tvorbu uživatelského rozhraní React
- **Mapa** – Leaflet
- **Databáze** – NoSQL databáze MongoDB
- **Platforma pro nasazení a vývoj** – Docker

Jednotlivé technologie jsou dále popsané v této sekci.

3.2.2 C#

C# [6] je objektově orientovaný jazyk vyvíjený společností Microsoft. C# se vyznačuje silnou typovostí.

3.2.3 .NET Core

.NET Core [7] je open-source vývojová platforma vyvíjená společností Microsoft a komunitou na GitHubu. Pro platformu vycházejí pravidelné aktualizace. Verze použitá v tomto projektu je 3.1, která byla vydána začátkem roku 2020.

3.2.4 ASP.NET Core

ASP.NET Core [8] je framework pro tvorbu webových aplikací. Je vyvíjen společností Microsoft. Oproti svému předchůdci ASP.NET je ASP.NET Core multiplatformní a open-source. Vývoj ASP.NET Core je těsně spjatý s platformou .NET Core, aktualizace vycházejí ve stejnou dobu jako aktualizace pro ni.

SignalR

SignalR je protokol pro oboustrannou komunikaci mezi klientem a serverem, který využívá transport pomocí WebSocketu v prostředí, které tento transport umožňují. Díky použití SignalR oproti čistému WebSocketu je možno vynechat nastavování komunikace, která je již vyřešená v knihovně pro SignalR a je možno věnovat se implementaci aplikace. [9]

3.2.5 MongoDB

MongoDB je NoSQL dokumentová databáze vyznačující se vysokou výkonností a snadnou škálovatelností. Záznam v MongoDB se nazývá dokument. Jedná se o datovou strukturu složenou z klíčů a hodnot. MongoDB dokumenty jsou strukturou podobné formátu JSON, mohou obsahovat vnořené dokumenty a pole. Jednotlivé dokumenty jsou uložené v kolekcích, které jsou analogické k tabulkám z relačních systémů. [10]

Dále MongoDB umožňuje použití 2d indexu, který bude potřeba pro geografické dotazy, které jsou popsané ve funkčních požadavcích. Například umožňuje vyhledat objekty v zadaném rozsahu od zadaného bodu a seřadit je podle vzdálenosti od zadaného bodu.

Indexy

Indexy slouží k efektivnímu dotazování nad kolekcemi. MongoDB poskytuje širokou škálu indexů, mezi které patří i prostorový index, který umožňuje mimo jiné vyhledávat nejbližší objekty od zadaného bodu a objekty nacházející se v zadané ploše. [11]

3.2.6 JavaScript

JavaScript je multiplatformní skriptovací jazyk. Jeho syntax je založený na jazycích jako Java nebo C. JavaScript podporuje objektově orientované a funkcionální programování. Nejčastější prostředí pro JavaScript je prohlížeč na klientské straně aplikací. Není ovšem na prohlížeč limitován, například v serverovém prostředí Node.js. [12]

3.2.7 TypeScript

TypeScript je nadstavba nad JavaScriptem vyvinutá společností Microsoft. TypeScript přináší do JavaScriptu statické typování a tedy i typovou kontrolu, která minimalizuje počet programátorských chyb. Veškerý zdrojový kód napsaný v TypeScriptu se kompiluje do JavaScriptu a umožňuje aplikace napsané v TypeScriptu spustit ve všech prostředích, která podporují JavaScript. [13]

3.2.8 React

React je JavaScriptová knihovna, vyvinutá společností Facebook, sloužící k vytváření uživatelského rozhraní. React funguje na principu komponent, které si udržují svůj stav. Následně se komponenty skládají dohromady a tvoří komplexní celky. Díky tomu, že logika komponent je napsaná v JavaScriptu, je možno lehce pracovat s velkými objemy dat bez nutnosti jejich ukládání do DOMu [14]. [15]

Create React App

Create React App [16] je nástroj vyvinutý společností Facebook. Slouží k zjednodušení procesů inicializace, spuštění a testování React aplikací. Tento nástroj byl použit z důvodu zjednodušení nastavení projektu. Autor se tedy mohl soustředit především na samotnou implementaci.

JSX

JSX je syntaktické rozšíření, které do JavaScriptu zavádí prvky XML[22]. V jednotlivých komponentách je tedy možné používat JSX zápis a kompilátor následně tento zápis přeloží do čistého JavaScriptu. Toto rozšíření zpřehledňuje zdrojový kód a umožňuje spojení logiky vykreslování s logikou uživatelského rozhraní. [17]

3.2.9 Leaflet

Leaflet je open-source JavaScriptová knihovna pro vytváření interaktivních map. Leaflet poskytuje jednoduché a zároveň rychlé rozhraní pro vytváření aplikací. Funguje na většině dostupných platforem a poskytuje možnost využívání modulů, které ke knihovně přidávají dodatečné funkcionality. [18]

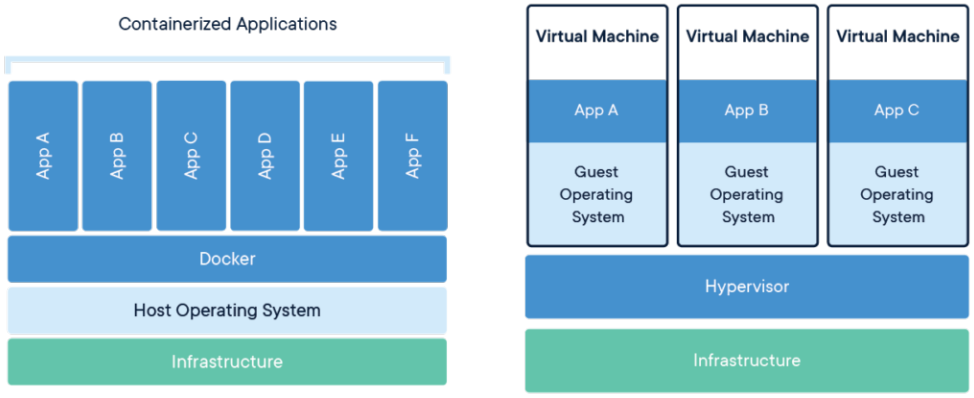
3.2.10 Docker

Docker je platforma pro tvorbu a běh aplikací [19]. Docker umožňuje zabalení a běh aplikace v izolovaném prostředí zvaném kontejner. Kontejnery jsou oproti běžným virtuálním strojům odlehčené, protože běží přímo v operačním systému hostujícího stroje, zatímco při běžné virtualizaci je potřeba vlastní operační systém se všemi jeho vrstvami, jak je k vidění na obrázku 3.1.

Před spuštěním aplikace v kontejneru je potřeba mít k dispozici obraz, ze kterého se kontejner vytvoří. Obraz se vytváří za pomocí souboru zvaného Dockerfile[23], který obsahuje instrukce pro vytvoření obrazu.

[22] https://www.w3.org/XML/

[23] https://docs.docker.com/engine/reference/builder/



Obrázek 3.1: Rozdíl virtualizace a kontejnerizace [21]

Docker Compose

Docker Compose je nástroj, který slouží ke spouštění více dockerových kontejnerů najednou [20]. Pomocí souboru ve formátu YAML je možno vytvořit definici pro více kontejnerů složené z dockerových obrazů a následně je pomocí jednoho příkazu spouštět.

3.2.11 CI a CD

Continuous integration (CI) a continuous delivery (CD) jsou sada principů a standardů, které umožňují vývojářským týmům snadnější a spolehlivější dodávání změn ve zdrojovém kódu [22].

Cílem CI je nastavit proces, při kterém se po změně zdrojového kódu projekt automaticky sestaví, otestuje a případně provede další akce v závislosti na projektu [22]. Příklad takového procesu je:

1. Zdrojový kód se dostane do hlavní větve Git repozitáře.
2. Spustí se testy.
3. Vytvoří se dockerový obraz.
4. Dockerový obraz se přiřadí do dockerového repozitáře.

GitHub Actions

Github Actions [23] umožňují vytváření CI a CD procesů hostovaných na GitHubu. Definice těchto procesů se nacházejí v YAML[24] souborech přímo v repozitářích se zdrojovým kódem. Díky spoustě již vytvořených akcí není

[24] https://yaml.org/

třeba elementární procesy vytvářet od začátku, ale je možno použít již připravené akce. Například naklonování Git repozitáře nebo vytvoření a nasazení dockerového obrazu již mají vytvořené akce.

Github Actions poskytují dva tisíce minut běhu zdarma na hostovaných strojích za měsíc. Pro další minuty je již potřeba zaplatit. Tyto informace jsou platné k datu 14. 5. 2020, aktuální informace o cenách lze najít v dokumentaci Guthub Actions[25].

[25]https://help.github.com/en/github/setting-up-and-managing-billing-and-payments-on-github/about-billing-for-github-actions

Kapitola 4

Implementace webové aplikace

Tato kapitola se zabývá samotnou implementací webové aplikace a popisem nástrojů, které byly při vývoji využité. Aplikace dostala název **TreeLoc**. Název je odvozen od anglických slov *tree* a *location*.

4.1 Vývojové prostředí

Tato sekce se zabývá vývojovými prostředími, která byla využita při vývoji webové aplikace.

4.1.1 Visual Studio

Pro vývoj serverové části bylo použito prostředí Visual Studio 2019 Community. Visual Studio je integrované vývojové prostředí on firmy Microsoft určené pro vývoj a ladění aplikací [24]. Verze Community je zdarma. To dává Visual Studiu značnou výhodu oproti konkurenčnímu vývojovému prostředí Rider[26] od firmy JetBrains, které v čase psaní této práce nemá žádnou verzi produktu zcela zdarma.

4.1.2 Visual Studio Code

Pro vývoj klientské části aplikace bylo využito prostředí Visual Studio Code. Visual Studio Code je odlehčený textový editor zdrojového kódu, který má v základu podporu pro JavaScript, TypeScript a Node.js [25]. Dále poskytuje velké množství rozšíření, které zajišťují podporu pro spoustu dalších moderních programovacích jazyků a nástrojů.

Visual Studio Code má zabudovanou podporu pro Git a Docker. Pro Git umožňuje procházení změn a spouštění Git příkazů přímo z editoru. Pro

[26] https://www.jetbrains.com/rider/

Docker poskytuje uživatelské rozhraní pro správu dockerových kontejnerů a obrazů.

Při vývoji této aplikace bylo využito rozšíření `Prettier`[27]. Toto rozšíření umožňuje automatické formátování zdrojového kódu.

4.2 Systém správy verzí

Tato sekce se zabývá systémem správy verzí, který byl využit při vývoji této webové aplikace. Mezi systémy správy verzí patří například SVN[28], RCS[29] nebo Git[30]. Zadání vyžaduje publikaci projektu na GitHubu[31]. GitHub je webová služba podporující verzování zdrojového kódu za pomoci Gitu. Pro tento projekt byl tedy zvolen Git.

[27] https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode
[28] https://subversion.apache.org/
[29] https://www.gnu.org/software/rcs/
[30] https://git-scm.com/
[31] https://github.com/

4.3 Implementace serverové části

Tato sekce se zabývá implementací serverové části aplikace. Jak je zmíněno v kapitole s návrhem aplikace, serverová část aplikace je implementována v jazyce C#. Zdrojový kód serverové části je verzovaný na GitHubu v repozitáři `prixladi/treeloc-backend`[32].

4.3.1 Databáze

Jak je uvedeno v návrhové části, tak byla využita databáze MongoDB. Pro C# existuje knihovna, která zajišťuje plně asynchronní komunikaci s MongoDB [26].

4.3.2 Struktura projektů

Serverová část aplikace je rozdělena na projekty:

- **Docker-compose** – Projekt obsahuje soubor **docker-compose.yml**. Tento soubor obsahuje definice spouštěných služeb a slouží nástroji Docker Compose k jejich spuštění.
- **TreeLoc.Database** – Jedná se o knihovnu, která obsahuje definice databázových tříd využitých v projektu. Dále obsahuje společný kontext pro připojení k databázi.
- **TreeLoc.Core** – Tento projekt je také knihovna. Obsahuje společné části zdrojového kódu užívané napříč ostatními projekty.
- **TreeLoc.Api** – Tento projekt je samotná webová služba, která poskytuje aplikační rozhraní. Projekt obsahuje vlastní `Dockerfile` soubor, který slouží k vytvoření dockerového obrazu služby. Tato služba odpovídá službě poskytující aplikační rozhraní navržené v sekci 3.1.5 s architekturou aplikace.
- **TreeLoc.Loader** – Jedná se o projekt, který odpovídá službě sloužící k nahrávání dat o dřevinách do aplikace navržené v sekci 3.1.5 s architekturou aplikace. Stejně jako **TreeLoc.Api** tento projekt obsahuje `Dockerfile`, který slouží k vytvoření dockerového obrazu služby.

4.3.3 Docker-compose

Soubor `docker-compose.yml` tohoto projektu obsahuje definici tří kontejnerů popsaných níže. Dále umožňuje nastavování proměnných prostředí a mapování portů.

[32] https://github.com/prixladi/treeloc-backend

- **mongodb** – Tento kontejner vychází z obrazu `mongo:latest`, a jak název napovídá, jedná se o kontejner, který obsahuje MongoDB databázi.
- **api** – Tento kontejner nevychází z publikovaného obrazu, ale vytváří se pomocí `Dockerfile` v projektu **TreeLoc.Api**.
- **loader** – Tento kontejner nevychází z publikovaného obrazu, ale vytváří se pomocí `Dockerfile` v projektu **TreeLoc.Loder**.

4.3.4 TreeLoc.Database

Projekt obsahuje definice tří databázových dokumentů a jejich indexů. Prvním dokumentem je `VersionDocument`, který obsahuje informace o verzi dat.

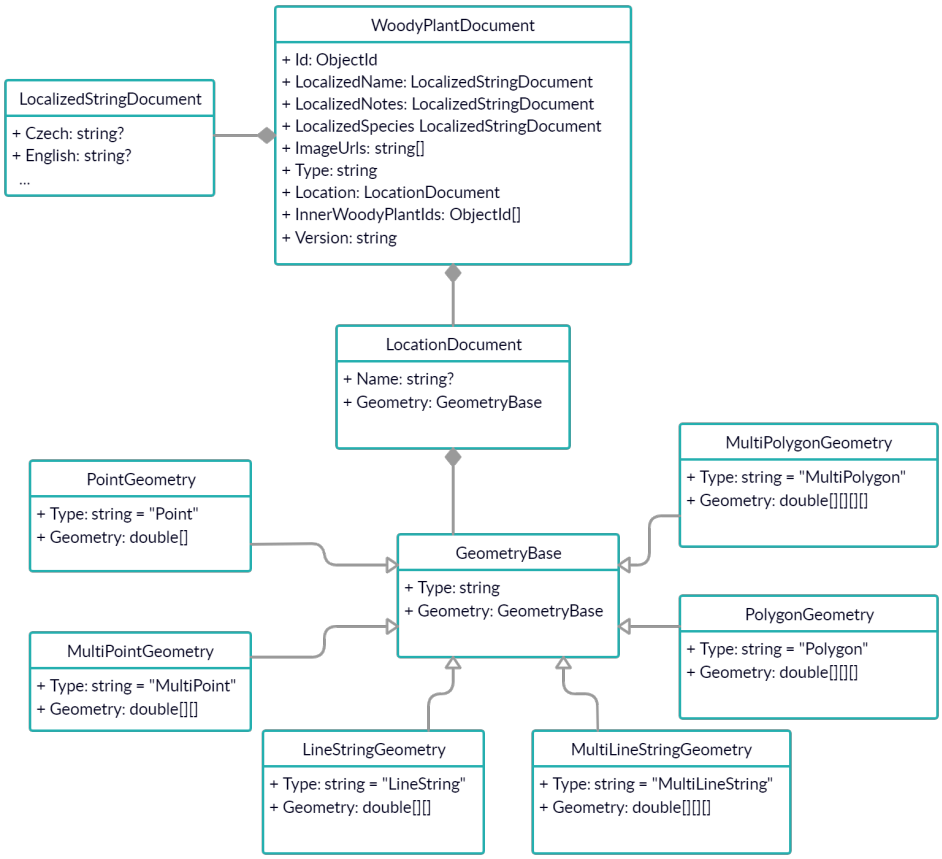
Další dokument je `ResourceDocument`. Jeden dokument odpovídá jedné URL adrese datové sady a drží si u sebe příznak, zda byla datová sada stažena.

Poslední dokument je `WoodyPlantDocument`. Jedná se o klíčový dokument, který obsahuje veškerá data o dřevinách. Jeho struktura se skládá z podmnožiny vlastností z navržené OFN. Strukturu dokumentu lze vidět na obrázku 4.1.

Indexy

Dokument `WoodyPlantDocument` obsahuje indexy:

- **LocalizedNames.Czech** – Index na vlastnosti `Czech` na vnořeném dokumentu `LocalizedNames`, který umožňuje rychlé řazení podle této vlastnosti.
- **LocalizedNotes.Czech** – Index na vlastnosti `Czech` na vnořeném dokumentu `LocalizedNotes`, který umožňuje rychlé řazení podle této vlastnosti.
- **LocalizedSpecies.Czech** – Index na vlastnosti `Czech` na vnořeném dokumentu `LocalizedSpecies`, který umožňuje rychlé řazení podle této vlastnosti.
- **Geo2DSphere** – Geografický index na vlastnosti `Geometry` na vnořeném dokumentu `Location`, který umožňuje geografické vyhledávání.
- **Text** – Textový index nad vlastnostmi `LocalizedNames.Czech`, `LocalizedNotes.Czech` a `LocalizedSpecies.Czech`, který umožňuje textové vyhledávání. MongoDB v současné verzi neumí českou lokalizaci, index tedy neumí odfiltrovat česká stop slova a nezná kořeny slov.



Obrázek 4.1: Struktura dokumentu WoodyPlantDocument

4.3.5 TreeLoc.Loader

Projekt využívá framework ASP.NET Core ve verzi 3.1 a je konfigurovatelný pomocí proměnných prostředí:

- **MONGO_URL** – Tato proměnná obsahuje připojovací řetězec k databázi.
- **MONGO_DATABASE_NAME** – Tato proměnná obsahuje jméno databáze.
- **DISCOVERY_URL** – Tato proměnná obsahuje URL adresu, kterou aplikace periodicky volá a zjišťuje z ní odkazy na datové sady.
- **DISCOVERY_INTERVAL** – Tato proměnná nastavuje dobu periody volání **DISCOVERY_URL** ve vteřinách.

- **LOADER_INTERVAL** – Tato proměnná nastavuje dobu čekání po nahrání datové sady ve vteřinách, po kterém dojde k nahrání další datové sady.
- **REMOVE_OLD** – Příznak, zda se po spuštění mají smazat všechna data, která byla uložená v databázi z předchozího běhu.

Nahrávání dat

Služba periodicky odesílá HTTP GET požadavek na adresu uloženou v proměnné prostředí `DISCOVERY_URL`. Interval periody je určen proměnnou prostředí `DISCOVERY_INTERVAL`. Jako odpověď očekává seznam URL adres datových sad. Po získání adres je služba uloží do databáze a označí je jakožto nestažené.

Dále služba v periodickém intervalu, který je určen proměnnou prostředí `LOADER_INTERVAL`, stahuje datové sady, jejichž adresy jsou uložené v databázi a jsou označené jakožto nestažené. Po stažení datové sady se data přetransformují na databázové dokumenty a adresa stažené datové sady se označí v databázi jakožto stažená.

V případě, že je nastavený příznak v proměnné `REMOVE_OLD`, dojde po spuštění služby **TreeLoc.Loader** ke smazání všech dat o dřevinách a smazání údajů o URL adresách datových sad.

Funkcionalita nahrávání dat splňuje funkční požadavek **R8**.

SignalR

TreeLoc.Loader služba poskytuje SignalR koncový bod `/signalr`, na který se klienti mohou připojit a server jim odesílá informace o tom, že byla načtena další data. Jak je popsáno v sekci 3.2.4, SignalR umožňuje obousměrnou komunikaci. V současné době je využita jen komunikace ze serveru na klienta. Do budoucna je možno přidat odesílání informací o uživatelově poloze a filtru na mapě, aby bylo možno odesílat je ty informace o změně dat, které jsou pro uživatele relevantní.

4.3.6 TreeLoc.Api

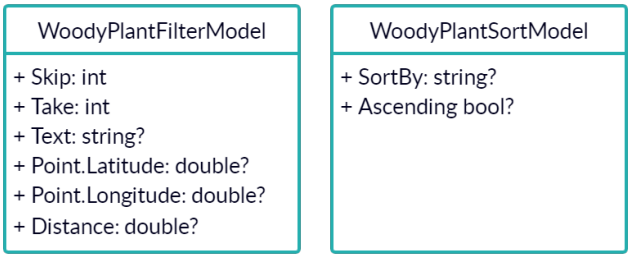
Stejně jako **TreeLoc.Loader** tento projekt využívá ASP.NET Core framework. Služba vznikající z tohoto projektu poskytuje HTTP aplikační rozhraní pro klientské aplikace, které je popsáno v sekci 4.3.7.

V projektu je využita knihovna `MediatR`[33], která řeší odesílání a zpracovávání zpráv za běhu programu.

Dále jsou v projektu využity knihovny `MongoDB`, která slouží ke komunikaci s databází, a `Swashbuckle.AspNetCore`, která je popsaná v sekci 4.3.8.

Projekt je konfigurovatelný přes proměnné prostředí:

[33] https://github.com/jbogard/MediatR/wiki



Obrázek 4.2: Struktura modelů filtrace a řazení dat

- **MONGO_URL** – Tato proměnná obsahuje připojovací řetězec k databázi.
- **MONGO_DATABASE_NAME** – Tato proměnná obsahuje jméno databáze.

4.3.7 HTTP API

TreeLoc.Api služba poskytuje HTTP aplikační rozhraní, pomocí kterého klientská aplikace získává data. Pro dokumentaci rozhraní byla využita knihovna `Swashbuckle`, která je popsána v sekci 4.3.8 o OpenAPI. Celkově jsou k dispozici tři koncové body.

GET /api/v1/woodyPlants

Modely pro filtraci a řazení dat, které tento koncový bod přijímá, jsou vidět na obrázku 4.2.

Filtrační model `WoodyPlantFilterModel` obsahuje pět vlastností. Povinné vlastnosti `Skip` a `Take` definují stránkování. Nepovinná vlastnost `Text` definuje textový filtr. Nepovinné vlastnosti `Point.Latitude` a `Point.Longitude` se využívají pro určení bodu geografického vyhledávání.

Model pro řazení dat `WoodyPlantSortModel` obsahuje dvě vlastnosti. Nepovinná vlastnost `SortBy` určuje, podle jaké vlastnosti se budou data řadit. Nepovinná vlastnost `Ascending` určuje, zda se data budou řadit vzestupně, nebo sestupně.

Koncový bod vrací vyfiltrované a seřazené náhledy dřevin a celkový počet dřevin, které odpovídají filtru bez stránkování.

GET /api/v1/woodyPlants/{id}

Na HTTP dotazovací metodu GET vrátí detail dřeviny se zadaným identifikátorem `{id}` v url.

GET /api/v1/info

Na HTTP dotazovací metodu GET vrátí informace o verzi dat na serveru.

4.3.8 OpenAPI

OpenAPI specifikace definuje standard pro popis HTTP aplikačního rozhraní, který je lidsky i strojově čitelný. Tento popis může být použit nástroji pro generování kódu k vygenerování klientské nebo serverové části aplikace. Dále může být využit nástroji pro generování dokumentace. [27]

Swagger

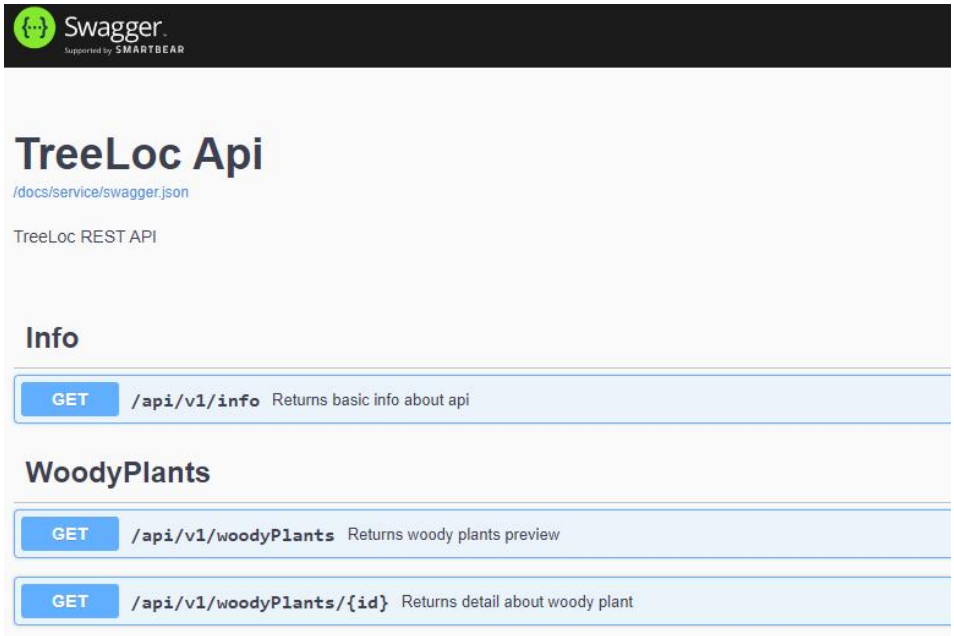
Swagger [28] je sada nástrojů, které slouží k práci s OpenAPI specifikací. Pro ASP.NET Core existuje knihovna `Swashbuckle.AspNetCore`, která umožňuje práci se `Swagger` nástroji. Umožňuje generovat OpenAPI popis za pomoci XML značek v kódu. Popis ve formátu JSON je po spuštění aplikace dostupný na koncovém bodu `/docs/service/swagger.json`.

Dále knihovna poskytuje endpoint `/swagger`, který pomocí vygenerovaného popisu a nástroje Swagger UI [29] generuje uživatelské rozhraní, které může uživatel aplikačního rozhraní použít k testování koncových bodů aplikace. Ukázka rozhraní vygenerovaného pro tuto aplikaci je vidět na obrázku 4.3.

4.3.9 CI

Pro serverový projekt byla vytvořena definice CI pro Github Actions[34], která zajistí vytvoření dockerových obrazů a jejich přiřazení do repozitářů na Docker Hubu[35]. Definice CI se nachází v souboru `.github/workflows/main.yml` v serverovém repozitáři. Definice využívá již existující akci `docker/build-push-action@v1`[36], která vytvoří obraz za pomocí předané cesty k `Dockerfile` souboru a následně, za pomoci předaných přihlašovacích údajů, přiřadí obraz do předaného dockerového repozitáře. Pro službu **TreeLoc.Api** byl zvolen repozitář v Docker Hubu `treeloc/backend-api`. Pro službu **TreeLoc.Loader** byl zvolen repozitář v Docker Hubu `treeloc/backend-loader`.

[34] popsáno v sekci 3.2.11
[35] https://hub.docker.com/
[36] https://github.com/docker/build-push-action



Obrázek 4.3: Ukázka uživatelského rozhraní vygenerovaného pomocí nástroje Swagger UI v aplikaci TreeLoc

4.4 Implementace klientské části

Tato sekce se zabývá implementací klientské části aplikace. Jak již bylo zmíněno v kapitole s návrhem aplikace, klientská část aplikace je implementována v jazyce JavaScript s využitím nadstavby TypeScript a knihovny React. Zdrojový kód klientské části je verzovaný na Githubu v repozitáři `prixladi/treeloc-frontend`[37].

V celé aplikaci je využita knihovna Ant Design[38], která umožňuje využití existujících komponent a usnadňuje vývoj aplikace, jelikož komponenty vyžadují již jen minimální konfiguraci pro použití.

4.4.1 Konfigurace aplikace

Aplikace je konfigurovatelná pomocí proměnných prostředí:

- **API_URL** – URL adresa aplikačního rozhraní služby **TreeLoc.Api**.
- **LOADER_URL** – URL adresa **TreeLoc.Loader** služby. Využívá se k připojení ke SignalR koncovému bodu.

4.4.2 Struktura aplikace

Klientská část aplikace je rozdělena dvě stránky, na seznam dřevin a na mapu dřevin. U obou částí se na vrcholu nachází navigace, která jako základ využívá komponentu `Menu`[39]. Navigace obsahuje odkazy na seznam dřevin, mapu dřevin a na GitHub repozitář projektu. Přechody mezi stránkami jsou vyřešeny pomocí balíčku `react-router-dom`[40], který poskytuje sadu komponent pro směrování mezi stránkami pomocí cesty v URL adrese. Stránka se seznamem dřevin je směrována přes cestu `/list` a stránka s mapou přes cestu `/map`. Stránka s mapou je nastavena jako výchozí, takže po příchodu do aplikace bude uživatel přesměrován na `/map`. Ukázka nastavení směrování, která pochází přímo z aplikace, je na obrázku 4.4.

4.4.3 Seznam dřevin

Většinu obsahu stránky se seznamem dřevin pokrývá tabulka, která obsahuje informace o dřevinách, které se získávají ze serveru pomocí koncového bodu aplikačního rozhraní `/api/v1/woodyPlants`. Pro tabulku byla využita komponenta `Table`[41]. V seznamu jsou zobrazeny základní informace o dřevinách, tedy *Název*, *Druh* a *Poznámka*. Tato funkcionalita splňuje funkční požadavek **R1**.

[37] https://github.com/prixladi/treeloc-frontend
[38] https://ant.design/
[39] https://ant.design/components/menu/
[40] https://www.npmjs.com/package/react-router-dom
[41] https://ant.design/components/table/

```
<Router>
  <NavBar />
  <Switch>
    <Route exact path="/">
      <Redirect to={_Map} />
    </Route>
    <Route exact path={_Map}>
      <MapPage />
    </Route>
    <Route exact path={_List}>
      <ListPage />
    </Route>
  </Switch>
</Router>
```

Obrázek 4.4: Konfigurace směrování v klientské aplikaci

Dále je, pro dřeviny obsahující souřadnice, v záložce akce možno využít tlačítko na zobrazení dřeviny v mapě. Více o mapě je napsáno v další sekci.

Filtrace a řazení

V seznamu dřevin je možno filtrovat a řadit. Řadit lze podle polí *Název*, *Druh* i *Poznámka*. Řadit lze vzestupně i sestupně a konfiguruje se pomocí šipek, které se nacházejí v hlavičce tabulky u názvů sloupců. Pro filtraci se využívá vyhledávání na základě textu zadaného do textového pole nacházejícím se nad seznamem. Toto vyhledávání na serveru využívá textový index popsaný v sekci 4.3.4, vyhledává tedy nad poli *Název*, *Druh* a *Poznámka*. Tyto funkcionality splňují funkční požadavky **R2** a **R3**.

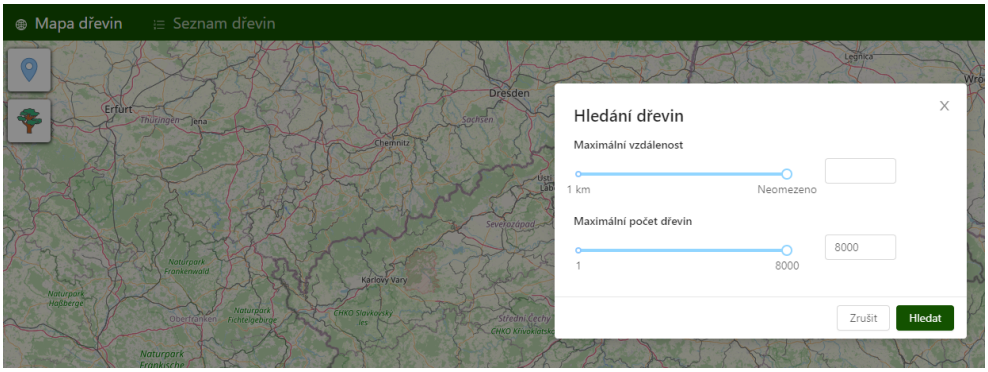
4.4.4 Mapa dřevin

Jak je zmíněno v návrhu aplikace, tak pro práci s mapou byla zvolena knihovna Leaflet. Pro samotnou mapu byly zvolené mapové dlaždice z OpenStreetMap[42].

Poloha

Na mapě se nachází polohu určující značka, která se nachází ve středu České republiky. Po příchodu na stránku se aplikace pokusí zjistit uživatelovu polohu. V případě, že uživatel svolí ke zjištění polohy, značka se přesune na zjištěnou polohu a načtou se data s výchozím filtrem. Pro zjištění uživatelovy

[42] www.openstreetmap.org



Obrázek 4.5: Ukázka vyhledávání v mapě z aplikace TreeLoc

polohy byl využit balíček `react-geolocated`[43]. Tato funkcionalita splňuje funkční požadavek **R5**.

Uživatel může libovolně měnit polohu na mapě. Slouží k tomu první tlačítko v levém horním rohu, které je vidět na obrázku 4.5. Po stisku tlačítka je možno kliknout na místo v mapě a značka určující polohu se přesune.

Vyhledávání

Vyhledávání je možno nastavit pomocí dialogového okna, které je vidět na obrázku 4.5. V okně je možno zvolit vzdálenost, ve které budou dřeviny hledány a počet dřevin, kolik má být vráceno. Po stisknutí tlačítka s nápisem *Hledat* se pošle dotaz na server s filtrem sestaveným z vybraných možností. Zobrazení okna s vyhledáváním je možno vyvolat pomocí kliknutí na druhé tlačítko v levém horním rohu, které lze vidět na obrázku 4.5. Tyto funkcionality splňují funkční požadavky **R6** a **R7**.

Data

Data jsou v mapě zobrazena v závislosti na svém prostorovém typu. Body se na mapě zobrazí pomocí ikon stromu. Lomené čáry se zobrazí jako zelené čáry a mnohoúhelníky se zobrazí jako zelené plochy. Tato funkcionalita splňuje funkční požadavek **R4**.

V případě, že by bylo hodně dat na jednom místě, mapa by se mohla stát značně nepřehlednou. Z tohoto důvodu bylo zvoleno rozšíření pro Leaflet `Leaflet.markercluster`[44], které umí shlukovat body do skupin, které se při přiblížení expandují na menší skupiny, případně na již samostatné objekty. `Leaflet.markercluster` umí shlukovat pouze body. Z tohoto důvodu

[43] https://www.npmjs.com/package/react-geolocated
[44] https://github.com/Leaflet/Leaflet.markercluster

bylo využito rozšíření `Leaflet.Deflate`[45], které umí převádět lomené čáry a mnohoúhelníky na body.

Každá dřevina v mapě má přiřazené vyskakovací okno s informacemi o dřevině, které se otevře po kliknutí na dřevinu.

4.4.5 CI

Stejně jako pro serverový projekt bylo u klientského projektu pomocí Github Actions vytvořeno CI. Princip je stejný jako u serverové části v sekci 4.3.9, ale s rozdílem, že existuje jen jedna služba a pro tu byl zvolen Docker Hub repozitář `treeloc/frontend`.

[45] https://github.com/oliverroick/Leaflet.Deflate

Kapitola 5

Data a demo aplikace

V této kapitole se práce zabývá transformací podmnožiny existujících dat do podoby vzniklé OFN a následným využitím dat pro demo aplikaci.

5.1 Transformace dat

Nejprve byla exportována data ze sekce památných stromů z Ústředního seznamu ochrany přírody[46]. Jedná se zhruba o osm tisíc dřevin, z nichž dva a půl tisíce obsahuje souřadnice. Data byla transformována pomocí konzolového projektu, který autor vytvořil přímo pro tento účel.

Projekt je dostupný ve větvi `transform`[47] v repozitáři se serverovou částí aplikace. Projekt byl přímo zakomponován do existující webové aplikace, aby mohl využívat již existující datové modely, které byly v aplikaci vytvořeny, a nebylo tedy nutno vytvářet nové. Větev již není udržována, takže webová aplikace v této větvi je zastaralá, ale datové modely jsou stále aktuální.

Data byla přetransformována do formátu JSON. Bylo vytvořeno čtrnáct JSON souborů, každý obsahující data pro jeden z krajů České republiky. Pro data byl vytvořen GitHub repozitář `prixladi/treeloc-data`[48], který na své kořenové úrovni obsahuje všechny JSON soubory s daty.

5.2 Demo aplikace

Jak je popsáno výše v sekci 4.3.5 s nahráváním dat, aplikace dostane předanou URL adresu, ze které získává adresy datových sad. V produkčním běhu si aplikace bude data stahovat ze SPARQL[49] koncového bodu NKODu. Dotaz se pošle jakožto HTTP GET a SPARQL dotaz se zakóduje do parametru.

[46] https://drusop.nature.cz/ost/chrobjekty/pstromy/
[47] https://github.com/prixladi/treeloc-backend/tree/transform
[48] https://github.com/prixladi/treeloc-data
[49] https://www.w3.org/TR/rdf-sparql-query/

Jelikož ale zmíněný koncový bod není v současné době nasazen, tak jsou jako ukázková data použita přetransformovaná data popsaná v předchozí sekci. Do repozitáře `prixladi/treeloc-data` byl přidán soubor `Discovery.csv`, který simuluje odpověď SPARQL koncového bodu a vrací seznam URL adres datových sad, které v tomto případě jsou JSON soubory přetransformovaných dat v stejném repozitáři.

Ve výchozím nastavení aplikace využívá jako datový zdroj právě výše popsaný soubor `Discovery.csv`. Aplikace tedy tímto způsobem simuluje postupné přidávání datových sad poskytovateli dat. Jelikož je služba **TreeLoc.Loader** parametrizovatelná, je možno nastavit prodlevy mezi načítáním jednotlivých datových sad. Samotné spuštění aplikace je popsáno v sekci administrátorské dokumentace 6.3.

Pro produkční nasazení aplikace stačí změnit proměnnou prostředí s adresou koncového bodu na adresu SPARQL koncového bodu NKODu se zakódovaným dotazem.

Kapitola 6

Dokumentace

Tato kapitola se zabývá dokumentací webové aplikace. Obsahuje dokumentace ze tří různých pohledů, tedy uživatelského, programátorského a administrátorského.

6.1 Uživatelská dokumentace

V této sekci je aplikace zdokumentována z pohledu uživatelského. Obsahuje popis jednotlivých funkcionalit, které může uživatel využívat při používání aplikace.

6.1.1 Testovací aplikace

Testovací běh aplikace je dostupný na `https://treeloc.azurewebsites.net/`. Jedná se o služby nasazené v cloudovém prostředí Azure. Služby běží na sdíleném stroji a využívají autorovo *Azure for students* předplatné, které má jen omezené množství zdrojů.

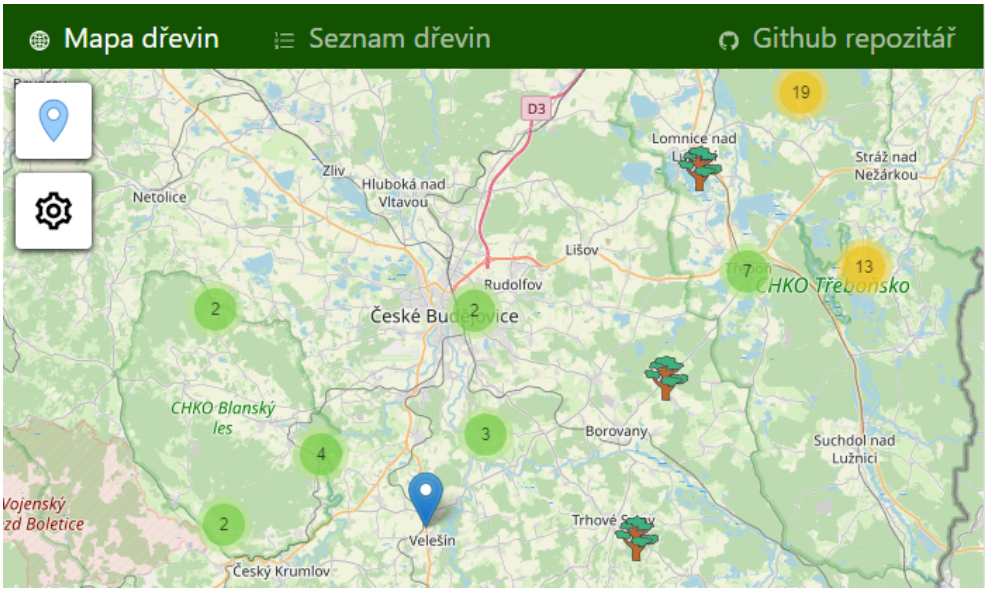
Databáze není nasazena jakožto samostatný kontejner, ale je využit sdílený cluster na MongoDB Atlas [30], který je zdarma a má jen minimální výkon.

6.1.2 Mapa

Mapa je výchozí stránka aplikace. Na vrcholu stránky se nachází menu společné pro všechny stránky aplikace, které slouží k přechodu mezi jednotlivými stránkami a na stránku repozitáře projektu. Ukázka stránky s mapou je na obrázku 6.1.

Při prvním příchodu do aplikace je uživatel vyzván k povolení zjištění polohy. Po potvrzení mu jsou načtena data s výchozím filtrem.

V levém horním rohu se nacházejí dvě tlačítka:

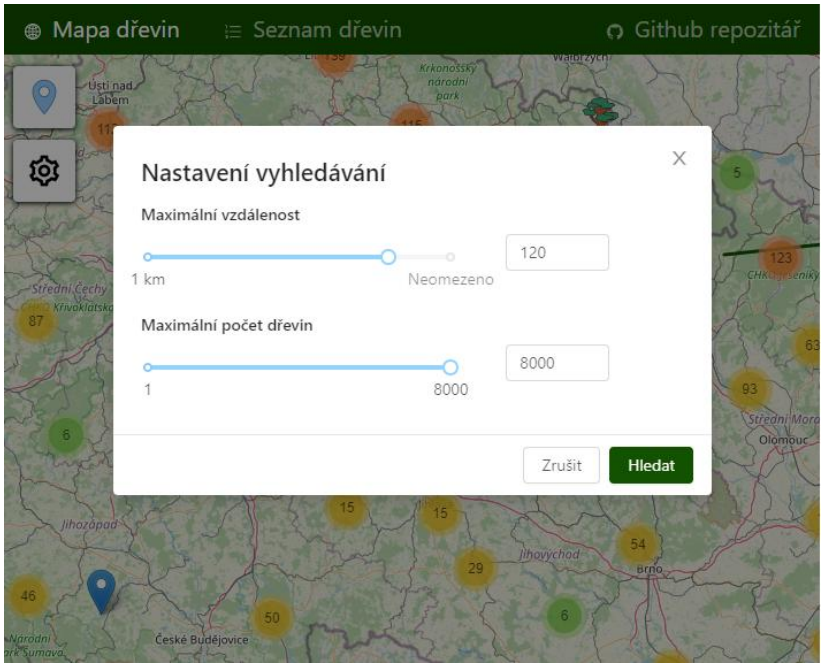


Obrázek 6.1: Ukázka stránky s mapou z aplikace TreeLoc

- První tlačítko odshora umožňuje měnit uživatelovu polohu v mapě. Po kliknutí na tlačítko se změní kurzor na mapě a uživatel může kliknout na libovolné místo v mapě a značka určující polohu se přesune na toto místo.
- Druhé tlačítko odshora otevře dialogové okno, které umožňuje měnit parametry vyhledávání dřevin v mapě. Toto okno je popsáno dále v této sekci.

Mapu jde klasicky přibližovat a oddalovat. Pokud se na mapě nachází více dřevin v malé vzdálenosti od sebe, tak jsou body seskupeny do skupin, které na sobě mají číslo indikující počet dřevin nacházejících se ve skupině. Skupiny mají různou barvu podle počtu dřevin, které obsahují. Pokud se uživatel chce dostat k samostatným dřevinám, je možno přibližovat mapu, dokud se skupiny nerozdělí na menší skupiny, případně samostatné dřeviny. Nebo je možné kliknout na skupinu dřevin, která se následně expanduje na menší skupiny nebo samostatné dřeviny a přiblíží mapu.

Uživatel může měnit parametry hledání pomocí dialogového okna s nastavením. V nastavení lze měnit vzdálenost vyhledávání. Výchozí hodnota pro vzdálenost je **neomezeno**. Dále je možno vzdálenost nastavit na **1** až **150** kilometrů od současné polohy. Další parametr je počet dřevin, který lze nastavit od **1** do **8 000**, kde **8 000** je výchozí hodnota. Vyhledávání probíhá vždy od současné uživatelovy polohy. Ukázku dialogového okna s nastavením vyhledávání dřevin na mapě lze vidět na obrázku 6.2.



Obrázek 6.2: Ukázka dialogového okna s nastavením hledání dřevin v mapě z aplikace TreeLoc

Po kliknutí na dřevinu v mapě se u dřeviny zobrazí okno s informacemi, jako například název, poznámka a URL adresy obrázků, pokud nějaké má. Dále se zobrazí vzdálenost od polohy, od které byly dřeviny vyhledávány.

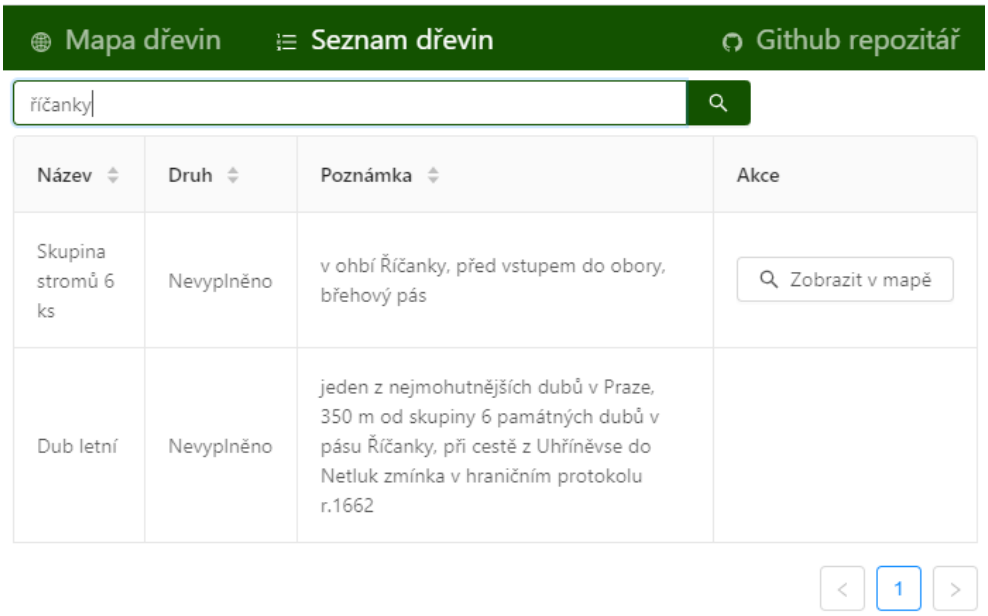
6.1.3 Seznam dřevin

Seznam dřevin se skládá z vyhledávacího pole na vrcholu stránky hned pod navigačním menu, tabulky s informacemi o dřevinách a stránkovacího menu v pravé spodní části stránky. Ukázku stránky se seznamem dřevin lze vidět na obrázku 6.3.

Po příchodu na stránku se načte první stránka dat bez jakéhokoliv filtru a s výchozím řazením. V tabulce se nacházejí informace o dřevinách a případné akce.

Uživatel může dřeviny řadit podle polí *Název*, *Druh* a *Poznámka*. Řazení probíhá pomocí kliknutí na šipky nacházející se vpravo od příslušného pole v hlavičce tabulky. Jedno kliknutí zapne vzestupné abecední řazení, druhé kliknutí zapne sestupné abecední řazení, třetí kliknutí řazení vypne. Šipky následně indikují, jaké řazení je zapnuté.

Filtrace je možná pomocí textového pole nad seznamem dřevin. Uživatel



Obrázek 6.3: Ukázka seznamu dřevin z aplikace TreeLoc

může vyhledat požadovaná slova, která mají některá ze zobrazovaných polí *Název*, *Druh* nebo *Poznámka* obsahovat. Uživatel napíše požadovaný text do textového pole a následně klikne a ikonu hledání vedle textového pole nebo použije klávesu *enter*.

V případě, že má dřevina vyplněné souřadnice, tak se ve sloupci akce zobrazí tlačítko s textem *Zobrazit v mapě*. Po kliknutí na tlačítko bude uživatel přesměrován na stránku s mapou a zobrazí se příslušná dřevina a dřeviny okolo podle výchozího filtru. Dřevina, u které bylo stisknuto tlačítko pro zobrazení v mapě, má automaticky otevřené okno s informacemi, aby bylo jasné, o kterou ze zobrazených dřevin se jedná.

6.2 Programátorská dokumentace

V této sekci se nachází programátorská dokumentace serverové a klientské části aplikace.

6.2.1 Spuštění serverové části

Pro sestavení a spuštění projektu existují dva způsoby.

Docker

První a preferovaný způsob je pomocí Dockeru a programu Docker Compose. Projekt se dá spustit z příkazové řádky pomocí příkazu:

```
$ docker-compose up
```

Tento příkaz sestaví dockerové obrazy a následně vytvoří a spustí dockerové kontejnery. V případě opětovného spuštění již nedojde k sestavování obrazů, pouze se vytvoří a spustí kontejnery. Pokud tedy dojde ke změně zdrojového kódu, je potřeba znovu vytvořit obraz. Toho lze docílit přidáním přepínače *build*. Příkaz tedy bude vypadat takto:

```
$ docker-compose up --build
```

Dále je možno projekt spouštět ve Visual Studiu a to nastavením výchozího projektu na projekt `Docker-compose`, který je popsán v sekci 4.3.2 se strukturou projektů, a následným spuštěním. Projekt je možno spouštět v konfiguraci pro nasazení i pro lazení.

Po spuštění budou služby dostupné na adrese `localhost` a portech, které jsou nastavené v konfiguraci v souboru `docker-compose.yml`, který se nachází v kořenovém adresáři projektu. Ve výchozím nastavení se jedná o porty `4545` pro službu **TreeLoc.Api** a `4546` pro službu **TreeLoc.Loader**. Pokud porty nejsou k dispozici, je potřeba je v konfiguračním souboru změnit.

.NET Core SDK

Druhý způsob je sestavení pomocí .NET Core SDK v3.1.0, které je ke stažení na stránkách Microsoftu[50]. Tento postup zde není podrobně rozepsán, jelikož preferovaný způsob spouštění projektu je pomocí Dockeru.

6.2.2 Dokumentace serverové části

Na obrázku 6.4 se nachází struktura adresářů serverové části.

[50] https://dotnet.microsoft.com/download/dotnet-core/3.1

- `.github` ..... Adresář obsahující definice pro Github Actions
- `src` ..... Adresář se soubory se zdrojovým kódem
  - `TreeLoc.Api` ..... Adresář obsahující stejnojmenný projekt
  - `TreeLoc.Api.UnitTests` .. Adresář obsahující testy k stejnojmennému projektu
  - `TreeLoc.Core` ..... Adresář obsahující stejnojmenný projekt
  - `TreeLoc.Core.UnitTests` . Adresář obsahující testy k stejnojmennému projektu
  - `TreeLoc.Database` ..... Adresář obsahující stejnojmenný projekt
  - `TreeLoc.Loader` ..... Adresář obsahující stejnojmenný projekt
  - `TreeLoc.Loader.UnitTests` ..... Adresář obsahující testy k stejnojmennému projektu
- `tests` . Adresář obsahující soubory ke spuštění testů a generování pokrytí kódu

Obrázek 6.4: Adresářová struktura serverové části aplikace

Celá aplikace je napsaná podle SOLID principů [31]. Jedná se o principy, jejich dodržováním se kód stává čitelnější, snadněji rozšiřitelný a snadněji udržovatelný. Jedním z těchto principů je *Single-responsibility principle*. Tento princip říká, že třída by měla být zodpovědná jen za jednu činnost, pro kterou byla navržena.

TreeLoc.Database

Výchozím adresářem je `src/TreeLoc.Database`.

Projekt obsahuje definice databázových dokumentů. Aby se dal nový dokument pohodlně používat ve zbytku aplikace, musí dědit od abstraktní třídy `DocumentBase` a musí být dekorován atributem `DbCollectionAttribute`, který určuje název kolekce dokumentu.

TreeLoc.Core

Výchozím adresářem je `src/TreeLoc.Core`.

Projekt obsahuje společné třídy pro projekty **TreeLoc.Api** a **TreeLoc.Loader**. Nejdůležitější třídou je abstraktní třída `RespositoryBase<T>`, kde `T` je konkrétní databázový dokument, ke kterému bude mít třída přístup. Třída poskytuje klasické CRUD operace nad kolekcí dokumentů.

TreeLoc.Api

Výchozím adresářem je `src/TreeLoc.Api`.

Projekt na své kořenové úrovni obsahuje třídu `Startup`, ve které dochází k nastavení *dependency injection*, což je mimo jiné jeden ze SOLID principů [31]. Dále tato třída konfiguruje, jakým způsobem se budou zpracovávat příchozí HTTP požadavky. Příklad takovéto konfigurace je vidět na obrázku 6.5.

```
public void Configure(IApplicationBuilder app)
{
  app.UseRouting();
  app.UseCors(CorsConfig.Setup);

  app.UseExceptionHandling();

  app.UseEndpoints(builder =>
  {
    builder.MapControllers();
  });

  app.UseSwagger(SwaggerConfig.SetupSwagger);
  app.UseSwaggerUI(SwaggerConfig.SetupSwaggerUI);
}
```

Obrázek 6.5: Konfigurace zpracování HTTP požadavků

Další důležitou součástí jsou třídy zvané `Controllery`, které se nacházejí v adresáři `Controllers`. Tyto třídy definují cesty ke koncovým bodům aplikačního rozhraní a modely, které tyto koncové body přijímají.

Základem tohoto projektu je zpracování požadavků přicházející na aplikační rozhraní. Pro každý HTTP požadavek se vytvoří třída implementující rozhraní `IRequest<T>` z knihovny MediatR, kde `T` je návratová hodnota, a umístí se do adresáře `Requests`. Tato třída slouží jakožto třída pro přenos příchozí informace. Při přijetí HTTP požadavku se odešle nová instance příslušné třídy implementující `IRequest<T>` a zpracování dále probíhá pomocí třídy implementující rozhraní `IRequestHandler<T>`.

Pro zpracování požadavku se vytvoří třída `IRequestHandler<T>`, kde `T` je třída implementující `IRequest<T>`, a umístí se do adresáře `Handlers`. Tato třída obsahuje byznys logiku aplikace.

Přístup k databázi se provádí přes třídy, které dědí od abstraktní třídy `RespositoryBase<T>`.

Ve třídách v adresáři `IoC` se nachází konfigurace *dependecy injection*, kterou využívá třída `Startup`.

Ostatní třídy zahrnují modely, objekty pro přenos dat, konfigurační třídy, transformační třídy atd.

TreeLoc.Loader

Výchozím adresářem je `src/TreeLoc.Loader`.

Stejně jako `TreeLoc.Api` obsahuje konfigurační třídu `Startup`.

Většina logiky této služby se nachází v třídách v adresáři `Services`. Třída `DiscoveryService` je zodpovědná za získávání adres datových sad. Třída `LoaderService` slouží ke stahování datových sad.

V adresáři `SignalR` se nachází třída definující SignalR koncový bod `/signalr` a rozšiřující funkce pro odesílání dat na řečený koncový bod.

Přístup k databázi se provádí přes třídy, které dědí od abstraktní třídy `RespositoryBase<T>`.

Dependecy injection se nastavuje ve třídách umístěných v adresáři `IoC`.

Aplikační rozhraní

Dokumentace aplikačního rozhraní je automaticky generována nástrojem Swagger z XML komentářů ve zdrojovém kódu, jak je popsáno v sekci 4.3.8. Uživatelské rozhraní dokumentace je tedy dostupné po spuštění aplikace na adrese a portu, na které běží služba `TreeLoc.Api` a koncovém bodu `/swagger`. U testovací aplikace, která běží v Azure, lze dokumentaci najít na adrese `https://treeloc-api.azurewebsites.net/swagger`.

6.2.3 Spuštění klientské části

Pro sestavení a spuštění projektu existují dva způsoby.

Docker

První způsob spuštění je pomocí Dockeru a programu Docker Compose. V případě, že se uživatel nachází v kořenovém adresáři projektu, tak se projekt dá spustit z příkazové řádky pomocí příkazu:

```
$ docker-compose up
```

Tento příkaz sestaví dockerový obraz a následně vytvoří a spustí dockerový kontejner. V případě opětovného spuštění již nedojde k sestavování obrazu, pouze se vytvoří a spustí kontejner. Pokud tedy dojde ke změně zdrojového kódu, je potřeba znovu vytvořit obraz. Toho lze docílit přidáním přepínače *build*. Příkaz tedy bude vypadat takto:

```
$ docker-compose up --build
```

Po spuštění bude aplikace dostupná na adrese `localhost` a portu, který je nastavený v konfiguraci v souboru `docker-compose.yml`, který se nachází v kořenovém adresáři projektu. Ve výchozím nastavení se jedná o port `80`. Z tohoto důvodu je potřeba, aby byl port `80` k dispozici, případně je ho potřeba v konfiguraci změnit.

Yarn

Druhý způsob spuštění je pomocí správce balíčků Yarn[51]. V případě, že se uživatel nachází v kořenovém adresáři projektu, tak lze aplikaci spustit sekvencí příkazů:

```
$ yarn
$ yarn start
```

Touto cestou bude aplikace spuštěna na adrese `localhost` a portu `3000`. Příkaz `yarn start` běží v interaktivním režimu, tedy v případě nedostupnosti portu `3000` bude uživatel vyzván ke spuštění na jiném portu, který mu bude nabídnut. Jedná se o neoptimalizované sestavení, které je určené pro vývoj. Pro sestavení určené pro nasazení aplikace do produkce se použije sekvence příkazů:

```
$ yarn
$ yarn build
```

[51] https://yarnpkg.com/

Tyto příkazy vytvoří optimalizované sestavení, které je možné nasadit na webový server. Tento postup již není dále rozebírán, jelikož preferované nasazení pro tuto aplikaci je pomocí vytvořeného obrazu přes Docker.

6.2.4 Dokumentace klientské části

Na obrázku 6.6 se nachází struktura adresářů klientské části. Důležité adresáře a soubory jsou dále detailněji rozepsány. Pokud není řečeno jinak, komponenty se nacházejí ve stejnojmenném souboru s příponou `.tsx`.

- `.github` ..... Adresář obsahující definice pro Github Actions
- `config` ..... Adresář obsahující konfigurační soubory běhu aplikace
  - `docker` ..... Adresář s konfiguračními soubory pro běh v Dockeru
  - `nginx` ..... Adresář s konfiguračními soubory pro Nginx
- `public` ..... Adresář obsahující statické soubory
- `src` ..... Adresář se soubory se zdrojovým kódem
  - `Api` ..... Adresář obsahující logiku komunikace s aplikačním rozhraním
  - `Common` ..... Adresář obsahující společné komponenty a funkce
  - `Hooks` ..... Adresář obsahující nástroje pro správu stavu
  - `ListPage` ..... Adresář s komponentami pro stránku se seznamem
  - `MapControls` ..... Adresář s definicemi tlačítek na mapě
  - `MapPage` ..... Adresář s komponentami pro stránku s mapou
  - `Navbar` ..... Adresář s komponentami pro navigační menu
  - `Popups` ..... Adresář s komponentami pro vyskakovací okna na mapě
  - `Services` ..... Adresář s pomocnými funkcemi a modely

Obrázek 6.6: Adresářová struktura klientské části aplikace

config/docker

Adresář obsahuje javascriptový soubor `config.js`, který vytváří proměnnou držící proměnné prostředí.

config/nginx

Adresář obsahuje soubor `nginx.conf`, který slouží k nastavení webového serveru.

src/Api

V adresáři jsou typesriptové soubory s definicí koncových bodů aplikačního rozhraní a s funkcemi pro volání aplikačního rozhraní.

src/Common

V adresáři se nacházejí typescriptové soubory se společnými prvky pro všechny stránky, soubory s konstantami a soubory s pomocnými funkcemi.

src/Hooks

Tento adresář slouží jakožto úložiště pro společné Hooky[52]. Každý Hook má svůj vlastní typescriptový soubor.

- **useTitle** změní titulek stránky.
- **useUserMapMarker** zařídí přidání uživatelovy pozice do mapy.
- **useSignalR** zajišťuje připojení k SignalR koncovému bodu serveru.
- **useWoodyPlantsLoader** zajišťuje načítání dřevin podle předaného filtru.
- **useWoodyPlantsMapControl** zajišťuje přidání nastavení vyhledávání dřevin do mapy a následné vyhledávání dřevin dle vybraného filtru.

src/ListPage

Adresář obsahuje komponentu `ListPage`, která je definována v souboru `index.tsx` a slouží jakožto hlavní komponenta pro seznam dřevin.

Dále obsahuje komponentu `ListSearch`, kterou využívá `ListPage` komponenta.

Soubor `units.ts` obsahuje pomocné funkce pro `ListPage` komponentu, jako například funkci pro transformaci dat do seznamu.

src/MapControls

Adresář obsahuje typescriptové soubory se třídami pro tlačítka na mapě. `FindWoodyPlantsControl` je tlačítko pro nastavení vyhledávání dřevin a `SetPositionControl` je tlačítko pro hledání pozice.

src/MapPage

Adresář obsahuje komponentu `MapPage` v souboru `index.tsx`, která slouží jakožto komponenta pro vykreslení mapy.

Dále obsahuje komponentu `MapLogic`, sloužící k ovládání veškeré logiky mapy, kterou využívá komponenta `MapPage`. Například obsahuje přidání tlačítek pro ovládání nebo samotné načtení a přidání dat do mapy.

Další soubor `utils.ts` obsahuje pomocné funkce pro transformaci dat přijatých z aplikačního rozhraní na data zobrazitelná v mapě.

[52] https://reactjs.org/docs/hooks-intro.html

Komponenta `FindPlantsCard` definuje dialogové okno pro nastavení hledání dřevin.

src/NavBar

Adresář obsahuje komponentu `NavBar` v souboru `index.tsx`. Tato komponenta slouží k vykreslení navigačního menu na vrcholu obrazovky.

src/Popups

Adresář obsahuje typescriptový soubor pro vytváření vyskakovacích textových oken. V současnosti je jen jediné takové okno a to v mapě po kliknutí na dřevinu.

src/Services

Adresář obsahuje typescriptové soubor pro funkce usnadňující volání aplikačního rozhraní. Dále obsahuje typescriptový soubor pro definice typů modelů využívaných v aplikaci.

6.3 Administrátorská dokumentace

V této sekci je popsána dokumentace k administraci aplikace.

Jednotlivé části aplikace je možné spustit samostatně, jak je popsáno v sekcích 6.2.1 a 6.2.3. Tyto způsoby spouštění aplikace jsou vhodnější spíše jen pro vývoj. Pro nasazení aplikace na produkční prostředí jsou vhodnější metody nasazení popsané níže v této sekci.

6.3.1 Nasazení aplikace

Pro nasazení aplikace nebo pro simulaci produkčního běhu aplikace je vhodné použít repozitář na GitHubu `prixladi/treeloc`[53], který slouží jakožto rozcestník pro serverovou a klientskou část aplikace. Dále obsahuje návod ke spuštění v souboru `Readme.md` a obsahuje `docker-compose.yml` soubor, který slouží ke spuštění celé aplikace pomocí Docker Compose.

Požadavky pro spuštění aplikace jsou následující:

- **Git** – Git je zapotřebí k naklonování repozitáře `prixladi/treeloc`. Tento krok není nutné provádět, repozitář stačí stáhnout nebo jen zkopírovat soubor `docker-compose.yml`, který se nachází v kořenovém adresáři repozitáře. Git je ke stažení zde `https://git-scm.com/downloads`.
- **Docker** – Docker je zapotřebí k celému běhu aplikace. Docker je ke stažení zde `https://docs.docker.com/get-docker/`.
- **Docker Compose** – Docker Compose je potřeba k zapnutí aplikace. V některých instalacích Dockeru, například té pro Windows, není potřeba Docker Compose instalovat, protože se nainstaluje zároveň s Dockerem. V případě, že je Docker Compose potřeba doinstalovat zvlášť, tak je ke stažení zde `https://docs.docker.com/compose/install/`.

V případě splnění všech požadavků pro spuštění je potřeba na stroj, na kterém se bude aplikace spouštět, dostat soubor `docker-compose.yml` z `prixladi/treeloc` repozitáře. Soubor se může stáhnout, zkopírovat nebo případně naklonovat s celým repozitářem příkazem:

```
$ git clone https://github.com/prixladi/treeloc.git
```

V případě, že se administrátor nachází v adresáři, který obsahuje soubor `docker-compose.yml`, a má spuštěný Docker, aplikace se spustí příkazem:

```
$ docker-compose up
```

[53] https://github.com/prixladi/treeloc

Když vše proběhne v pořádku, klientská aplikace bude dostupná na adrese `localhost` a portu `80`. Aplikační rozhraní bude dostupné na stejné adrese a portu `8080`.

V případě, že se do dockerového repozitáře dostane nová verze obrazu, je potřeba před spuštěním aplikace tuto novou verzi stáhnout, jelikož pokud Docker najde obraz lokálně, tak se již nesnaží stahovat novější verzi. Nové verze obrazů se získají příkazem:

```
$ docker-compose pull
```

Spouštět aplikaci je možno i bez pomoci uvedeného `docker-compose.yml` souboru. Obrazy aplikace se nacházejí ve veřejných Docker Hub repozitářích, jak je popsáno v sekcích 4.3.9 a 4.4.5. Jsou tedy volně ke stažení. Záleží tedy na prostředí, kam se aplikace nasazuje. Jediné, co je potřeba ke správnému běhu kontejnerů s aplikací, jsou správě nastavené proměnné prostředí, které jsou popsané pro každou část aplikace v příslušných sekcích kapitoly implementace aplikace.

V případě nasazení do Azure App Service[54] stačí jen vybrat nasazení přes dockerové kontejnery, nastavit názvy Docker Hub repozitářů a proměnné prostředí.

Kontejner s databází není třeba nasazovat společně s aplikací, ale je možno využít některé existující služby poskytující MongoDB a na serveru správně nastavit proměnné prostředí s URL adresou databáze.

[54] https://azure.microsoft.com/services/app-service/

Kapitola 7

Testování

Tato kapitola se zabývá testováním webové aplikace. Nejprve jsou rozepsány unit testy serverové části a následně uživatelské testování celé aplikace.

7.1 Unit testy

Unit testy mají za úkol otestovat nejmenší testovatelné části aplikace, tedy v rámci objektových jazyků se jedná o třídy a jejich metody. Pro účel jednoho testu se vytváří instance testované třídy a místo jejích závislostí se jí předávají takzvané *Mock* instance, které mají funkcionalitu nadefinovanou přímo pro daný test.

Pro serverovou část byly vytvořeny projekty pro unit testy využívající testovací framework xUnit. Framework xUnit je nejnovější technologií pro unit testy v .NET aplikacích [32].

7.1.1 Pokrytí kódu

Pro zjištění pokrytí kódu testy byl přidán balíček `coverlet`[55], který pomocí přepínačů při pouštění testů dokáže vygenerovat XML soubory s pokrytím kódu. Dále existuje multiplatformní .NET nástroj `reportgenerator`[56], který pomocí vygenerovaných XML souborů s pokrytím kódu vytvoří HTML soubory, které dohromady vytváří uživatelské rozhraní s detailním popisem pokrytí zdrojového kódu.

Do serverového projektu byl přidán soubor `tests/Dockerfile`, který při vytváření obrazu spustí testy s generováním pokrytí. Dále za pomoci nástroje `reportgenerator` vytvoří HTML soubory s pokrytím kódu, které po spuštění jakožto kontejner uloží na disk hostujícího stroje a následně ukončí činnost.

[55] https://github.com/coverlet-coverage/coverlet
[56] https://danielpalme.github.io/ReportGenerator/

Dále byl přidán soubor `tests/docker-compose.yml`, který obsahuje definici pro kontejner vzniklý z výše popisovaného obrazu a dále vytvoří kontejner z obrazu `nginx:1.12-alpine`, který spustí webový server využívající HTML soubory s pokrytím kódu vygenerované kontejnerem s testy.

Spuštění testů a webového serveru je možné v případě, že se uživatel nachází v kořenové adresáři serverové části projektu, příkazem:

```
$ docker-compose -f ./tests/docker-compose.yml up --build
```

Uživatelské rozhraní je následně dostupné na adrese `localhost` a na portu `80`. Mapování portů se dá změnit v souboru `tests/docker-compose.yml`.

Současné pokrytí

Procenta pokrytí serverové části aplikace lze vidět v tabulce 7.1

Projekt	Pokrytí řádek [%]	Pokrytí větví [%]
TreeLoc.Api	96.3	72.8
TreeLoc.Loader	94.4	64.7
TreeLoc.Core	97.5	75
TreeLoc.Database	84.8	/
Vše dohromady	95.5	72

Tabulka 7.1: Tabulka pokrytí serverové části testy

Lze tedy vidět, že pokrytí řádek je na velmi dobré úrovni a blíží se sto procentům, ale dalo by se zapracovat na pokrytí rozvětvení kódu.

7.2 Uživatelské testování

Aplikace byla otestována čtyřmi uživateli. Pro uživatele byl sestaven seznam úkonů, které měl vykonat.

- **Vyhledání dřeviny podle jména** – Uživatel dostal název dřeviny a musel zjistit, zda dřevina existuje. Všichni uživatelé dostali existující dřevinu.
- **Vyhledání padesáti nejbližších dřevin od centra Prahy** – Uživatel musel v mapě zobrazit padesát nejbližších dřevin od centra Prahy a žádné jiné.
- **Vyhledání všech dřevin v oblasti padesáti kilometrů od současné polohy** – Uživatel musí v mapě zobrazit dřeviny v oblasti padesáti kilometrů od jeho současné polohy a žádné jiné.

Dále je popsán samotný průběh testování jednotlivými uživateli. Každý uživatel dostal jen základní informace, o jakou aplikaci se jedná a následně začal postupně plnit úkoly.

Muž 43 let

Testujícím byl muž ve věku 43 let. Osobní počítač a internet používá každý den, ale jen pro rekreační účely.

Před vyhledáváním dřeviny podle jména uživatel nejprve prozkoumával výchozí stránku, tedy stránku s mapou. Následně si v menu všiml seznamu dřevin a překlikl na něj. Ihned vyplnil vyhledávací textové pole a nalezl dřevinu.

Při vyhledávání dřevin v okolí Prahy uživatel nejprve otevřel, pomocí tlačítka v levém horním rohu, dialogové okno s nastavením vyhledávání a několikrát zkusil změnit parametry. Následně si všiml tlačítka pro změnu polohy a několikrát na něj klikl, než si uvědomil, že po prvním kliknutí se změní kurzor a uživatel může změnit polohu. Po tomto zjištění již uživatel bez problémů úlohu zvládl.

Vyhledávání dřevin v oblasti padesáti kilometrů uživatel zvládl bez sebemenších problémů, protože již byl obeznámen s nastavením vyhledávání.

Muž 23 let

Testující byl muž ve věku 23 let. Osobní počítač a internet využívá každý den pro rekreační i profesní účely.

Při vyhledávání dřeviny podle jména uživatel ihned přešel na seznam dřevin. Do textového pole nejprve zadal jen kořen vyhledávaného slova a aplikace mu nevrátila žádný výsledek. Následně doplnil celý vyhledávaný text a aplikace mu vrátila vyhledávanou dřevinu.

Vyhledávání nejbližších dřevin a vyhledávání dřevin v zadaném okruhu zvládl uživatel zcela bez problémů, jen s mírným zdržením při prozkoumávání funkcionalit mapy.

Žena 39 let

Testující byla žena ve věku 39 let. Osobní počítač používá jen minimálně a to jen pro jednoduché úkony.

Při vyhledávání dřevin podle názvu se testující zdržela na mapě dřevin a nevěděla, jak se dostat do seznamu dřevin. Po lehké pomoci se testující přesunula do seznamu dřevin a objevila textové pole pro filtraci. Po počátečních obtížích již bez problému našla hledanou dřevinu.

Při vyhledávání nejbližších dřevin testující opět potřebovala pomoc, tentokrát s nalezením tlačítka pro změnu polohy a pro nastavení vyhledávání.

Při vyhledávání dřevin v zadaném okruhu již testující neměla problém, jelikož dialog pro nastavení hledání znala z předchozího kroku.

Žena 20 let

Testující byla žena ve věku 20 let. Osobní počítač používá pro rekreační účely a pro účely studia.

Při vyhledávání podle názvu testující nejprve chvíli zůstala na mapě, ale následně našla horní menu, ve kterém přešla na seznam dřevin. Po chvilkovém prozkoumávání uživatelského rozhraní narazila na textové vyhledávací pole a začala vyhledávat dřevinu, ale stejně jako druhý testující zadala jen kořen slova a aplikace nic nenašla. Následně zkusila zadat celý text a aplikace našla správnou dřevinu.

Při vyhledávání nejbližších dřevin testující chvíli hledala tlačítko pro změnu polohy. To se jí bez pomoci podařilo najít, ale měla problém se změnou současné pozice a to opět z důvodu opakovaného klikání na tlačítko pro změnu pozice místo kliknutí do mapy.

Hledání dřevin v zadaném okruhu již testující nečinilo žádné problémy.

7.2.1 Výsledky uživatelského testování

V této sekci se rozebírají výsledky uživatelského testování a jaké problémy byly tímto testováním odhaleny.

Celkově se uživatelé dokázali v aplikaci poměrně snadno orientovat. Většina úkonů trvala přiměřeně dlouhou dobu. Z testování ale vyplynulo několik problémů.

Textové vyhledávání

Z uživatelského testování vyplývá, že uživatelé mají problémy s vyhledáváním v seznamu podle textu. To je nejspíše zapříčiněno textovým indexem, který

neumí rozpoznávat kořeny slov a to z důvodu, že textový index v MongoDB nepodporuje českou lokalizaci.

Ideálním řešením tohoto problému by bylo využití jiné služby pro textovou indexaci. Například Elasticsearch[57] nebo Apache Solr[58]. Tato funkcionalita není nejdůležitější funkcí aplikace, z tohoto důvodu žádná taková služba využita nebyla a bylo zvoleno jednodušší řešení a to indexace rovnou v databázi, kde se ukládají celá data. A také tento problém není blokující, jelikož zadáním přesného nebo dostatečně blízkého výrazu se docílí požadovaného vyhledání.

Změna pozice v mapě

Většina uživatelů neměla problém s nacházením tlačítek pro ovládání mapy. Ale následně měli problém se změnou pozice v mapě. Toto je patrně zapříčiněno tím, že po kliku na tlačítko pro změnu pozice se jen změní kurzor na mapě a je potřeba ještě jednou kliknout do prostoru pro změnu pozice.

Toto by se dalo vyřešit vyzváním uživatele k učinění další kroku po kliku na tlačítko. Nejlépe za pomoci popisného okna, které se zobrazí po kliknutí na tlačítko.

Dialogové okno s nastavením vyhledávání

Uživatelé poukazovali na to, že vzhled okna pro vyhledávání jim kazí celkový dojem aplikace. Samotné okno prý působí rušivým dojmem a posuvníky pro změnu kvantity jsou nevzhledné.

Pro vyřešení tohoto problému by bylo třeba navrhnout nový design dialogového okna, který by byl méně rušivý.

[57] https://www.elastic.co/
[58] https://lucene.apache.org/solr/

Závěr

V rámci práce došlo k navržení a implementaci OFN pro dřeviny. Dále byla navrhnuta, vytvořena a otestována webová aplikace **TreeLoc**. Tato aplikace stahuje a zobrazuje v mapě a v seznamu data splňující navrženou OFN. Aplikace umožňuje filtraci a vyhledávání zobrazovaných dat. Aplikace je rozdělena na serverovou a klientskou část. Serverová část je implementována v jazyce C# ve frameworku ASP.NET Core. Klientská část je implementována v jazyce JavaScript a využívá knihovnu React.

Dále byla přetransformována část již existujících dat, aby splňovala vzniklou OFN. Existuje funkční demo aplikace, která tato data stahuje a simuluje produkční běh aplikace včetně postupného přidávání dat.

Za pomoci konfigurace přes proměnné prostředí může být aplikace napojena na reálný zdroj dat a tyto data začít stahovat.

I přes drobné nedostatky, které vyplynuly z uživatelského testování, ukazuje aplikace výhody poskytování dat ve formátu OFN. Vzniklou OFN pro dřeviny lze nalézt na webu otevřených dat[59].

Pro aplikaci vznikla dokumentace, která popisuje fungování a nasazení z různých pohledů.

Aplikace byla otestována jednak unit testy a bylo na ní provedeno i uživatelské testování, ze kterého vzešlo několik vylepšení, která mohou být do budoucna implementována.

Budoucí vývoj

OFN se do budoucna bude rozvíjet podle požadavků poskytovatelů. Poskytovatelé mohou návrhy na vylepšení OFN zapisovat přímo do *Issue sekce*[60] repozitáře OFN na GitHubu.

[59] https://ofn.gov.cz/dřeviny

[60] https://github.com/opendata-mvcr/otevrene-formalni-normy/issues

Co se týče webové aplikace, do budoucna by bylo možné doladit drobné nedostatky, které jsou rozebrané v sekci 7.2.1. Dále by bylo možné, v reakci na budoucí vývoj OFN, přidávat do aplikace další vlastnosti pro dřeviny.

Na serveru by bylo možno vylepšit nahrávání dat o kontrolu časové známky datové sady, aby bylo možno aktualizovat již existující datovou sadu.

Dále by bylo možno vylepšit odesílání informací o změně dat přes SignalR, jak je popsáno v sekci 4.3.5.

Literatura

[1] Jan Kucera, Dusan Chlapek, Jakub Klímek a Martin Necaský. Methodologies and Best Practices for Open Data Publication. In Martin Necaský, Jaroslav Pokorný a Pavel Moravec, editoři, *Proceedings of the Dateso 2015 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Neprivec u Sobotky, Jicin, Czech Republic, April 14, 2015*, svazek 1343 z *CEUR Workshop Proceedings*, strany 52–64. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1343/paper5.pdf.

[2] Jakub Klímek. DCAT-AP representation of Czech National Open Data Catalog and its impact. *J. Web Semant.*, 55:69–85, 2019. DOI: 10.1016/j.websem.2018.11.001. URL: https://doi.org/10.1016/j.websem.2018.11.001.

[3] AION CS, s.r.o. Zákon č. 106/1999 Sb. 2020. URL: https://www.zakonyprolidi.cz/cs/1999-106 (citováno 15. 05. 2020).

[4] Zach Gollwitzer. Introduction to Software Architecture (Monolithic vs. Layered vs. Microservices). 2019. URL: https://zachgoll.github.io/blog/2019/architecture-types/ (citováno 14. 05. 2020).

[5] Stack Overflow. Developer Survey Results 2019. 2019. URL: https://insights.stackoverflow.com/survey/2019 (citováno 05. 04. 2020).

[6] Microsoft. C# documentation. 2020. URL: https://docs.microsoft.com/en-gb/dotnet/csharp/ (citováno 14. 05. 2020).

[7] Microsoft. Dotnet Core documentation. 2020. URL: https://docs.microsoft.com/en-us/dotnet/core/ (citováno 14. 05. 2020).

[8] Microsoft. What is ASP.NET Core? 2020. URL: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core (citováno 16. 02. 2020).

[9] Patrick Fletcher. Introduction to SignalR. 2020. URL: https://docs.microsoft.com/cs-cz/aspnet/signalr/overview/getting-started/introduction-to-signalr (citováno 19. 04. 2020).

[10] MongoDB, Inc. Introduction to MongoDB. 2020. URL: https://docs.mongodb.com/manual/introduction/ (citováno 16. 02. 2020).

[11] MongoDB, Inc. MongoDB indexes. 2020. URL: https://docs.mongodb.com/manual/indexes/index.html (citováno 16. 02. 2020).

[12] Mozilla and individual contributors. A re-introduction to JavaScript. 2020. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript (citováno 16. 02. 2020).

[13] Microsoft. TypeScript. 2020. URL: http://www.typescriptlang.org/ (citováno 16. 02. 2020).

[14] Mozilla and individual contributors. Document Object Model (DOM). 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (citováno 16. 02. 2020).

[15] Facebook Inc. React. 2020. URL: https://reactjs.org/ (citováno 16. 02. 2020).

[16] Guil Hernandez. Getting Started with Create React App, 2020. URL: https://blog.teamtreehouse.com/getting-started-create-react-app-tool (citováno 16. 02. 2020).

[17] Facebook Inc. Introducing JSX. 2020. URL: https://reactjs.org/docs/introducing-jsx.html (citováno 16. 02. 2020).

[18] Vladimir Agafonkin. Leaflet. 2019. URL: https://leafletjs.com/ (citováno 05. 04. 2020).

[19] Docker, Inc. Docker overview. 2020. URL: https://docs.docker.com/engine/docker-overview/ (citováno 16. 02. 2020).

[20] Docker, Inc. Overview of Docker Compose. 2019. URL: https://docs.docker.com/compose/ (citováno 29. 03. 2020).

[21] Docker, Inc. What is container? 2020. URL: https://www.docker.com/resources/what-container (citováno 16. 02. 2020).

[22] Isaac Sacolick. What is CI/CD? Continuous integration and continuous delivery explained. 2020. URL: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html (citováno 03. 05. 2020).

[23] GitHub, Inc. Github Actions. 2020. URL: https://github.com/features/actions (citováno 03. 05. 2020).

[24] Microsoft. Welcome to the Visual Studio IDE. 2020. URL: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019 (citováno 16. 02. 2020).

[25] Microsoft. Visual Studio Code - Getting Started. 2020. URL: https://code.visualstudio.com/docs (citováno 16. 02. 2020).

[26] MongoDB, Inc. MongoDB .NET Driver. 2015. URL: https://mongodb.github.io/mongo-csharp-driver/ (citováno 29. 02. 2020).

[27] OpenAPI Initiative. OpenAPI Specification. 2020. URL: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md (citováno 29. 03. 2020).

[28] SmartBear Software. OpenAPI Specification. 2020. URL: https://swagger.io/about/ (citováno 29. 03. 2020).

[29] SmartBear Software. Swagger UI. 2020. URL: https://swagger.io/tools/swagger-ui/ (citováno 29. 03. 2020).

[30] MongoDB, Inc. MongoDB Atlas. 2020. URL: https://www.mongodb.com/cloud/atlas (citováno 10. 05. 2020).

[31] Bipin Joshi. *Beginning Solid Principles and Design Patterns for Asp.net Developers*. Apress, 2016. ISBN: 978-1484218471.

[32] .NET Foundation. About xUnit.net. 2020. URL: https://xunit.net/ (citováno 10. 05. 2020).

Dodatek A

Seznam použitých zkratek

API Application program interface

CRUD Create, Read, Update, Delete

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

NoSQL Not only SQL

OFN Otevřená formální norma

RDF Resource Description Framework

SPARQL Simple Protocol and RDF Query Language

SQL Structured Query Language

URL Uniform Resource Locator

XML Extensible Markup Language

YAML YAML Ain't Markup Language

Dodatek B

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy aplikace
    - `frontend` ..... zdrojové kódy klientské části aplikace
    - `backend` ..... zdrojové kódy serverové části aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF