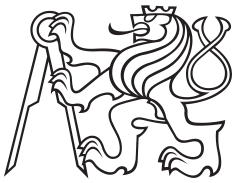
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Webové komponenty pro otevřené formální normy

Vojtěch Luňák

Vedoucí: Mgr. Miroslav Blaško, Ph.D.
Studijní program: Softwarové inženýrství a technologie
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Luňák** Jméno: **Vojtěch** Osobní číslo: **499152**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Webové komponenty pro otevřené formální normy

Název bakalářské práce anglicky:

Web components for open formal norms

Pokyny pro vypracování:

Otevřené formální normy (OFN) [1] jsou specifikace syntaktické a sémantické podoby datových sad publikovaných jako otevřená data, které se řídí moderními webovými standardy. Cílem práce je vytvoření webových komponent pro plnění dat dle OFN v rámci frameworku SForms [2]. SForms je javascriptová knihovna pro interaktivní webové formuláře. Komponenty by měly pomoct efektivně vyplňovat vybrané OFN, včetně zadání RÚIAN adresy [3, 5], adresního místa, konkrétní geo-lokace, výběr datumu a času.
Postup:
1) seznamte s technologiemi Sémantického webu pro reprezentaci (OWL, RDF, JSON-LD) a dotazovaní znalostí (SPARQL)
2) analyzujte stav OFN formulářů v aplikaci Správce OFN dat [4] z hlediska uživatelské přívětivosti a kvality nasbíraných dat
3) prozkoumejte relevantní řešení nalezených problémů
4) navrhněte a implementujte vybrané komponenty
5) implementované komponenty otestujte na vybraných OFN formulářích s alespoň třemi uživateli

Seznam doporučené literatury:

[1] Otevřené formální normy (OFN), online na https://data.gov.cz/ofn/
[2] SForms library, online na https://github.com/kbss-cvut/s-forms.
[3] RÚIAN, online na https://www.cuzk.cz/ruian/
[4] Správce dat Sémantického slovníku pojmů, online na https://github.com/opendata-mvcr/sgov-data-management.
[5] Tomáš Le, Application easing the search for addresses in the RÚIAN registry, 2020, online at https://dspace.cvut.cz/handle/10467/88341

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Mgr. Miroslav Blaško, Ph.D. skupina znalostních softwarových systémů

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **30.01.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Mgr. Miroslav Blaško, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Rád bych poděkoval vedoucímu mé práce, Mgr. Miroslavu Blaškovi, Ph.D., za odborné rady, věcné připomínky, a zájem a vstřícnost při konzultacích a diskuzích.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 26. května 2023

. . . . . .

Vojtěch Luňák

Abstrakt

Tato práce se zabývá vytvořením komponent pro SForms v kontextu otevřených formálních norem. SForms je javascriptová knihovna pro zobrazování interaktivních formulářů založená na frameworku React a technologiích sémantického webu. Otevřené formální normy jsou technická doporučení pro datové sady používaná zejména v české veřejné správě.

Vzniklé komponenty zefektivňují především zadávání adresy a zeměpisných souřadnic pomocí interakce s mapou a integrace Registru územní identifikace, adres a nemovitostí.

Klíčová slova: sémantický web, propojená data, React, TypeScript, SForms, vývoj softwaru, RÚIAN, webové formuláře

Vedoucí: Mgr. Miroslav Blaško, Ph.D.

Abstract

This thesis focuses on the creation of components for SForms in the context of open formal norms. SForms is a javascript library for displaying interactive forms based on the React framework and Semantic Web technologies. Open formal norms are technical recommendations for datasets used mainly in Czech public administration.

The resulting components streamline, in particular, the input of addresses and geographical coordinates through interaction with the map and integration of the Registry of Territorial Identification, Addresses and Real Estate.

Keywords: semantic web, linked data, React, TypeScript, SForms, software development, RÚIAN, web forms

Title translation: Web components for open formal norms

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Ve světě otevřeně dostupných dat je potřeba nastavit, jak otevřená data budou strukturovaná a jakým způsobem budou definované pojmy, které budou k poskytnutí. Pro účel definování pojmů slouží sémantický slovník pojmů. Ten zahrnuje pojmy, jejich definice a vazby na legislativu, vzájemné sémantické vazby pojmů mezi sebou i sémantické vazby pojmů na standardní veřejné slovníky používané v zahraničí. [Ne3] Otevřené formální normy jsou specifikace sloužící na výměnu dat, které jsou navázané na sémanticky slovník pojmů.

Formuláře otevřených formálních norem jsou formuláře nad otevřenými propojenými daty definovaných dle otevřených formálních norem[1].

Hotové slovníky jsou například pro témata *Sportoviště, Turistické cíle, Aktuality, Události*. Formulářům nad těmito daty se budu věnovat nejvíce.

Posláním otevřených formálních norem je sjednocení definic entit a jejich vztahů ve specifickém kontextu. To umožní interoperabilitu mezi objekty, které tato data používají, při dodržení těchto norem. A pro vizualizaci a uživatelsky pohodlnou práci s těmito daty pro sběr dle otevřených formálních norem slouží právě formuláře. Knihovnou zobrazující tato data ve formě formuláře je javascriptová knihovna SForms. Její výhodou je, že se jedná o obecnou knihovnu pro zobrazování formulářů nad daty ve formátu RDF. Jenže některé její vlastnosti se pro určitá využití nehodí. Jde mi v této práci o vytvoření rozšiřujících komponent pro knihovnu SForms, které budou realizovány jako samostatná TypeScriptová knihovna, s přihlédnutím na využití konkrétních dat a s přihlédnutím na uživatele, kteří s těmito daty pracují.

Typických uživatelů bude několik druhů. Obecně se jedná o úředníky, ale komponenta bude použitelná i pro širokou veřejnost zadávající adresu. Co se úředníku týče, pak už záleží jakou agendu úředník zastává:

- úředník zadávající turistický cíl
- úředník zadávající sportoviště
- úředník zadávající aktualitu
- úředník zadávající událost

[1] https://www.zakonyprolidi.cz/cs/1999-106p3a-3

Výše uvedená zadávaná témata mají některé stejné společné vlastnosti. Jedná se o objekty s umístěním a v některých případech i s adresou. Tato skutečnost je znázorněná v tabulce číslo 1.1 níže.

	Adresa	**Umístění / poloha**
Turistický cíl	ANO/NE	ANO
Sportoviště	ANO	ANO
Aktualita	NE	ANO
Událost	NE	ANO

Tabulka 1.1: Srovnání slovníků a jejich vlastností týkajících se adresy a umístění

1.1 Motivace

Existuje generátor, který ze slovníků otevřených formálních norem (dále v textu zkráceně OFN) generuje JSON-LD data ve formátu RDF. SForms umí RDF data vizualizovat ve formě interaktivního formuláře. Ovšem některá data jsou strukturovaná a definovaná tak, že SForms neumožňuje jejich jednoduché vyplnění. Je tedy žádoucí tuto knihovnu rozšířit takovým způsobem, aby se její základní použití co nejvíce zobecnilo. Tedy aby formulář poskytoval co nejjednodušší vyplnění pro co nejširší škálu dat.

1.2 Cíl práce

Cílem této bakalářské práce je vytvořit knihovnu React komponent, která rozšíří javascriptovou knihovnu SForms a zefektivní sběr dat vybraných OFN. Tím se docílí rychlejší a jednodušší práci uživatelů s formulářem. Důležité je, aby komponenta poskytovala tuto funkčnost i pro budoucí slovníky otevřených formálních norem a pomohlo se tím kvalitnímu a efektivnímu sběru dat.

Část I

Relevantní technologie a kontext práce

Kapitola 2
Technologie sémantického webu

Běžný internet v dnešní době je ideální prostředí pro výměnu informací a dat mezi lidmi. Webové stránky jsou v principu dokumenty, většinou i s odkazy na další dokumenty - stránky. Nejčastěji jsou zobrazovány webovými prohlížeči, aby byly pro uživatele čitelné a jednoduše použitelné. V posledních letech se pozornost upíná směrem k sémantickému webu, webu propojených dat, aby byly stránky čitelné i pro stroje. To otevírá dvířka novým příležitostem práce s daty.

2.1 Sémantický web

Sémantický web je typ webu, který se soustředí na tvorbu "databáze" propojených dat - dat, která jsou jednoznačně identifikovaná, mají vztahy mezi sebou a jsou strojově pochopitelná. V klasickém webu je analýza stránky strojem náročná a nepřesná, ale v sémantickém webu, kde jsou data a jejich vztahy jasně definovány, je práce a analýza nad těmito strukturovanými daty možná.

2.2 RDF

RDF je zkratka pro anglické Resource Description Framework, česky přeneseně systém popisu zdrojů. Je to jazyk pro reprezentaci dat. RDF popisuje propojené entity unikátními URI - *universal resource identifier*. Velkou důležitost má i vztah mezi entitami, který je také definován URI. Tímto způsobem dokáže RDF jednoznačně popsat data a jejich vztahy mezi sebou.

Tuto reprezentaci propojení dat realizuje RDF takzvanými trojicemi - *subjekt + predikát + objekt*. Příkladem trojice je například "Sportoviště má adresu". Subjektem je *sportoviště*, predikát je *má* a objektem je *adresa*. A aby byl příklad kompletní, uvedu stejnou situaci s ukázkovými URI - viz obrázek 2.1. Ty nám zajistí jednoznačnost všech částí trojice.

Obrázek 2.1: Model ukázkového příkladu RDF

2.3 Propojená data

Propojená data (angl. Linked Data) je sada principů, které umožňují publikovat data na webu tak, že se mohou odkazovat na jiná data publikovaná tímto způsobem, a to jednotným způsobem, bez ohledu na zdroj ze kterého data pochází. Tuto sadu principů formuloval Tim Berners-Lee, vynálezce sítě WWW. Principy odpovídají tomu, jak funguje dnešní web dokumentů - webových stránek. Jsou 4 a zní takto:

1. Všechny entity se pojmenovávají pomocí IRI.
2. Používejte HTTP(S) IRI, aby mohli lidé a aplikace přistupovat k informacím o entitách pomocí stávajícího protokolu HTTP(S).
3. Když někdo na takové IRI přistoupí, poskytněte data o entitě v RDF.
4. Přidejte do dat odkazy na jiná, související data, aby se uživatelé mohli dozvědět více.[Kl3]

2.4 SPARQL

SPARQL je dotazovací jazyk nad daty formátu RDF. Tato data formují svou propojeností RDF grafy a jazyk SPARQL umí z těchto dat vybírat, umí kombinovat data z různých zdrojů, umí průniky a sjednocení.

Data:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Query:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

This query, on the data above, has one solution:

Query Result:

title
"SPARQL Tutorial"

Obrázek 2.2: Ukázkový příklad dotazu SPARQL, zdroj: `https://www.w3.org/TR/rdf-sparql-query/#WritingSimpleQueries`

2.5 JSON-LD

Speciální nadstavbou formátu JSON pro propojená data je formát JSON-LD (JavaScript Object Notation-Linked Data). Ten je výhodný v tom, že je založený na velice používaném formátu, takže lidé jsou schopni takto formátovaná data číst i bez hlubší znalosti propojených dat. Stejně tak existuje mnoho již existujících knihoven a řešení, které pracují se syntaxí formátu JSON, takže zaintegrování JSON-LD je v tomto ohledu velice hladké. V čem se JSON-LD liší od klasického JSON? JSON-LD používá univerzální identifikující mechanismus v podobě identifikátoru IRI - *international resource identifier*. Dále formát JSON-LD představuje myšlenku kontextu - značí se klíčovým slovem `@context`.

Když spolu dva lidé komunikují, odehrává se konverzace ve sdíleném prostředí, typicky nazývaným "kontext konverzace". Tento sdílený kontext umožňuje každému používat zkrácené výrazy, jako křestní jméno společného kamaráda, aby komunikace probíhala rychleji, ale ne na úkor přesnosti. Kontext v JSON-LD funguje stejně. Umožňuje dvěma aplikacím používat zkrácené výrazy, aby byla komunikace efektivnější, ale stále jednoznačná.[SLK+23]

Dalšími novými klíčovými slovy jsou:

- `@id` - přiřazuje unikátní URL objektu
- `@type` - určuje datový typ (také definovaný jednoznačně dle IRI) daného objektu
- `@value` - přiřazuje objektu hodnotu - používáno většinou společně s `@type`

Obrázek 2.3: Jednoduchá JSON-LD datová struktura, zdroj: `https://json-ld.org`

Kapitola 3

Otevřená propojená data v ČR

V českém právním systému se tvoří prostředí pro sjednocení pravidel ohledně otevřených dat. Otevřenými daty se rozumí informace zveřejňované způsobem umožňujícím dálkový přístup v otevřeném a strojově čitelném formátu, jejichž způsob ani účel následného využití není povinným subjektem, který je zveřejňuje, omezen a které jsou evidovány v národním katalogu otevřených dat.[AC23] Níže představuji dva koncepty, jejichž znalost je pro pochopení následujících částí zásadní.

3.1 Sémantické slovníky pojmů

Sémantické slovníky pojmů definují pojmy ve specifickém kontextu. To zaopatřuje specificitu (víme, o jaký pojem v jakém kontextu se jedná) a univerzálnost (používání napříc kontexty). Takový pojem si představme jako slovo nebo sousloví s jednoznačnou definicí a jejím zdrojem. Zdrojem je například znění zákona, konkrétní paragraf v tomto zákoně, ale i obecná znalost, například význam pojmu tak, jak mu rozumí a používá ho určitá skupina expertů.[**?**] Dále v práci je často zaměňován *sémantický slovník pojmů* se slovem *slovník*.

3.2 Otevřené formální normy

Otevřené formální normy jsou technická doporučení zaměřená na vybrané datové sady, která zajišťují, že stejná data publikovaná různými poskytovateli budou interoperabilní. Tím je umožněno taková data jednodušeji využívat nezávisle na tom, od kterého jsou poskytovatele. Na základě OFN existují slovníky, které jsou jediným místem pravdy, na které se musí všichni konzumenti strukturovaných sémantických dat dle OFN obrátit. [v3c] Otevřené formální normy jsou udržovány projektem Evropské unie KODI[1].

[1] https://www.mvcr.cz/clanek/kodi.aspx

Kapitola 4

SForms

SForms[1] je knihovna napsaná ve frameworku React.[BLH+23] Slouží k zobrazení sémantických dat ve formě formulářů. Tato sémantická data jsou ve formátu JSON-LD. Pro práci s uspořádáním dat slouží atributy objektů JSON-LD, které mohou definovat pořadí otázek nebo jejich úroveň v hierarchii. SForms obsahuje i základní formulářová omezení, také definovaná atributy JSON-LD dat. Takové omezení je například "povinná položka". V práci se nejvíce věnuji použití SForms nad datovými sadami dle OFN. Knihovna SForms je udržováná skupinou ználostních a softwarových systémů na katedře počítačů FEL ČVUT[2].

4.1 Ukázka formuláře

Následujícím obrázkem chcí znázornit, jak může vypadat jeden objekt, se kterým SForms pracuje. Jedná se o objekt vytržený z celkového kontextu, ale zachytit komplexitu celého formuláře by bylo náročné a pro představení struktury dat, se kterými SForms pracuje, nepotřebné.

[1] https://github.com/kbss-cvut/s-forms
[2] https://kbss.felk.cvut.cz/web/

```
{
    "@id": "v-sgov-pojem:adresní-místo-c68cc36caedf854cf0705592f4008d43-q",
    "@type": ["doc:question", "form:trope-question"],
    "has-importance": "ft:important",
    "has-preceding-question": "v-sgov-pojem:zemepisná-šírka-c68cc36caedf854cf0705592f4008d43-q",
    "has-question-origin": "v-sgov-pojem:adresní-místo-c68cc36caedf854cf0705592f4008d43-q-qo",
    "has-label": {
        "@language": "cs",
        "@value": "URL adresního místa"
    },
    "has-main-processing-aspect-target": "v-sgov-pojem:adresní-místo",
    "has-parent-question-entity": "v-sgov-pojem:lokalizace-prostorového-objektu",
    "has-processing-aspect-target": "v-sgov-pojem:adresní-místo",
    "http://onto.fel.cvut.cz/ontologies/kodi/sgov2sforms/has-related-question-level": 4,
    "is-question-of-aspect": "v-sgov-pojem:adresní-místo",
    "is-question-of-intrinsic-mode": "v-sgov-pojem:adresní-místo",
    "label": {
        "@language": "cs",
        "@value": "URL adresního místa"
    }
}
```


Obrázek 4.1: Objekt popisující adresní místo a jeho vztahy s dalšími objekty ve struktuře JSON-LD

Kapitola 5

Pravidla návrhu uživatelských rozhraní (UI) a poskytnutí pohodlného uživatelského zážitku (UX)

V této kapitole představím pravidla návrhu UI (uživatelských rozhraní) se zaměřením na formuláře. Stejně tak rozeberu postupy, které jsou aplikovány obecně, aby se uživateli s daným uživatelským rozhraním efektivně a jednoduše pracovalo.

5.1 Pravidla UX pro návrh formulářů

V rámci analýzy formulářů OFN a návrhu zlepšujících řešení musím stále respektovat normy a postupy spojené s uživatelsky centrovaným návrhem. Jinak řečeno, má řešení musí stále odpovídat pravidlům, jak navrhovat uživatelsky přívětivé formuláře. Některá pravidla vytyčil v diplomové práci Bc. Tomáš Klíma, který se věnoval tvorbě knihovny SForms mimo jiné i po stránce uživatelského ovládání.[Kl1] Tato pravidla lze shrnout takto:

- položky formuláře na sebe logicky navazují
- položky formuláře jsou logicky seskupené
- formulář jde od jednoduchých položek k položkám komplexním
- formulář uživatele upozorní, když je vstup chybný
- formulář uživateli radí, jak by mohla odpověď vypadat
- formulář uživateli naznačí povinnou a nepovinnou položku

5.2 RAIL model

RAIL[1] model je struktura uvažování o návrhu a měření uživatelské použitelnosti aplikací definovaná skupinou na vývoj webů web.dev[2] společnosti

[1] https://web.dev/rail/
[2] https://web.dev

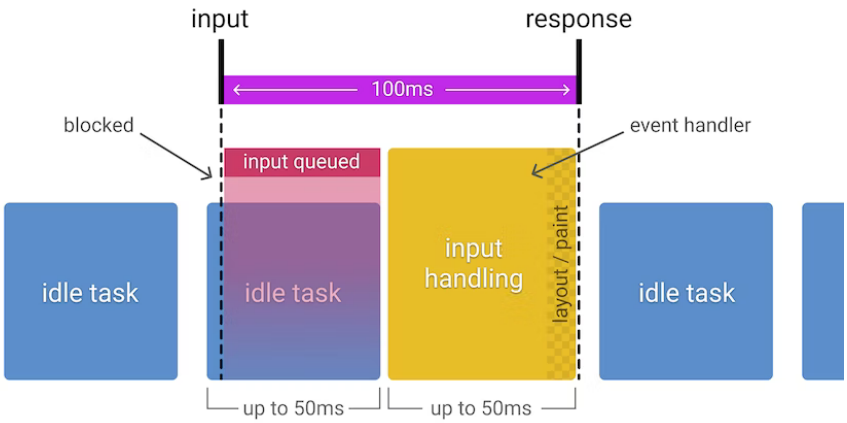
Google.[tea23] Z tohoto modelu vyplývají i metodiky, jak ideálního stavu dosáhnout. Ty jsou zahrnuty ve zkratce RAIL. Zkratka RAIL v sobě chová první písmena následujících anglických slov:

- **R** - response
- **A** - animation
- **I** - idle
- **L** - load

Model RAIL definuje ideální cíle uživatelského zážitku a metodiky jak těchto cílů dosáhnout. Bonusem jsou zřetelně vymezené metriky, pomocí kterých se dá uživatelská interakce hodnotit. Tyto metriky zmíním ke každému atributu RAIL modelu zvlášť v podsekcích níže.

5.2.1 R - response

Response, česky odpověď/odezva, se zaměřuje na procesování uživatelova vstupu. Dle RAIL stačí na poskytnutí odezvy uživatelovi 100 ms, ale doporučuje 50 ms. To je z toho důvodu, že počítá nejen s odezvou na danou uživatelovu interakci, tedy cílovou reakci aplikace, které chce uživatel dosáhnout, ale i s časem, který potřebuje aplikace na zpracování uživatelova vstupu jako takového (například kliknutí myší, přejetí prstem, posun stránky).



Obrázek 5.1: Schéma procesování vstupů v čase aplikací dle RAIL modelu, zdroj: https://web.dev/rail/

5.2.2 A - animation

V celé řadě dnešních aplikací (webových i nikoliv) se dnes z různých důvodů vyskytují animace. Nebudu brát v potaz animace z estetického hlediska. Estetické animace není pro tuto práci potřeba uvažovat. Zaměřím se na animace takzvaně funkční. To jsou animace, které pomáhají uživatelům k

orientaci v aplikaci. Navádí (např. na mapě), poskytují informace o stavu aplikace (načítací animace) nebo jinak řídí průběh interakce s aplikací. RAIL metodika udává, že maximální čas na vyprodukování jednoho snímku animace by měl být 10 ms. Tím se docílí plynulého vykreslení animace prohlížečem. Hlavním východiskem splnění této metriky je vizuální plynulost aplikace.

5.2.3 I - idle

V tomto konxextu slovo idle lze volně přeložit jako "nečinný". Ideou této částí modelu RAIL je využívání chvilkové nečinnosti aplikace k přípravě následující práce. Prioritou stále zůstává 50ms odezva na uživatelovu interakci, ale pokud jsou k dispozici nějaké funkce, které aplikace dokáže udělat, aniž by zasáhla do času odezvy, tak se jedná o efektivní využití zdrojů. Propojení "nečinnosti"s odezvou je znázorněno na obrázku 5.1.

5.2.4 L - load

Posledním pilířem modelu RAIL je načítání. Nejdůležitější načítání je z pohledu RAIL to prvotní. Doba trvání tohoto prvotního načtení se anglicky nazýva TTI - time to interactive. Pokrývá to čas od navštívení webové stránky až po moment, kdy je uživatel schopen se stránkou interagovat a stránka je tuto interakci schopna zprocesovat. Aktuálně RAIL udává, že by TTI aplikací neměl přesáhnout 5 sekund pro středně výkonné mobilní telefony s připojením 3G. Pak už začíná být uživatel frustrován. Narozdíl od limitu odezvy, která se díky stálým kognitivním schopnostem lidí měnit již příliš nebude, se schovívavost vůči TTI může v budoucnu měnit.

Část II

Analýza formulářů OFN v SForms a identifikace uživatelsky nevhodných částí

Kapitola 6

Aktuální stav SForms

V této kapitole analyzuji, jak SForms aktuálně funguje, jak se pracuje s formuláři v SForms a identifikuji místa ke zlepšení uživatelské použitelnosti v kontextu otevřených formálních norem.

Pro identifikaci aktuálního stavu SForms a formulářů vykreslovaných touto knihovnou vycházím specificky z průchodů formulářem pro *Turistický cíl*. Jako referenční implementaci používám ukázkový formulář[1] vykreslený knihovnou SForms.

6.1 Formulář - Turistický cíl

Otevřená formální norma pro objekt *Turistický cíl*[2] je již finální, proto se nabízí jako ideální kandidát k analýze. Také obsahuje atributy, na které se, dle zadání bakalářské práce, mám soustředit - zeměpisné souřadnice a adresu. Celková struktura OFN pro *Turistický cíl* je naznačena na obrázku 6.1. Zde mě bude především zajímat vazba na *Umístění*, které má v sobě zapouzdřenou geometrii (zeměpisné souřadnice) a adresu.

Adresa je také definovaná dle OFN[3]. Hlavním prvkem adresy je adresní místo. Adresním místem se rozumí takové místo v terénu, kterému lze ve vztahu ke stavebnímu objektu jednoznačně přiřadit adresu.[v3a] Dále v práci bude často zaměňováno adresní místo s adresou.

6.2 Možné změny v SForms

Při analýze formuláře *Turistický cíl* jsem objevil několik možností, jak zefektivnit a zkvalitnit sběr dat. Vycházím z předchozí evaluace, kterou mi poskytl vedoucí práce. Evaluační průchody jsem si sám vyzkoušel, abych odhalil neintuitivní a neefektivní části formuláře. Možnosti, jak formulář obecně udělat intuitivnějším je několik:

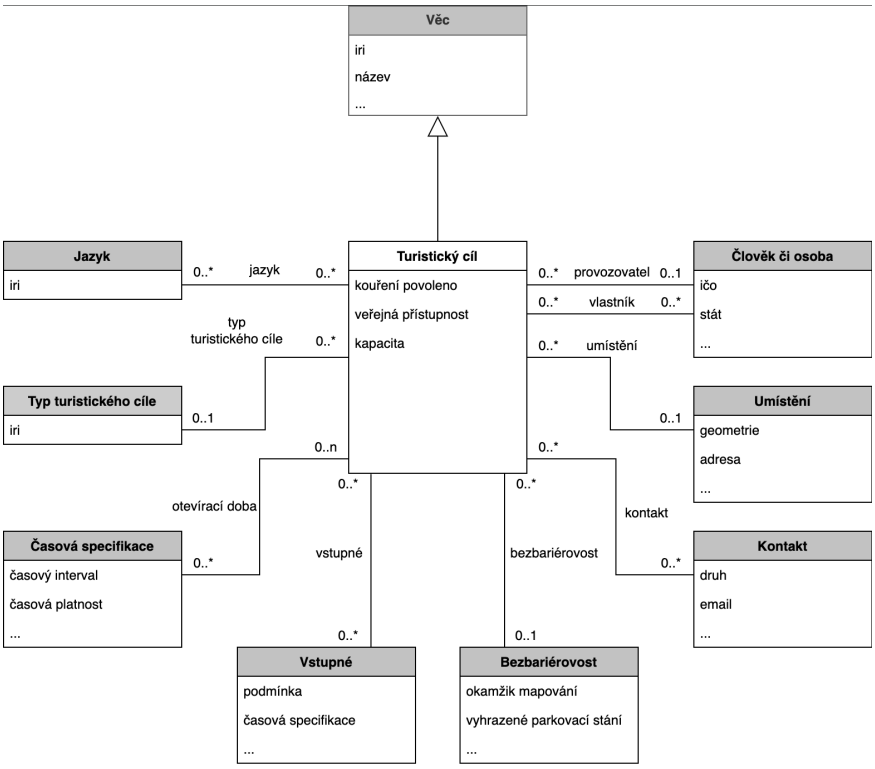
- změna pořadí sekcí ve formuláři

[1] https://s-forms-kbss.netlify.app/?path=/story/sforms–tourist-destination-form-2
[2] https://ofn.gov.cz/turistické-cíle/2020-07-01/
[3] https://ofn.gov.cz/adresy/2020-07-01/

- skrývání sekcí formuláře
- přidání mapy
- deaktivace formulářových polí (zákaz manuální modifikace)

Zároveň ale musím pracovat při analýze se schopnostmi uživatelů. Po uživatelích lze obecně očekávat, že se v dané agendě budou orientovat. Je ale zapotřebí, aby pro ně vyplňování bylo co nejrychlejší a nejpohodlnější. V rámci mé práce definuji jedno hlavní kritérium pro finální prioritizaci požadavků:

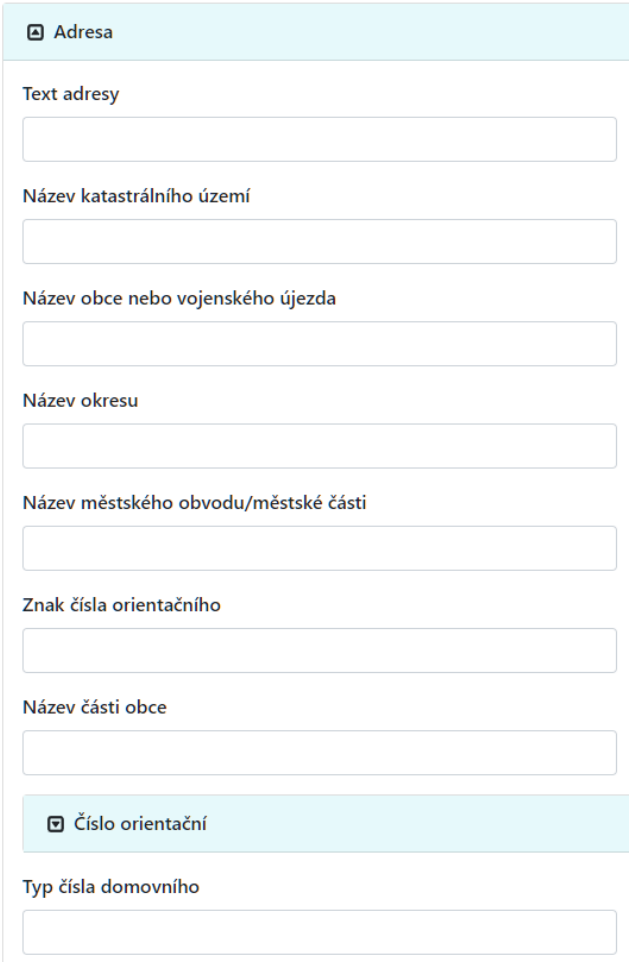
Rychlost vyplnění formuláře - bude preferován návrh, který umožní uživatelům rychleji zadávat data



Obrázek 6.1: Diagram datového modelu turistického cíle, zdroj: https://ofn.gov.cz/turistické-cíle/2020-07-01/obrázky/turistický-cíl.svg

6.3 Zadání adresy

Zadávání adresy aktuálně probíhá vyplněním množiny formulářových polích rozdělených do několika kategorií. Struktura adresy je přejata ze zákonných vyhlášek. Je definována ve slovníku Slovník zákona č. 111/2009 Sb., o základních registrech.[v3b] Místy nevhodná souslednost polí a komplexita reprezentace adresy bohužel zadávání informací do formuláře ztěžuje. Zároveň se zadává kód adresního místa, který je těžko dohledatelný v registrech a má pro uživatele složitý formát (číslo s různým počtem číslic).



Obrázek 6.2: Ukázka složení adresy dle OFN v SForms, zdroj: https://s-forms-kbss.netlify.app

Při zadávání adresy neprobíhají kontroly vstupů. To narušuje kvalitu sbíraných dat i použitelnost pro uživatele.

6.3.1 Vytyčení pojmů spojených s adresou

Pro definování pojmu neúplné adresy, který se používá dále v práci, se mohu odkázat na definici adresy dle českého statistického úřadu. Z této definice je modelována adresa i ve slovnících propojených dat:

Adresou se rozumí kombinace názvu okresu, názvu obce nebo vojenského újezdu, názvu části obce nebo v případě hlavního města Prahy názvu katastrálního území a názvu městského obvodu, čísla popisného nebo evidenčního, názvu ulice a čísla orientačního a dále zvláštních údajů pro doručování prostřednictvím poštovních služeb, která jednoznačně určuje adresní místo.[v3b] [7]

Z výše uvedeného vyplývají následující položky adresy:

- název ulice

- číslo popisné / číslo evidenční
- číslo orientační
- název katastrálního území Prahy + název městského obvodu / název části obce
- poštovní směrovací číslo (PSČ)
- název obce / název vojenského újezdu
- název okresu

Aby byla komponenta uživateli schopna **jednoznačně** určit adresu, musí uživatel zadat alespoň tyto položky:

1. Možnosti **minimální úplné adresy**
   a. název ulice + číslo popisné / číslo orientační + název obce
   b. název ulice + číslo popisné / číslo orientační + PSČ

   Pokud chybí přesně jedna z trojice položek v případě 1.a. i 1.b., tak se jedná o neúplnou adresu. Neúplná adresa je tedy následujícího typu:

2. Možnosti **neúplné adresy**
   a. číslo popisné + číslo orientační + název obce / PSČ / název části obce (+ další položky bez názvu ulice)
   b. název ulice + číslo popisné / číslo orientační (+ další položky bez názvu ulice)
   c. název ulice + název obce / PSČ (+ další položky bez čísla popisného a čísla orientačního)

Pokud chybí více než dva z trojice 1.a. nebo 1.b., tak adresu nelze určit a musí se uživatel snažit zadat adresu jiným způsobem, například výběrem bodu z mapy.

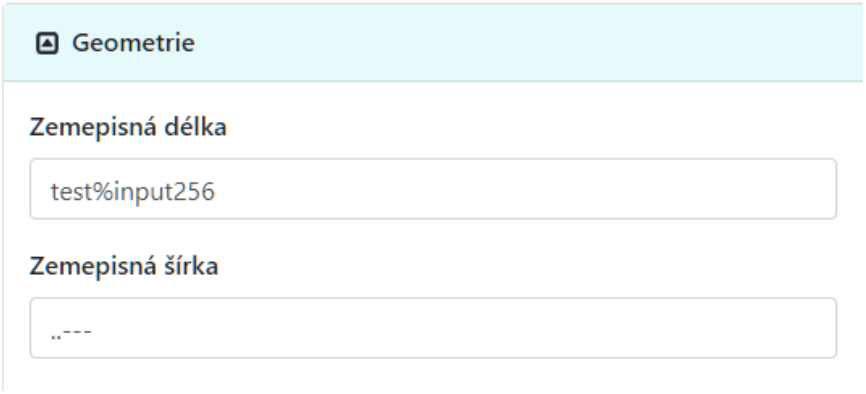
6.4 Zadání zeměpisných souřadnic

Zeměpisné souřadnice se vyplňují přímočaře formou formulářových polí - zeměpisná délka a zeměpisná šířku. Pro uživatele je vyplňování polí běžnou aktivitou při používání nejen webových aplikací. Při vyplňování souřadnic vyvstává ve formuláři několik problémů. Formulářová pole nejsou ošetřena před chybnými vstupy čísel a v aktuálním stavu mohou pole přijmout i textový vstup. S tím souvisí, že uživatel na to není upozorněn, a to může být matoucí.

Stejně tak uživatel není vizuálně ujištěn o správnosti zadaných souřadnic - chybí např. zobrazení polohy na mapě. To může být zásadní, kdyby se ve vstupu vyskytl překlep, případně kdyby uživatel špatně zkopíroval souřadnice.

Nedostatky zadávání souřadnic jdou shrnout takto:

- chybí kontrola vstupu - formulář dovolí psát do pole všechny znaky
- chybí upozornění na špatný vstup
- chybí vizuální kontrola, jestli souřadnice souhlasí s adresou
- není ošetřeno zadávání hodnot jiných souřadnicových systémů



Obrázek 6.3: Ukázka absence kontroly vstupu souřadnic v SForms, https://s-forms-kbss.netlify.app

6.4.1 Vytyčení pojmů spojených s polohou

V práci definuji polohu jako bod určený buď zeměpisnými souřadnicemi nebo umístěním na mapě (vizuální reprezentace bodu). Dále polohu dělím na dva druhy:

- **přesná poloha** = přesné zeměpisné souřadnice (definují jednoznačně žádoucí bod), nebo přesné umístění na mapě (definuje jednoznačně žádoucí bod)
- **přibližná poloha** = přibližné zeměpisné souřadnice (souřadnice bodu, který leží v okruhu 500 metrů od žádoucího bodu), nebo přibližné umístění (definuje bod, který leží v okruhu 500 metrů od žádoucího bodu)

6.5 Validace vstupů

I bez přihlédnutí na špatně nastavené typy inputových polí pro formulář *Turistický cíl* je výchozí validace vstupů v SForms ještě v začátcích. Aplikace SForms používá při renderování polí knihovnu stylizovaných komponent React Bootstrap, která zaobaluje HTML `<input>` elementy do reactových komponent. Tím se i přenáší zabudovaná validace v prohlížečích. Tento typ validace je jednoduchý a rychlejší než vlastní typ validace.[Cor23a]

V prohlížečích validace inputových elementů probíhá ve výchozím stavu až při události `onSubmit`. To je událost, která nastane, když se v HTML

elementu `<form>` stiskne tlačítko. V SForms tato událost nastává při uložení formuláře. Jenže toto chování nemusí být žádoucí pro všechny situace. Více v sekci 7.5 Validace vstupů u existujících řešení.

SForms tedy umí využit všechnu zabudovanou validaci prohlížečů, která obnáší:

- povinná pole
- typy polí - email, číslo, heslo
- minimální a maximální délka vstupů (počet znaků)
- minimální a maximální hodnota čísla
- vzor - regulární výraz, který musí vstup splňovat

Nicméně tato pravidla často nejsou definovaná v datech a SForms tedy na všechny vstupy nahlíží jako textová pole bez omezení.

Jedinou vlastní nadstavbou SForms, co se validace týče, je kontrola povinných polí. To jsou pole, která se vyrenderovala pro objekt, který má ve své struktuře dvojici

```
"requires-answer" : true
```

Vztah s názvem `requires-answer` není ve formátu celkového IRI, protože je v JSON-LD struktuře definovaný kontext (viz sekce 2.5 JSON-LD):

```
"requires-answer": {
  "@id": "http://onto.fel.cvut.cz/ontologies/form/requires-
    answer",
  "@type": "http://www.w3.org/2001/XMLSchema#boolean"
}
```

Listing 6.1: Ukázka nastavení kontextu

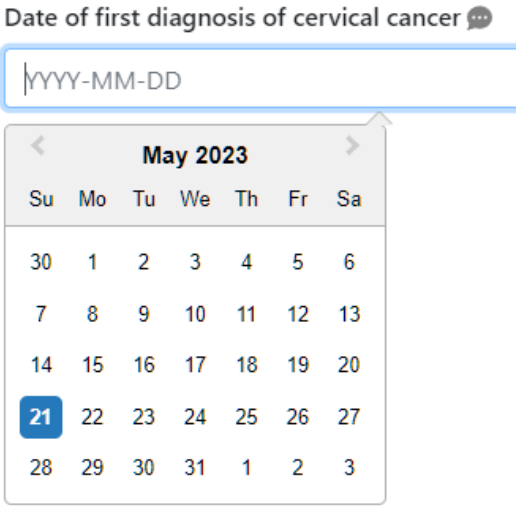
Pole s hodnotou **true** pro atribut `requires-answer` se bude při nevyplnění při `onSubmit` události tvářit jako chybné a chybová hláška bude také vlastní (ne ta zabudovaná v prohlížeči). To je dáno implementací React-Bootstrap[4] vlastní validace. Chyba a její hláška (pokud je odpověď chybná) bude znázorněna v JSON-LD objektu dvojicemi (celé IRI nastaveno v kontextu podobně jako u `requires-answer`)

```
"has-valid-answer": false
"has-validation-message": "Poli Text adresy chybí hodnota"
```

6.6 Výběr data a času

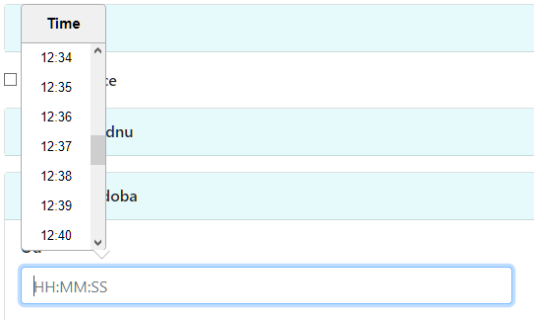
Datum se v SForms zadává do inputu s maskou YYYY-MM-DD (například hodnota 2000-02-15). Po kliknutí do inputového pole se také zobrazí doplňující grafika kalendáře, kterou mohou uživatelé znát z mobilních aplikací.

[4] https://react-bootstrap.github.io



Obrázek 6.4: Ukázka zadávání data, https://s-forms-kbss.netlify.app

Čas se zadává do inputu s maskou HH:MM:SS (například 19:30:00). Po kliknutí do inputového pole se objeví seznam, kde lze vybrat hodinu a minutu. To se může pro nějaké scénáře hodit, ale pro drtivou většinu případů je minutová granularita výběru až moc velká pro rychlé zadání času. Stejně tak se v reálném světe otevírací doby turistických cílů pohybují nejčastěji v násobcích čtvrt hodin (např. 8:00-18:30, skoro nikdy např. 8:03-17:23).



Obrázek 6.5: Ukázka zadávání času, https://s-forms-kbss.netlify.app

6.7 Shrnutí nalezených problémů

Z uživatelského otestování formuláře pro *Turistický cíl* vyplynulo, že největší problém dělá zadávání zeměpisných souřadnic a adresních míst. Zadávání data a času může být sice zefektivněno, ale problém s používáním formuláře nebyl tak razantní jako pro souřadnice a adresní místa. Po konzultaci s vedoucím práce jsme došli k závěru, že komponenty pro výběr data a času jsou aktuálně dostatečné. Seřazené problémy dle komplexity při práci s formuláři vypadají takto:

1. zadávaní adres / adresních míst
2. zadávání zeměpisných souřadnic
3. zadávání času
4. zadávání data

V analýze existujících řešení se budu zaměřovat hlavně na řešení, které se týkají adres a souřadnic - to budou především různé mapové komponenty. Na ty jsou uživatelé z používání webů zvyklí a použití mapy by mohlo vést částečně k vyřešení nalezených problémů v této kapitole.

Kapitola 7

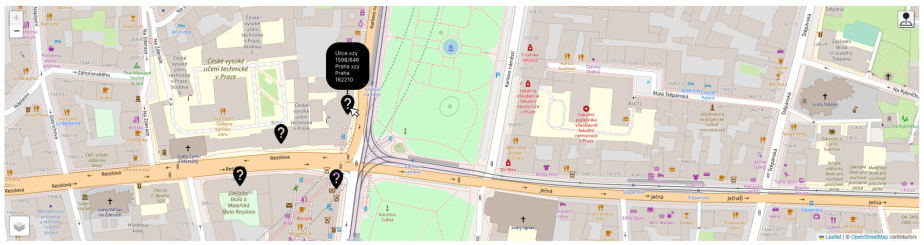
Analýza existujících řešení

V kapitole Analýza existujících řešení analyzuji postupy, které se v současné době aplikují při návrhu moderních webů v kontextu map, adres, souřadnic a formulářů. Také se již zaměřuji na webové technologie, které by šlo použit při implementaci výsledných komponent.

7.1 Obecná práce s mapou

V dnešní době už se interakce uživatele s mapou jaksi standardizovala. Běžný uživatel je zvyklý na interakce posunu, přiblížení a oddálení. Také je zvykem, že čím je mapa více přiblížená, tím více podrobných informací uživateli poskytne. To souvisí hlavně s adresními místy a "body pozornosti"(anglicky point of interest). Podrobnější informace mapa poskytuje i po kliknutí do mapy. Tyto druhy získání podrobností z mapy analyzuji v seznamu níže. Ten představuje oba přístupy.

1. Zobrazení orientováno úrovní přiblížení mapy
   Mapa zobrazí adresní místa a body pozornosti až při určitém stupni přiblížení mapy. To totiž znamená, že uživatel již tuší, kde své cílové adresní místo / bod pozornosti hledat a je pro něj relevantní vidět blízká adresní místa / body pozornosti. Tento princip funguje u všech známých online map - Google maps[1], Mapy.cz[2]. Po přiblížení se zobrazuje větší množství ikon reprezentující různé podniky, památky a podobně.



Obrázek 7.1: Prvotní návrh zobrazení podrobností při dostatečném přiblížení a najetí kurzorem na ikonu adresního místa, vytvořeno ve Figma

[1] https://www.google.com/maps
[2] https://mapy.cz/

2. Zobrazení míst po kliknutí
   Mapa zobrazí zakliknuté místo a jeho podrobnosti. Pokud je zaklikнuté místo bez ikonky adresního místa nebo jiného bodu pozornosti, tak se pouze označí vybrané místo a zobrazí se zeměpisné souřadnice odpovídající danému místu.

Kdybych měl vztáhnout toto obecné ovládání na kontext adresních míst, tak je uživatelsky přívětivé, aby uživatel adresní místo vybíral sám - tzn. aby mapa sloužila pouze jako prostředník, který uživateli ukáže dostupná adresní místa a ne, aby rovnou adresní místo volila. Takže nabídne uživatelům možnost zobrazit bližší informace o adresním místě a vybrat adresní místo, což doplní informace o adresním místě do formuláře. Stejně tak by se mapa neměla přehlcovat pro uživatele zbytečnými informacemi a body pozornosti. Oproti klasickým mapovým vyhledávačům (Google maps, Mapy.cz) bude potřeba adresní místa více zvýraznit a nefiltrovat, protože u těchto vyhledávačů se zobrazují pouze ta místa, která mají pro běžného uživatele webu nějaký význam - restaurace, obchod a podobné.

Ve dvou následujících podsekcích analyzuji existující React mapové komponenty. Kritéria pro porovnání řešení jsou:

- podpora JavaScriptového frameworku React
- dokumentace mapové knihovny
- open-source
- jednoduchost implementace

7.1.1 Google maps react

Jedním z kandidátu pro implementaci mapy v rámci rozšiřujících webových komponent je knihovna google-maps-react[3]. Ta poskytuje React komponenty nad mapami google. Výhodou tohoto řešení je jednoduchá aplikovatelnost, jelikož google mapy jsou velice známé a používané. Dále existuje na internetu velké množství tutoriálů a příkladů, například hned v popisu knihovny na GitHubu. Nevýhodou je potřeba založení projektu v Google cloud ekosystému a vygenerování Google maps API klíče k používání reactové knihovny.

Výhody	Nevýhody
známé a používané mapy	nutná potřeba API klíče
široká dokumentace a příklady	

Tabulka 7.1: Shrnutí výhod a nevýhod použití knihovny google maps react

[3] https://github.com/fullstackreact/google-maps-react

7.1.2 React Leaflet

Dalším kandidátem je knihovna React Leaflet[4], což je knihovna reactových komponent nad JavaScriptovou knihovnou Leaflet[5].[Cc22] [Aga23] Knihovna Leaflet i React Leaflet je velice rozšířená mezi vývojáři pro její jednoduchost, dokumentaci a použitelnost. Pro práci s mapovými dlaždicemi nepotřebuje vývojář pro svůj projekt žádný API klíč.

Výhody	Nevýhody
široká dokumentace a příklady	není masově používáno
není potřeba externího API	
založeno na známé JS knihovně Leaflet	

Tabulka 7.2: Shrnutí výhod a nevýhod použití knihovny React Leaflet

7.2 Zadání zeměpisných souřadnic

V této sekci se věnuji čistě zadávání zeměpisných souřadnic izolovaně, tedy bez vlivu jiné interakce s formulářem.

V mapových vyhledávačích Google maps a Mapy.cz lze hledat dle souřadnic po napsání souřadnic za sebou přímo do vyhledávacího pole. To slouží k rychlému vyhledání. I přesto, že hlavním kritériem pro funkcionalitu vyvíjené komponenty má být rychlost vyplnění dat, tak v tomto případě převáží přehlednost, která nakonec povede k rychlejšímu a přesnějšímu vyplnění. To bude dáno tím, že uživatel bude vyplňovat dvě různá formulářová pole - zvlášť pro zeměpisnou délku a zeměpisnou šířku. Zeměpisné souřadnice lze vybrat i kliknutím do mapy.

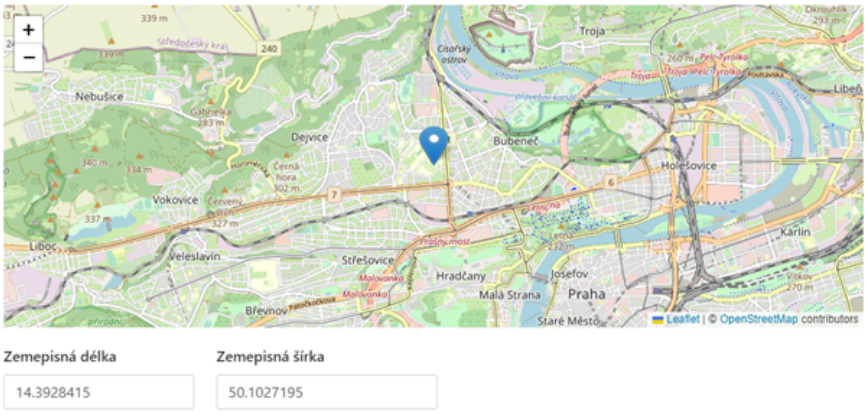
Pro přehledné zadání souřadnic musí být uživatel schopen vyhledávat v mapě, která bude sloužit ke kontrole zadaných souřadnic nebo k výběru souřadnic přímo z mapy. Oproti oběma zmíněným mapovým vyhledávačům se nabízí jedna modifikace. Aby uživatel měl zpětnou vazbu ihned po vstupu nebo změně souřadnic, je žádoucí, aby mapa zobrazovala zadané souřadnice hned, a ne až po kliknutí na tlačítko "vyhledat"(někdy také ikonka lupy).

7.3 Zadání adresy / adresního místa

V případě zadávání adresy se zaměřuji na rychlost vyplnění dat, ale stále beru ohled na pravidla návrhu formulářů. To znamená, že chceme od uživatele získat data rychle, ale ne na úkor kvality dat a uživatelské přívětivosti formuláře.

Existujících variant na zadávání adresy je vcelku mnoho. V následujících podsekcích jmenuji ty, které jsou nejrychlejší, nejrozšířenější a nejjednodušší.

[4] https://react-leaflet.js.org
[5] https://leafletjs.com

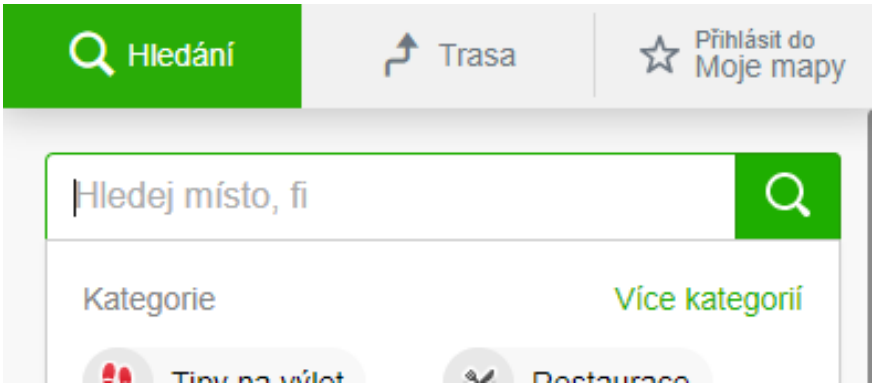


Obrázek 7.2: Výběr po kliknutí

7.3.1 Jedno inputové pole

Uživatel má k dispozici pouze jedno formulářové pole, kam napíše text adresy. S touto variantou se váže i možnost fulltextové vyhledávání, kdy se na základě uživatelova vstupu hledají odpovídající záznamy z databáze adresních míst. Tyto záznamy se uživateli "našeptávají" a poskytují jednodušší výběr. Tato varianta je používána i v aplikacích jako Google Maps a Mapy.cz. Výhodou této varianty je rychlost a jednoduchost zápisu adresy. Nevýhodou je potřeba implementace "našeptávání", aby měl uživatel návrhy adres i v případě, kdy nezná přesnou adresu.

Obrázek 7.3: Inputové pole google maps, zdroj: `https://www.google.com/maps`



Obrázek 7.4: Inputové pole mapy.cz, zdroj: `https://www.mapy.cz`

7.3.2 Zadání adresy výběrem bodu z mapy

V momentě, kdy uživatel nezná adresu, ale zná přesnou polohu (pozice na mapě, nebo zeměpisné souřadnice), kterou chce zadat ve formě adresního místa, nezbývá nic jiného, než zadat adresní místo výběrem adresního místa z mapy. Obecně je tato možnost jediným řešením, kdy uživatel zná polohu, ale nezná adresu, a je také nejrychlejším zadáním dat v tomto případě. Tento typ interakce s mapou mohou uživatelé znát z Google Maps. Podobný princip je v jistém slova smyslu i u výběru pobočky, kam vám eshopy doručí zboží.

7.3.3 Vyplnění více formulářových polí odpovídajících částem adresy

Stále i v současnosti se vyskytují na mnoha webových stránkách formuláře, kdy uživatel vyplňuje adresu dle jejích prvků (viz podsekce 6.3.1 Pole adresy). Setkat se s tímto typem je možné především na webech veřejné správy nebo státních institucí. Toto vyplňování je velice přesné a uživatelům je jasné, jakou část adresy zrovna zadávají.

Za předpokladu, kdy uživatel zná všechny prvky adresy, tak je tento způsob zadávání zbytečně zdlouhavý, protože uživatel musí překliknout do každého pole zvlášť a vyplnit požadovanou informaci. Mnohdy je lepší použít podobnou alternativu - viz podsekce 7.3.5.

7.3.4 Zadání kódu adresního místa

Zadání kódu adresního místa je nejrychlejší a nejjistější způsob zadávání adresy. Jedná se o číselnou posloupnost. Podle hlavního kritéria analýzy je rychlost na prvním místě. Proto by se nabízelo, aby tato možnost byla implementována. Navíc, s přihlédnutím k typu vstupu - číslo, je pravděpodobné, že uživatel bude dané číslo do formulářového pole kopírovat. Toto kopírování urychlí celý proces ještě více. Nabízí se propojit první a čtvrtou možnost a implementovat rozpoznání, kdy uživatel zadává kód adresního místa, a kdy se snaží napsat adresu. To by nepřehlcovalo formulář a dovolilo uživateli psát jen do jednoho pole a přijmout různé vstupy. Ovšem někoho by toto řešení mohlo zmást. V běžné praxi se člověk s adresními místy a jejich kódy nesetká, nicméně někteří úředníci by znalost adresních míst mít mohli a tento způsob zadání by jim práci značně ulehčil.

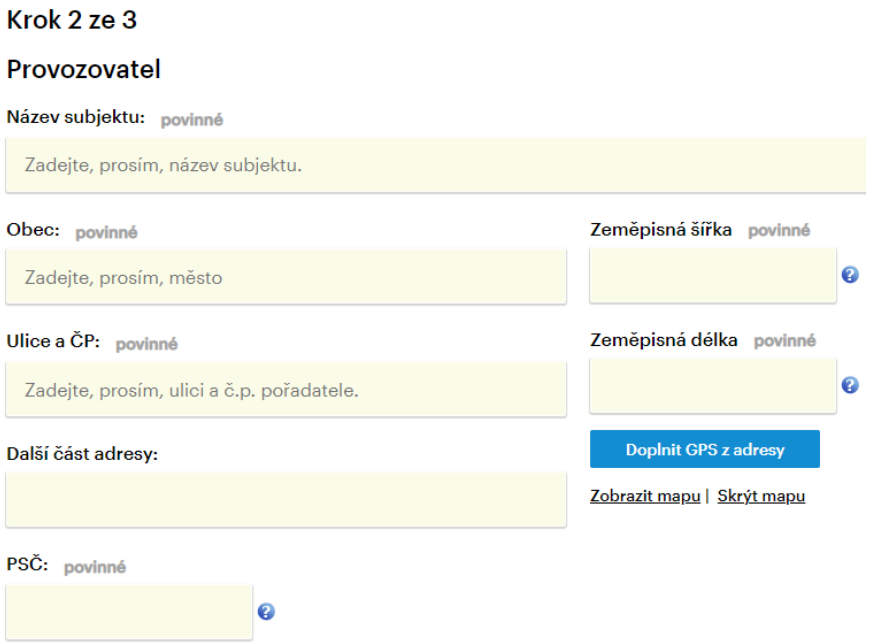
7.3.5 Vyplnění více formulářových polí odpovídající zadání minimální úplné adresy

Typ vyplňování osobních údajů a fakturačních adres na e shopech je dnes velice intuitivní a požaduje po zákaznících jen minimální úsilí. Za dobu surfování na internetu se uživatelé naučili tento již standardní způsob vyplňování adresy. Dá se říct, že už ji vyplňují automaticky, pokud se jedná o jim známé adresy. Narozdíl od možnosti představené v podsekci 7.3.3 je zadání jednodušší, protože stačí vyplnit základní pole (chybí například část obce) a související

informace jsou v jednom formulářovém vstupu - například název ulice + číslo popisné.

Na obrázku 7.5 je výstřižek ze zadávání adresy na eshopu Datart.cz. Podobně je to na dalším obrázku 7.6 z portálu kudyznudy.cz, kde se ale při zadání adresy očekávají i zeměpisné souřadnice.

Obrázek 7.5: Zadání v eshopu s elektronikou Datart.cz, zdroj: `https://www.datart.cz`



Obrázek 7.6: Zadání adresy nové aktivity na portále kudyznudy.cz, zdroj: `https://www.kudyznudy.cz/aktivity/nova-aktivita`

7.4 Vyhledávání adresních míst

Protože SForms nijak nepodporuje vyhledávání adresních míst, musím se zaměřit také na to, jak budu moci poskytnout uživatelům data o adresních místech ve formuláři tak, aby nemuseli data hledat v externích webových aplikacích. Zaprvé tím urychlím celkový proces zadávání a také udržím v rámci SForms integritu toho, že informace o adresních místech budou pocházet z jednoho zdroje pravdy.

Adresní místa jsou k nalezení mnoha způsoby. Centrální registr RÚIAN[6] nabízí služby, kterými obstarává přístup k datům v registru.[V20] Tyto služby se liší v aktuálnosti dat, uživatelském rozhraní a i ve struktuře získaných dat. V podsekcích dále se věnuji analýze těchto služeb, abych byl schopen rozhodnout, jak lidé vyhledávají adresní místa nyní, a jakým způsobem budu moci implementovat ve své práci získání těchto dat a jejich automatické vyplnění do formuláře.

7.4.1 Veřejný dálkový přístup - VDP

Veřejný dálkový přístup je název webové aplikace Českého úřadu zeměměřického a katastrálního (ČÚZK), která slouží k vyhledávání různých prvků v registru RÚIAN. Systém je z roku 2013 a jeho tvorba byla financována částečně evropskou dotací. Úvodní stránkou je `https://vdp.cuzk.cz/`.

[6] https://www.cuzk.cz/ruian/

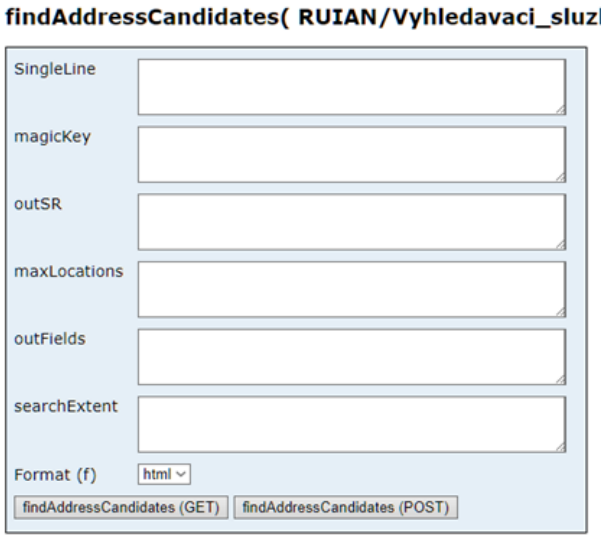
Výhodou této aplikace je, že data, nad kterými vyhledává, jsou skoro aktuální (maximálně několik hodin stará). Nevýhodou je uživatelská nepřívětivost. Ta odradí především laiky. Pro úředníky, nebo jiné pravidelné uživatele, poskytuje aplikace dobrý zdroj informací, ale i tak by se dalo vyhledávání zlepšit. Příkladem může být vyhledávání adresního místa. Uživatel musí data zadávat postupně, a vždy potvrdit zadání daného pole. To vyplnění prodlužuje. Zároveň chybí našeptávání.

7.4.2 Vyhledávací (geokódovací) služba nad daty RÚIAN

Dalším zdrojem dat z RÚIAN je geokódovací služba s názvem GeocodeSOE. Je to webové REST rozhraní, které lze konzumovat přímo z webových stránek. Nabízí endpointy, které vrací data ve formátu JSON, pro funkcionality:[AP19]

- suggest(text, location, distance, maxSuggestions) - "našeptání"
- find(text, bbox, location, distance, outSR, outFields, maxLocations, magicKey) - "hledání dle textu a polohy"
- findAddressCandidates(SingleLine, magicKey, outSR, maxLocations, outFields, searchExtent) - "hledání dle textu"

Data, nad kterými REST aplikační rozhraní pracuje, jsou vždy aktuální. Nevýhodou ovšem je absence kódu adresního místa ve vrácené datové struktuře. Také je tento způsob vyhledávání nepoužitelný pro laiky.



Obrázek 7.7: Uživatelské rozhraní GeocodeSOE, zdroj: web GeocodeSOE

7.4.3 Geoprohlížeč nad daty RÚIAN

Existuje grafická aplikace nad daty RÚIAN. Jedná se o Geoprohlížeč[7]. Je to oficiální webová grafická aplikace ČÚZK, kdy se uživatelům zobrazí interaktivní mapa ČR. Mapa je nezvyklá pro širokou veřejnost vizuálně, nabízí mnoho různých nastavení a poskytuje několik zobrazovacích vrstev, mezi kterými lze vybírat, kombinovat. Pro člověka, který s mapou nepracuje častěji je ovládání těžké. Výběr vrstev a filtrování informací, které chceme zobrazit/skrýt je neintuitivní. Běžný uživatel je zvyklý na změnu vzhledu mapy z aplikací jako Google maps a Mapy.cz, ale zde se musí několikrát kliknout a ještě trávit čas na výběru odpovídající vrstvy.

Pro někoho, kdo s mapou umí efektivně zacházet se jedná o velmi efektivní nástroj. Hlavním plusem je přesně onen počet vrstev - uživatel má možnost dostat se k velkému množství aktuálních informací z různých registrů a služeb. Mapa také podporuje fulltextové vyhledávání s našeptáváním.

[7] https://ags.cuzk.cz/geoprohlizec

Obrázek 7.8: Uživatelské rozhraní geoprohlížeče, zdroj: `https://ags.cuzk.cz/geoprohlizec`

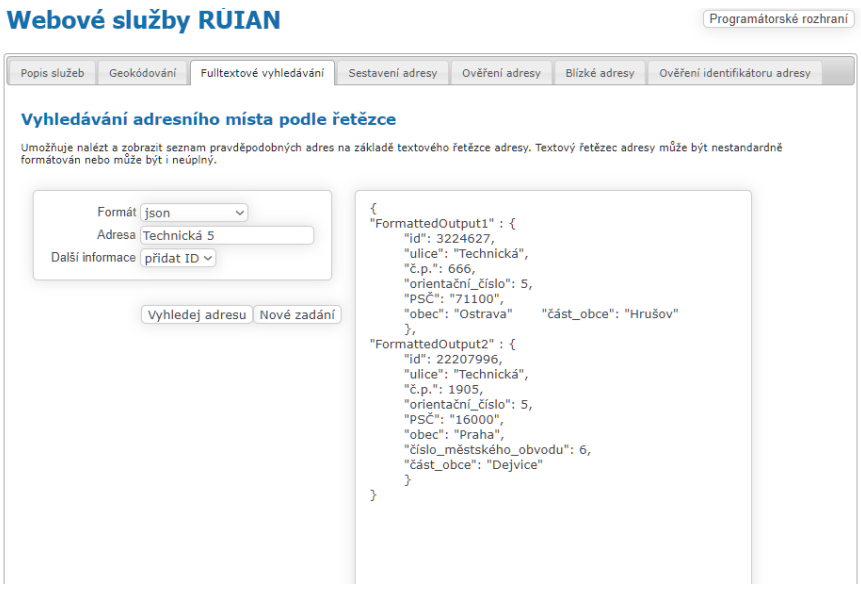
7.4.4 Webové služby RÚIAN

Webové služby RÚIAN je REST služba s webovým grafickým rozhraním. Služba poskytuje mnoho druhů vyhledávání - viz obrázek 7.9. U většiny funkcionalit je přítomno našeptávání. Stejně tak je umožněn uživatelům formát zadávání adresy, například výběr adresa / prvky adresy / identifikátor RÚIAN. Bohužel na stránkách Výzkumného ústavu geodetického, topografického a kartografického je implementováno a k dispozici online pouze ukázkové řešení, které se již neaktualizuje, a data, nad kterými služba pracuje, jsou z roku 2017. Stejně tak se zde opakuje problém s kódy adresních míst. Různé typy vyhledávání poskytují na výběr, zda k výsledku vyhledávání přidat ID (tedy kód adresního místa), nebo polohu. Obě tyto informace, které využíváme ve formulářích, ale dostat z jednoho dotazu nelze.

7.4.5 INSPIRE téma Adresy (AD)

INSPIRE (Infrastructure for Spatial Information in the European community) téma Adresy je projekt financovaným grantem EU. Je postavený nad daty RÚIAN pro různé kategorie prostorových objektů. Pro adresní místa je relevantní datová sada adres. Tato služba vystavuje webové rozhraní pro prohlížení a stahování dat. K tomu slouží služby WMS (web map service - prohlížecí služba) a WFS (web feature service - stahovací služba). Tento způsob procházení/stahování dat je uživatelsky náročný a není vhodný pro širokou veřejnost. Má tři identifikované výhody - data jsou aktualizovaná v řádu hodin; v odpovědi ve formátu GML (Geography Markup Language) jsou všechna potřebná data pro vyplnění informací o adresním místě; webová služba je zdokumentována[8].[M21] Nevýhodou pak je, že služba neposkytuje

[8] https://services.cuzk.cz/doc/inspire-ad-download.pdf



Obrázek 7.9: Uživatelské rozhraní webových služeb RÚIAN, zdroj: `http://www.vugtk.cz/euradin/ruian/rest.py`

efektivní formu našeptávání. Při dotazu našeptávání pro řetězec "hora"vrátí služba data o objemu vyšších desítek MB dat.

Rozhraní nabízí několik druhů dotazů, které by šly využít dle specifické potřeby:

- GetFeatureByPoint
- GetFeatureById
- GetAddressFW
- GetAddressByComponents

7.4.6 Aplikace usnadňující vyhledávání v systému RÚIAN - RUIAN-search

Pro vyhledávání v datech RÚIAN je použitelná i služba vyvinutá v rámci bakalářské práce studentem FIT ČVUT Tomášem Le. [Le20] Aktuálně tato služba není spuštěná, ale po konzultaci s doktorem Blaškem jsme došli k závěru, že katedra počítačů by případně mohla zařídit potřebné hardwarové a softwarové prostředí pro spuštění a správu této open source služby. Dále v textu budu tuto aplikaci také nazývat RUIAN-search. Kód je k dispozici v GitHub repozitáři na adrese https://github.com/letomas/RUIAN-search.

Služba má grafické webové rozhraní, které komunikuje přes REST API webového serveru, na kterém jsou uložena všechna data ze systému RÚIAN. Aplikace umožňuje nalézt adresní místa různými způsoby - zadáním adresy (obec, část obce, ulice, číslo domovní/orientační), nebo zadáním kódu adresního místa. Výhodou je přítomnost našeptávání pro jednotlivá pole formuláře.

Tento typ našeptávání ale neumožní rychlé zadání adresy. Pro použití je výhodou i dostupná dokumentace, jak ze strany použití, tak ze strany správy (administrace). Také je k dispozici Open SwaggerAPI dokumentace[9] endpointů služby.

Několik nevýhod tento systém má. Jedná se především o absenci fulltextového vyhledávání nad celým textem adresy, což může značně zpomalit a znepříjemnit vyplňování formulářů. Zátarasem pro použití tohoto řešení by mohla být i nedostatečná aktuálnost dat, jelikož ČÚZK aktualizuje jejich datové sady, se kterými pracuje tato aplikace, jednou měsíčně.

7.4.7 Porovnání existujících řešení

Na obrázku 7.10 je shrnuto porovnání existujících řešení, co se jejich použitelnosti a jednoduchosti týče. V tabulce na obrázku jsou buňky odpovídající služby označeny barvou a obsahují slovní popis. Slovní popis je faktickou odpovědí na typ vlastnosti a barva buňky naznačuje, zdali je to žádoucí, nebo nežádoucí stav. Rozdíly vlastností služeb jsou naznačeny následovně:

- **tmavě oranžová** - označuje hodnotu vlastnosti, která je nežádoucí
- **světle oranžová** - označuje hodnotu vlastnosti, která je spíše nežádoucí, ale není tak prioritní
- **šedá** - označuje hodnotu vlastnosti, která je čistě informativního charakteru - nemá vliv na porovnání
- **zelená** - označuje hodnotu vlastnosti, která je žádoucí

Z tohoto porovnání plyne, že služba veřejného dálkového přístupu je nepoužitelná, jelikož její aplikační rozhraní nelze konzumovat z jiných domén, než cuzk.vdp.cz.

Služba Inspire AD sice nabízí chytré vyhledávání podle bodu, ale pro našeptávání je nepoužitelná (soubor s výsledky našeptání má i vyšší desítky MB).

Služba GeocodeSOE by šla použít. Její našeptávání je úsporné a následně by se dalo po výběru adresy provolat API Inspire AD, kterému bychom vlastně do fulltext vyhledávání dali již celkový validní text adresy. Toto řešení je krkolomné, ale fungovalo by.

Služba RUIAN-search aktuálně nikde veřejně dostupná není, ale její nasazení na katedře je možné. Bylo by případně jednoduché službu upravit, škálovat, dostatečně zabezpečit a v budoucnu libovolně rozšířit. Z krátké analýzy použitých technologií služby RUIAN-search vyplývá, že hledání bez diakritiky a fulltext hledání nad celou adresou nebudou velké zásahy do implementace RUIAN-search.

[9] https://app.swaggerhub.com/apis-docs/letomas/Address-search-RUIAN/1.0.0-oas3

Služba / Vlastnost	RUIAN-search	VDP	GeocodeSOE	INSPIRE AD
Umí vyhledávat bez diakritiky	NE	ANO	ANO	ANO
Můžeme provozovat lokálně	ANO	NE	NE	NE
Fulltext vyhledávání jednotlivých položek adresy	ANO	NE	NE	NE
Fulltext vyhledávání celku adresy	NE	ANO	ANO	ANO
Aktuální data	NE	ANO	ANO	ANO
Potřebuje kombinaci služeb	NE	ANO	ANO	NE
Jaké služby?	X	VDP + INSPIRE AD	GeocodeSOE + INSPIRE AD	X
Garance neměnného API	ANO	NE	ANO	ANO
Efektivní response API endpointu	ANO	ANO	ANO	NE
Veřejné API	NE	NE	ANO	ANO

Obrázek 7.10: Porovnání existujících řešení vzhledem k jejich použitelnosti

7.5 Validace vstupů

V sekci 6.5 Validace vstupů, která se týkala validace v SForms, jsem analyzoval aktuální stav validace v SForms. Nabízí se pro porovnání analyzovat i validace u jiných služeb a aplikací, abych se mohl rozhodnout, jak k validaci v nových komponentách přistupovat. V prohlížečích validace inputových elementů probíhá ve výchozím stavu až při události `onSubmit`. To je událost, která nastane, když se v HTML elementu `<form>` stiskne tlačítko.

```
<form>
  <label for="latitude">Zemepisna sirka</label>
  <input id="latitude" name="latitude_coord"/>
  <button>Submit</button>
</form>
```

Listing 7.1: Ukázka minimálního formuláře

Co průměrný uživatel při reálném používání internetu dokáže vypozorovat, je několik přístupů k validaci formulářů:

- `onSumbit` - kontrola vstupů nastává při odeslání formuláře

  Výchozí chování prohlížečů. Idea je ta, že člověk vyplní formulář a až po skončení vyplňování dojde při pokusu o odeslání k validaci. Na

to jsou uživatelé zvyklí, nicméně pokud chybí kontrola `onBlur`, může být uživatel frustrován velkým počtem zobrazených chyb po celkovém vyplnění dat.

- `onBlur` - kontrola vstupů nastává po opuštění formulářového pole

  Nejrozšířenější typ validace. Poskytuje uživatelům instantní odezvu po dokončení zadávání vstupu a přechodu kamkoliv jinam v dané stránce. Uživatel tak hned vidí, jestli zadal data ve správné podobě a nemusí se k poli již více vracet. Používá např. eshop datart.cz[10] při zadávání fakturačních údajů.

- `onChange` - kontrola vstupů nastává po každé změně hodnoty ve formulářovém poli

  Nejrychlejší typ validace. Po každé změně hodnoty vstupu dojde k validaci hodnoty vůči předem definovaným pravidlům a omezením. Tato metoda se vyplatí u vyplňování důležitých informací, kdy je uživatel obeznámen se stavem korektnosti vyplnění pole ještě před tím, než ho chce úplně opustit. Na druhou stranu může být tato metoda pro některé uživatele rušivá, protože může docházet k vizuálním změnám po minimální interakci s aplikací.

Pro použití u komponent by se nejvíce hodila kombinace variant `onSubmit` a `onChange`. První metoda je vestavěným standardem formulářů a metoda druhá je nejrozšířenější.

[10] https://www.datart.cz/

Kapitola 8

Specifikace požadavků

Na základě aktuálního stavu slovníků OFN a potřeb zlepšení SForms jsou v této kapitole vymezeny požadavky na webovou komponentu tak, aby poskytla řešení na nalezené problémy při analýze.

Pro prioritizaci nalezených požadavků je použita metoda MoSCoW, která rozděluje požadavky do čtyř kategorií. Kategorie jsou následující:

- **M** - Must have - označuje požadavky, které jsou klíčové a jejich splnění je zásadní
- **S** - Should have - označuje požadavky, které by měly být splněny, ale jejich splnění není zásadní, to znamená, že aplikace bude fungovat, ale nebude třeba tolik uživatelsky přívětivá
- **C** - Could have - označuje požadavky, které budou splněny, pouze pokud vyzbyde čas a zdroje
- **W** - Won´t have - označuje požadavky, které nebudou splněny v rámci nynější práce

Formát vyjádření požadavků spolu s prioritou dle této metody je následující:
<Priorita dle MoSCoW> - FR<číslo požadavku> - Název požadavku
<Popis požadavku>

Prioritizace požadavků proběhla po konzultaci s vedoucím práce.

8.1 Funkční požadavky - zadání zeměpisných souřadnic

Požadavky na rozšiřující knihovnu komponent jsou následující:

M - FR1.1 - Vyhledávání v mapě
Uživatel je schopen interagovat se zobrazenou mapou. Druhy interakce jsou následující: posun všemi směry, přiblížení, oddálení.

S - FR1.2 - Okamžité informování uživatele o stavu vyplnění

Při pokusu o napsání nepovoleného znaku do pole pro zeměpisné souřadnice bude o chybném vstupu uživatel okamžitě informován.

M - FR1.3 - Kontrola vstupu zeměpisných souřadnic

Formulářová pole pro zeměpisné souřadnice uživateli nedovolí vložit nepovolené znaky (písmena, ostatní znaky různé od čísel).

M - FR1.4 - Propojení informací adresy a mapy

Bude existovat oboustranná propojenost mezi adresou a polohou na mapě. Z adresy se dá zjistit poloha a naopak.

C - FR1.5 - Přepínání mezi souřadnicovými systémy

Komponenta umožní zadávání souřadnic z jiných souřadnicových systémů než EPSG:4326. Např. v Česku používaný S-JTSK / Krovak East North - EPSG:5514.

C - FR1.6 - Přepínání mezi satelitním pohledem

Mapa umožní přepnutí vizuální podoby dlaždic mapy mezi klasickým výchozím pohledem a pohledem satelitním. Nebude implementováno kvůli licenčním podmínkám satelitních snímků.

S - FR1.7 - Nalezení aktuální polohy uživatele

Komponenta umožní mapu vycentrovat na uživatelovu aktuální polohu. Tuto polohu zvýrazní v mapě.

S - FR1.8 - Uložení vybrané pozice v mapě

Komponenta bude fixovat polohu na mapě, které byla naposledy v mapě zobrazena (vybrána). To znamená, že pokud by se uživatel k mapě kdykoliv vrátil, zobrazí se mu poslední náhled.

M - FR1.9 - Zobrazení a výběr adresních míst

Komponenty umožní uživatelům zobrazit adresní místa v mapě a umožní je vybrat. Toto vybrání souvisí s FR1.4 a FR2.3.

M - FR1.10 - Výběr libovolného bodu v mapě

Uživatel je schopen zvolit libovolný bod mapě, který není adresním místem. Nutné pro potřebu zadání turistického cíle, jehož umístění nemusí odpovídat adresnímu místu.

S - FR1.11 - Označení přesnosti určení uživatelovy polohy

Navazuje na FR1.7. Označení polohy uživatele bude souviset s přesností určení jeho polohy - uživatel bude informován o přesnosti určení jeho polohy.

8.2 Funkční požadavky - zadání adresy

Požadavky na rozšiřující knihovnu komponent jsou následující:

M - FR2.1 - Jednoduché vyplnění adresy

Uživatel je schopen psát prvky adresy v libovolném pořadí. Nepotřebné informace pro zadání adresy budou skryty.

S - FR2.2 - Našeptávač při vyplňování

Komponenta umožní chytré našeptávání a nabídne uživateli nejrelevantnější adresy. Funkční i pro neúplné (např. Technická), minimální (např. Technická 6) a minimální úplné (např. Technická 5, Praha 6).

M - FR2.3 - Propojení s vybraným adresním místem v mapě

Viz FR1.4 a FR1.9. Komponenta je oboustranně propojená s mapou a po zadání adresy se zobrazí odpovídající adresní místo (existuje-li takové) na mapě.

8.3 Funkční požadavky - další opravy a optimalizace

Co se formulářů vytvořeného knihovnou SForms týče, obsahují ještě pár nedostatků, které se již netýkají přímo adres a umístění, ale jejich optimalizace by pomohla obecně jakékoliv práci s formuláři. Těmito nedostatky jsou zobrazení časového rozpětí s velkou granularitou (např. otevírací doba) a absence volby "Nevím" v otázkách typu "Ano/Ne":

S - FR3.1 - Tříhodnotový boolean

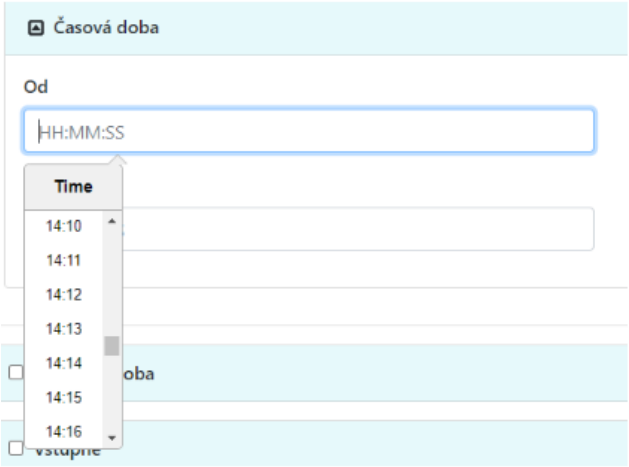
Pro obecného uživatele formulářů je žádoucí, pro zkvalitnění sbíraných dat, aby v otázkách typu Ano/Ne existovala i volba "Nevím", která je naprosto validní odpovědí a vypovídá něco jiného než odpověď "Ano" i odpověď "Ne".

Obrázek 8.1: Ukázka možného nového designu tříhodnotového booleanu v SForms

S - FR3.2 - Oprava výběru časového rozpětí

V současném stavu je výběr časového rozpětí v knihovně SForms implementován jako formulářové pole se vstupem, které má kontrolu vstupu pro formát HH:MM:SS. Dále se po kliknutí do pole vysune časový výběr, který

má ovšem moc velkou granularitu výběru. Pro nějaká použití by se tento způsob mohl příčit hlavnímu kritériu rozšíření SForms - nejrychlejšímu zadání dat.



Obrázek 8.2: Ukázka výběru času v SForms, zdroj: https://s-forms-kbss.netlify.app

Po konzultaci s vedoucím práce jsme došli k závěru, že požadavky FR3.1 a FR3.2 nebudou implementovány v rámci této práce. Důvodem je nižší priorita.

8.4 Funkční požadavky - irelevantní položky formulářů

Pro zjednodušení zadávání adresního místa nebo polohy by mohla přispět i změna struktury dat, která jsou strukturovaná dle OFN. Generování formuláře ze sémantického slovníku pojmů lze nakonfigurovat. Pro to, aby byla knihovna komponent obecná definuji ještě požadavek:

M - FR4.1 - Knihovna komponent není závislá na struktuře formuláře.

Komponenty půjdou použít dle definic polí formuláře - např. knihovna bude fungovat i pro formulář s absencí polí pro zeměpisné souřadnice.

8.5 Nefunkční požadavky

Napříč celou aplikací definuji několik nefunkčních požadavků. Ty jsem vydefinoval z požadavků na knihovnu komponent ze zadání a z důvodu potřebné interoperability s knihovnou SForms (rozhraní v českém jazyce, podpora prohlížečů).

M - NFR1.1 - Knihovna komponent je podporována na nejaktuálnějších verzích prohlížečů Google Chrome a Mozilla Firefox.

M - NFR1.2 - Webové komponenty budou otestovány alespoň třemi uživateli.
M - NFR1.3 - Uživatelské rozhraní komponent je v českém jazyce.
S - NFR1.4 - Implementované komponenty budou validovány dle metodiky RAIL.
S - NFR1.5 - Půjde zvolit i anglický jazyk pro uživatelské rozhraní komponent.
M - NFR1.6 - Našeptávací služba bude buď externí, nebo dockerizována, tak, aby bylo našeptávání jednoduše nasaditelné v aplikaci používající knihovnu komponent.

8.6 Výstup ze specifikace požadavků

Po finální specifikaci požadavků přichází prostor pro výběr technologií, které nejefektivněji splní vytyčené funkce. S přihlédnutím k počtu požadavků a jejich prioritám bude knihovna komponent plnit především požadavky spojené s adresou, souřadnicemi a jejich propojeností. Z tohoto důvodu bych nazval knihovnu komponent jako **geokomponentu**.

Zkráceně potřebuji technologie takové, abych měl k dispozici

- přístup k mapě a zeměpisným souřadnicím,
- sadu adresních míst ČR,
- našeptávání nad textem adresy (adresních míst) - dostupné vždy, když je geokomponenta využita,
- přístup k aktuální poloze uživatele.

Jako React knihovnu mapových komponent volím **React Leaflet**. Pro obsáhlou dokumentaci a nepotřebu externího API klíče se toto mapové řešení jeví jako snáze použitelné, tak při implementaci využiji tuto knihovnu.

Po zvážení kladů a záporů jsem se rozhodl, že použiji pro vyhledávání adresních míst a našeptávání službu **RUIAN-search** bakaláře Tomáše Le. Je ovšem nutné udělat několik oprav - hledání bez diakritiky, fulltext hledání nad celou adresou a oprava dockerizace k automatizovanému použití s knihovnou. Stáří dat postačuje i jeden měsíc, a to je realizovatelné. Zároveň toto řešení přináší velké plus v tom, že se jedná o službu, která bude moci být nasazena a udržována na katedře počítačů ve výzkumné skupině znalostních a softwarových systémů.

Kapitola 9
Použití SForms

Pro analýzu průběhu uživatelské interakce při zadávání adres a souřadnic do formulářů dle OFN jsem zpracoval tuto kapitolu, ve které popisuji nejčastější scénáře. Z těch potom vyvodím diagramy aktivit, které budou popisovat průběh interakce s geokomponentou celkově. To pomůže uvědomit si uživatelsky kritické části, na které budu muset při návrhu a implementaci brát ohledy.

9.1 Scénáře využití komponenty

Aby komponenta byla implementována co nejobecněji, ale zároveň použitelná pro různé situace zadávání adresy a souřadnic, je zpracováno několik scénářů, které popisují různorodé situace se zaměřením na znalost informací uživatele o adrese a/nebo poloze zadávaného objektu. Stále se odkazuji na FR2.1, nyní s ohledem na širokou použitelnost s uchováním jednoduchosti zadání informací. Abych docílil této obecnosti a přepoužitelnosti, beru v potaz i zadávání adresy v případech nespojených s *Turistickými cíli* a *Sportovišti* (u *Aktualit* a *Událostí* adresu řešit nemusím). Například při zadávání adresy bydliště uživatele nebo fakturačních údajů. To jsou situace, které by mohly být relevantní při zadávání adresy pro jiné, dosud nevzniklé nebo nefinální, slovníky.

Pro scénáře níže je nutné vytyčit několik pojmů. Pojmem **vlastní adresa** se myslí jakákoliv adresa, kterou příslušný uživatel zná, to znamená, že ji dokáže z paměti napsat. Pojmem **cizí adresa** se myslí adresa, kterou uživatel nezná, a potřeboval by ji vyhledat jiným způsobem.

Pořadí scénářů nemá žádný význam. Formát zápisu scénářů je následující:
SC<číslo scénáře> - <název scénáře>
Popis situace: <popis situace/ předpoklady>

Příklad: <příklad>

(<bližší popis/náznak možné realizace> - volitelné)

Nyní následují již samotné scénáře:

- **SC1 - uživatel zadává úplnou adresu**
  Popis situace: Uživatel má k dispozici úplnou adresu.

  Příklad: Uživatel zadává vlastní adresu. Uživatel zadává adresu vlastní firmy. Uživatel zadává adresu z papíru.

- **SC2 - uživatel zadává neúplnou adresu (viz sekce 6.3 Zadání adresy), a nezná polohu**
  Popis situace: Uživatel má k dispozici pouze neúplnou adresu a nezná polohu.

  Příklad: Uživatel zná ulici a číslo orientační - Technická 6, ale neví, kde se má toto místo nacházet. To znamená, že neví, zda-li se jedná o ulici s číslem orientačním v Praze, v Brně, nebo jinde.

  Možná realizace: Uživateli bude po vyplnění známých informací k dispozici našeptávač. Ten vypíše adresy odpovídající částečně uživatelovu vstupu. To buď uživateli dostatečně napoví a uživatel vybere některou adresu z našeptávaných, nebo nepůjde adresu zadat.

- **SC3 - zadávaná data nemají adresní místo, ale uživatel zná polohu**
  Popis situace: Uživatel zadává místo, které nemá adresu, ale uživatel zná jeho polohu.

  Příklad: Uživatel zadává turistický cíl bez adresy, například Sněžku, pramen Vltavy apod. Uživatel zadává aktualitu, nebo událost.

  Bližší popis: To nastane v případě zadání *Aktuality*, *Události* a může nastat u *Turistického cíle*. Pro tento scénář neřeší komponenta zadávání adresy, ale poskytne mu mapu, ve které bude moci uživatel hledat dle FR1.1, FR1.4, a vyplní se pouze zeměpisné souřadnice.

- **SC4 - uživatel zadává IRI / kód adresního místa**
  Popis situace: Uživatel zná kód adresního místa.

  Příklad: Uživatel zadává kód adresního místa pro turistický cíl nebo sportoviště, např. 21717451 pro Betlémskou kapli v Praze.

- **SC5 - práce s jiným souřadnicovým systémem než EPSG:4326**
  Popis situace: Uživatel zná souřadnice jiného souřadnicového systému než EPSG:4326, a chce najít a vybrat adresní místo na základě těchto souřadnic. Zná například souřadnice místa z katastru, kde se pracuje se souřadnicovým systémem EPSG:5514 (S-JTSK / Krovak East North), případně EPSG:5515 (S-JTSK/05 / Modified Krovak).

  Příklad: Uživatel zadává adresní místo na základě souřadnic systému

EPSG:5514, které zná. Například souřadnice Y: 743125,13, X: 1043330,49 pro Betlémskou kapli v Praze.

- **SC6 - uživatel zadává zeměpisné souřadnice v souřadnicovém systému EPSG:4326**
  Popis situace: Uživatel zná zeměpisné souřadnice místa v souřadnicovém systému EPSG:4326 (WSG84), jehož adresu chce zadat.

  Příklad: Uživatel zadává zeměpisnou délku 14.417491 a zeměpisnou šířku 50.08425 (souřadnice Betlémské kaple v Praze).

- **SC7 - uživatel hledá místo, jehož přesnou polohu zná**
  Popis situace: Uživatel zná polohu místa, jehož adresu chce zadat.

  Příklad: Uživatel najde v mapě Betlémskou kapli v Praze.

- **SC8 - uživatel zná neúplnou adresu a přibližnou polohu místa**
  Popis situace: Uživatel zná neúplnou adresu a přibližnou polohu místa.

  Příklad: Uživatel zná ulici a číslo orientační - Technická 6. Dále také zná přibližnou polohu a je tedy schopen identifikovat Technickou ulici a najít konkrétní adresu - Technická 6, Praha.

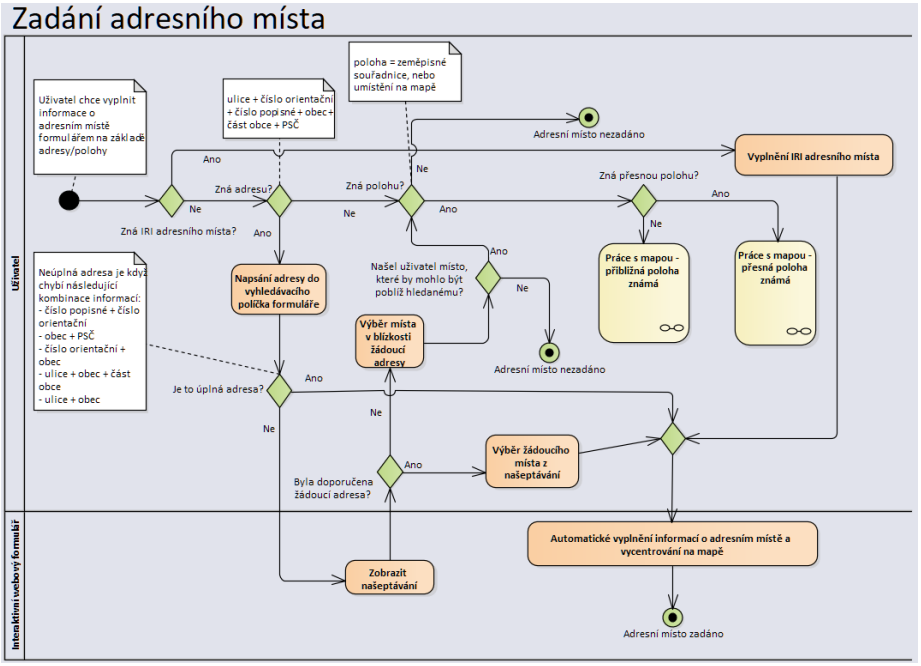
Průchody všech scénářů jsou znázorněny v následující sekci ve formě diagramů aktivit, ale již bez podrobnějšího popisu.

9.2 Diagramy aktivit

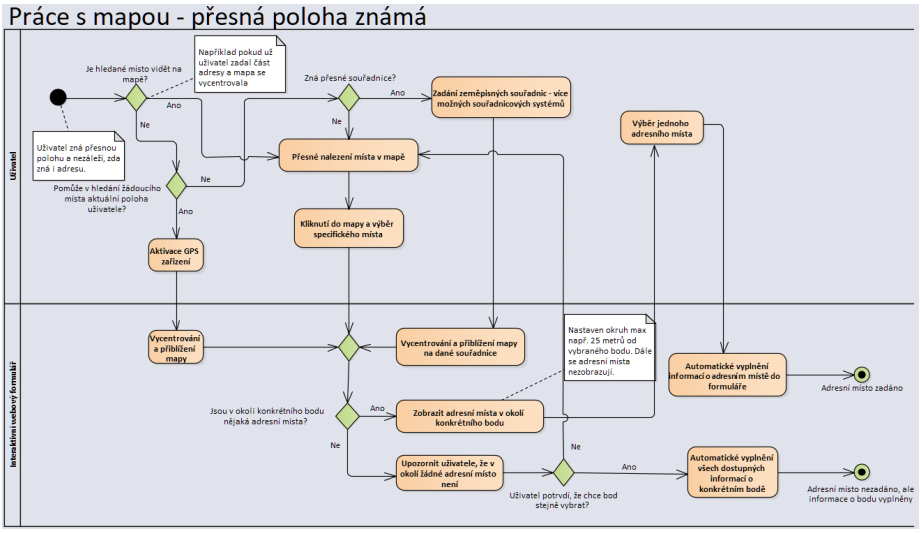
Ke znázornění průchodu uživatele procesem zadávání adresního místa jsem zvolil UML diagram aktivit. UML je velice rozšířené a jednoduché na pochopení. Aktivitu zadání adresního místa mám rozdělenou na dva logické bloky - zadávání pomocí adresy/kódu adresního místa a zadávání pomocí polohy. V diagramech na obrázcích 9.1, 9.2 a 9.3 popisuji interakci uživatele s formulářem používajícím geo komponentu.

Při modelování jsem se stále držel hlavní priority práce - nejrychlejší vyplnění/zadání dat. Proto nejdříve předpokládám zadávání kódu adresního místa, poté adresy a až na konec případně polohy. Je to čistě z důvodu rychlosti. Samozřejmě v realitě může uživatel zvolit cestu jinou.

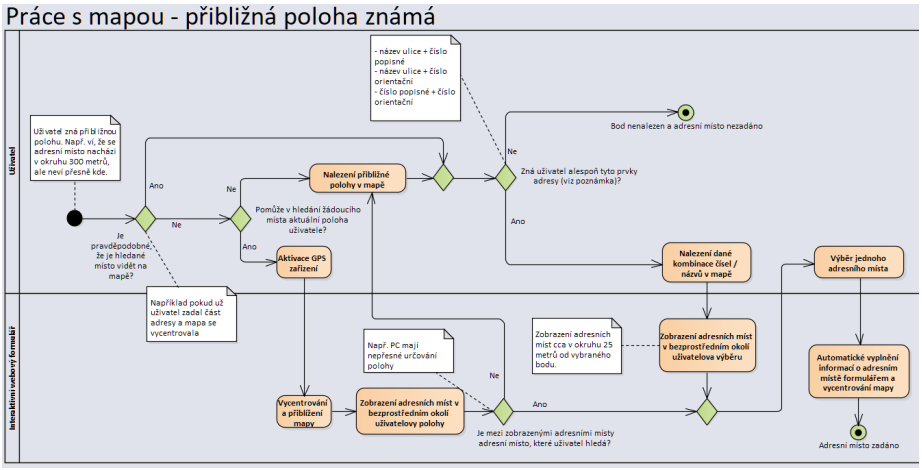
Proces zadávání polohy je rozdělen formálně na dva poddiagramy, a to z důvodu dvou možných přístupů k nalezení požadované polohy. V diagramech používám pojmy definované v podsekci 6.4.1 - přesná poloha a přibližná poloha. A vlastně i takto jsou rozděleny ony dva diagramy. V jistých částech jsou si podobné, hlavně co se týče typu asistence poskytované uživatelům. Liší se pak vstupními znalostmi uživatelů - např. diagram na obrázku 9.2 skončí vždy zadáním informací.



Obrázek 9.1: Hlavní diagram aktivit



Obrázek 9.2: Poddiagram aktivit pro přesnou polohu známou uživatelem



Obrázek 9.3: Poddiagram aktivit pro přibližnou polohu známou uživatelem

Kapitola 10

Rozšiřitelnost knihovny SForms

Pro završení části analýzy je nutné se zaměřit i na to, jakým způsobem lze knihovnu SForms rozšířit. Co se slovem rozšířit myslí? Myslí se tím přidání funkcionality nebo změny vzhledu výsledného vizuálu formuláře. To vše stále nad propojenými daty, se kterými knihovna pracuje. A takový způsob externího rozšíření je zatím jediný. Knihovně SForms lze poskytnout takzvaná mapovací pravidla, ve kterých oznámíme SForms, že nějaká data bude procesovat místo implicitní komponenty SForms jiná rozšiřující komponenta - např. geo komponenta.

10.1 Mapovací pravidla

Idea mapovacích pravidel je následující: zadáme SForms jaká data chceme zobrazovat, a to formou výrazu, který vrací boolean (mapovací pravidlo), a určíme, jakou React komponentou tato data chceme procesovat. Idea je celkem přímočará. Nicméně při analýze mapování dat týkajících se adresy a polohy jsme narazili společně s vedoucím práce na jistá úskalí. Museli jsme vymyslet obecný způsob mapování tak, abychom byli schopni **jednoznačně**, ale zároveň co nejvíce **obecně**, určit, která data formuláře popisují polohu / adresní místo a zdali se váží k jedné lokaci - pro případ, kdy by se ve formulářích objevilo několik různých adres, nebo zeměpisných souřadnic.

Jednoznačně znamená, že je zachována integrita dat. A to znamená, že data popisující jedno konkrétní místo (např. umístění turistického cíle) budou propojena a zprocesována geo komponentou spolu, a ne například s daty o umístění majitele turistického cíle. Nebo například nebudou nějaká data spojená s jednou lokací chybět.

Obecně znamená, že nechceme být vázaní nutně jen na případy použití vytyčené v úvodu práce (*Turistické cíle*, *Sportoviště*, *Události*,...), ale je žádoucí, aby komponenta zvládla v budoucnu zpracovat jakoukoliv strukturu formuláře, který bude obsahovat data spojená s polohou nebo adresou/adresním místem. Znovu se dá odkázat na FR2.1 společně s FR2.3 a FR1.4.

10.2 Mapování geo komponenty

Aby bylo možné geo komponentu jednoznačně namapovat, byl vytvořen dodatečný atribut *is-part-of-location* u těch dat, která patří k sobě - popisují jeden objekt a formují celek. Atribut *is-part-of-location* je tedy unikátní identifikátor propojených dat.

Pro obecnost komponenty je nutné zajistit, aby fungovala nad různě strukturovanými daty, kdy jsou otázky (data) spolu související různě zanořené do otázek (dat) jiných. Jelikož vycházíme z otevřených formálních norem, které ve slovnících zmíněných v úvodu definují entitu *Umístění* (obsahuje adresu a polohu), tak se nabízí variant omezené množství:

(úroveň odrážky naznačuje hloubku zanoření propojených dat)

Příklad 1:

- Umístění
  - Adresa
  - Geometrie
    - Zeměpisná šířka
    - Zeměpisná délka
    - Irelevantní otázka pro geo komponentu

Řešení 1: zobrazení otázek následovně

- Umístění
  - **Geo komponenta** - Adresa (zobrazení pomocné mapy + zobrazení adresy + zobrazení zeměpisných souřadnic)
  - Geometrie
    - **SKRYTO** - Zeměpisná šířka
    - **SKRYTO** - Zeměpisná délka
    - Irelevantní otázka pro geo komponentu

Příklad 2:

- Umístění
  - Geometrie
    - Zeměpisná šířka
    - Zeměpisná délka
    - Irelevantní otázka pro geo komponentu
  - Adresa

Řešení 2.1.: zobrazení otázek následovně

- Umístění

- **Geo komponenta** - Adresa (zobrazení pomocné mapy + zobrazení adresy + zobrazení zeměpisných souřadnic)
- Geometrie
  - **SKRYTO** - Zeměpisná šířka
  - **SKRYTO** - Zeměpisná délka
  - Irelevantní otázka pro geo komponentu
- **SKRYTO** - Adresa

Řešení 2.2.: zobrazení otázek následovně

- Umístění
  - Geometrie
    - **SKRYTO** - Zeměpisná šířka
    - **SKRYTO** - Zeměpisná délka
    - Irelevantní otázka pro geo komponentu
  - **Geo komponenta** - Adresa (zobrazení pomocné mapy + zobrazení adresy + zobrazení zeměpisných souřadnic)

Aby bylo docíleno tohoto namapování, byl vytvořen algoritmus pro mapovací pravidlo, který seskupoval relevantní otázky k jedné lokaci (*is-part-of-location*) a pak se rozhodoval, jaká data (na jakém místě / v jaké hloubce / na jaké úrovni propojených dat) budou zpracována geo komponentou tak, aby data k jedné lokaci byla propojená, ale aby nebyla narušena struktura formuláře a byla zachovaná pohodlná práce s formulářem a zvýšena rychlost vyplnění. Tento algoritmus bude fungovat i pro situace, kdy je součástí formuláře pouze adresa, nebo pouze poloha.

Tím vším výše zmíněným bychom chtěli splnit očekávání, že geo komponenta se bude mapovat efektivně s co nejmenším zásahem vývojáře (např. umělé vstupní argumenty do komponenty), případně uživatele (např. výběr typu zadávání). Dále, že mapování bude flexibilní - obecné - FR4.1. A také, že formuláře nad slovníky sad OFN budou pokryty všechny.

Abych toho docílil v plném rozsahu, nabízí se několik možností:

- preprocessing dat, který by před mapováním přidal atribut datům JSON-LD
  - inference - odvození dalších dat z dat existujících
- omezení případů, jak se komponenta bude mapovat
  - tím se sníží flexibilita, ale za to bude vždy udržovat integritu dat a fungovat dle očekávání
- změny při generování formuláře
  - změny dat JSON-LD již při generování (pomocné atributy, označení pořadí,...)
  - v aktuální situaci se toto používá přeneseně, tzn. ne při generování formuláře, ale před jeho použitím knihovnou SForms

10.2.1 Aktuální nevýhody mapování

Při analýze překážek výše jsem narazil na drobnost ze strany knihovny SForms, která ovlivňuje všechny rozšiřující knihovny SForms. V aktuálním stavu se spustí přemapování komponent po každém najetí myší na nadpis libovolné sekce ve formuláři. To znamená, že se pro všechny otázky daného formuláře kontroluje, zdali neexistuje nějaké pravidlo, které by je mapovalo na nějakou jinou komponentu. A po přidání algoritmu na nalezení souvisejících otázek je toto přemapování ještě o něco výpočetně náročnější. Zkoumání dopadů a případných souvislostí s mojí knihovnou komponent je mimo rozsah této práce.

Část III

Návrh a implementace

Kapitola 11

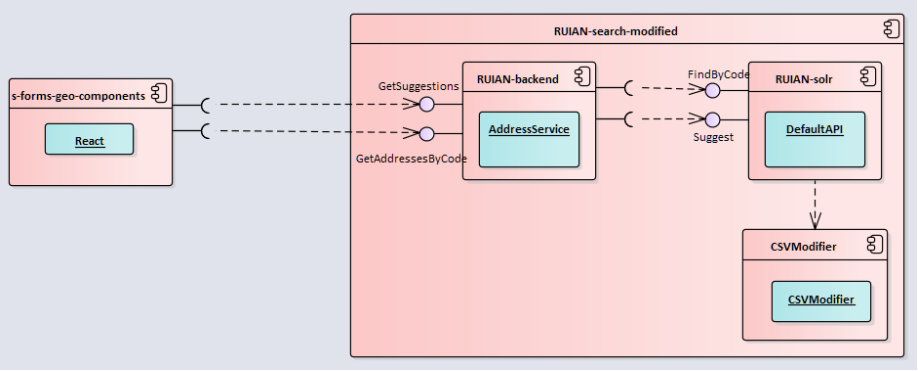
Návrh

V této kapitole rozeberu, jaký je přístup k návrhu knihovny poskytující rozšiřující komponenty pro knihovnu SForms. Také zmíním, jak přistoupím k rozšíření aplikace RUIAN-search (podsekce 7.4.6) tak, aby byl splněn požadavek FR2.2 a naznačím jak spolu tyto dva oddělené systémy budou spolupracovat.

11.1 Architektura aplikace

Pro dosažení kompletního konečného řešení je potřeba na aplikaci nahlížet jako na souhru dvou systémů - frontendu a backendu. Frontend je hlavním tématem této práce, ale pro splnění vytyčených požadavků je nutné využít i služby backendové, které budou realizovat našeptávání textu adresy.

Na diagramu níže je znázorněna struktura funkčního řešení, které splňuje cíle mojí bakalářské práce. Komponenta **s-forms-geo-components** (odpovídá cílové knihovně webových komponent) je frontendem a komponenta **RUIAN-search-modified** zaštiťuje použití upravené verze aplikace RUIAN-search. Zároveň lze z diagramu vyčíst, že moje knihovna reactových komponent konzumuje pouze rozhraní AddressService. Z důvodu celistvosti jsem v diagramu namodeloval i zbylou část upravené aplikace RUIAN-search.



Obrázek 11.1: Celkový diagram komponent

11.2 Backend - našeptávácí služba

Úlohou backendu, jenž vychází z projektu RUIAN-search Bc. Tomáše Le, je poskytnou dvě funkcionality. Těmi je fulltextové našeptávání textu adresy a hledání adresy pomocí kódu. Fulltextové našeptávání v původním aplikaci chybí. Na to bude potřeba vzít zřetel při návrhu rozšíření.

Originální projekt obsahuje i frontendovou část, která ale pro moje cíle potřeba není. Bude výhodné původní architekturu zredukovat. Použiji jen backendovou část aplikace, která vystavuje REST rozhraní, a rozšířím vyhledávající službu Apache Solr tak, aby poskytovala našeptávání pro text adresy.

Zredukovaná architektura aplikace je vidět na obrázku 11.1 v komponentě RUIAN-search-modified.

Interní struktura Spring boot Java aplikace dodržuje architekturu MVC (model-view-controller) s tím, že v Javě jsou implementovány jen části model a controller. S touto architekturou jsem se již setkal v průběhu studia, takže orientace v aplikaci je snadná.

11.2.1 Rozšíření a změna konfigurace Apache Solr

Aby Spring boot backend aplikace měla k dispozici dotazování na výsledky našeptávání, je potřeba změnit konfiguraci Apache Solr, která se týká indexace dat a dotazování nad těmito daty.

Indexace dat

Pro úpravu pravidel indexace dat jsem zvolil nejrychlejší možnou cestu. Je pravděpodobné, že to není efektivní, ale indexace v reálném provozu neprobíhá často a hlavní prioritou pro mě je funkční našeptávání bez ohledu na efektivnost.

Před samotným indexováním je nutné v Apache Solr vytvořit tzv. jádro, kterému bude přiřazeno schéma, jak má daná data zpracovávat.

Apache Solr indexuje data z RÚIAN ve formě csv souborů. Před jejich posláním službě Apache Solr probíhá jejich procesování, kdy se do csv souborů uměle přidávají sloupce tak, aby se s daty pak dalo lépe a uživatelsky pohodlně pracovat. Přesně tímto způsobem je možné přidat další sloupec, který bude obsahovat kompletní text adresy na daném řádku. Sloupec se jmenuje Text_adresy. Aby šlo pole Text_adresy použít pro dotazování, musí dojít k aktualizaci schématu tak, aby Apache Solr věděl, že specifický sloupec csv souboru obsahuje hodnotu Text_adresy.

Dotazovací pravidla

Platforma Apache Solr je nadstavbou nad vyhledávačem Apache Lucene. Apache Lucene je výkonný a obsáhlý vyhledávací engine ve formě knihovny napsané v Javě.[Fou23] Dotazování do Apache Solr má proto velmi podobnou

strukturu jako v Apache Lucene. Jednou z cest, jak se dotazovat neúplně na data je pomocí tzv. wildcards. Těchto wildcards, které mají podobu znaků, je několik, a fungují jako při dotazování jazykem SQL:

- `*` - zastupuje 0 až N libovolných znaků
- `?` - zastupuje přesně jeden libovolný znak

Po krátké analýze a zkoumání dokumentace Apache Solr[1] i Apache Lucene[2] mi přijde použití těchto wildcard při dotazování na návrhy výsledků jako nejrychlejší a zároveň nejjednodušší cesta.

Našeptávání bude probíhat nad prvkem Text_adresy a bude obsahovat wildcards. Zjednodušeně půjde o dotaz typu: [**uživatelův neúplný textový vstup**]*, například: **Technicka***.

11.2.2 Úprava aplikační logiky pro fulltextové našeptávání

Aby si Spring boot aplikace vystavující rozhraní externím službám (především hlavně mé knihovně komponent) byla schopna získat data z dotazu na fulltext vyhledávání, je nutné rozšířit část aplikace, která komunikuje se službou Apache Solr. Další úpravy, jako rozšíření rozhraní a kontroleru budou formální záležitosti. To je dáno výbornou použitelností frameworku Spring boot, který obsahuje interní anotace pro rychlou tvorbu opakujících se fragmentů kódu.

Další návrhová změna se dotýká modelu, se kterým aplikace pracuje. Aby správně fungovalo propojení s Apache Solr, je nutné rozšířit třídu Address o text adresy, který je využit právě pro našeptávání.

11.2.3 Dockerizace

Pro jednoduché a plynulé nasazení je potřeba upravit původní RUIAN-search docker-compose a efektivněji rozdělit služby a jejich kompetence - NFR1.6. Rychlá úprava je odstranění výchozí frontendové aplikace, protože pro mé použití potřebuji pouze logiku našeptávání. Původní docker-compose fungoval tak, že i po spuštění příkazu `docker-compose up` musel uživatel spouštět další skript, aby proběhla prvotní indexace dat.

Tento jeden zbytečný krok navíc lze odstranit a logiku stahování a indexování dat vložit přímo do docker image Apache Solr. Tento image jsem pracovně nazval RUIAN-solr. Postup spuštění RUIAN-solr bez dalších skriptů by byl následující:

1. Pokud neexistuje Apache Solr jádro s názvem **ruian**, vytvoří se
2. Pokud nejsou stažená a indexovaná žádná data, stáhnou se a naindexují do jádra **ruian**.
3. Pokud jsou data stažená a naindexovaná, ale jsou zastaralá, stáhnou se nová data a naindexují do jádra **ruian**.

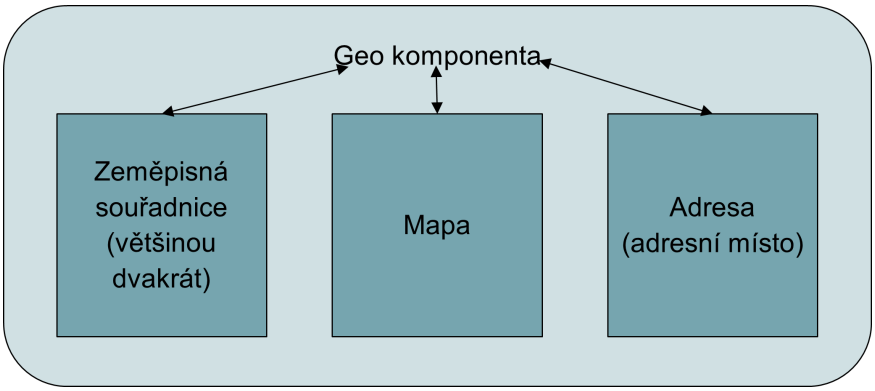
[1] https://solr.apache.org/guide/8_7/$the-standard-query-parser.html$
[2] https://lucene.apache.org/core/2_94/$queryparsersyntax.html$

4. Pokud jsou data stažená a aktuální, nic se neprovede.
5. Spustí se služba Apache Solr s aktuálními RÚIAN daty.

11.3 Návrh komponent

Hlavní geo komponentu jsem rozdělil na tři logické bloky : Adresa, Mapa, Souřadnice. Adresa i souřadnice jsou podkomponenty geo komponenty, které jsou založené na JSON-LD objektech formuláře. Mapa je pomocná podkomponenta, skrze kterou se realizují požadavky na zadání zeměpisných souřadnic, vizualizaci a výběr adresních míst.

Navíc toto logické rozčlenění umožní podkomponenty v geo komponentě v budoucnu různě zobrazovat podle toho, s jakými daty se pracuje - například jestli se má brát v potaz adresa, nebo pouze zeměpisné souřadnice.



Obrázek 11.2: Návrh geo komponenty

Co se uživatelského rozhraní týče, nechci se příliš odchylovat od vizuálu základní knihovny SForms. Uživatelé už jsou na tuto výchozí podobu zvyklí a nenašel jsem jiný důvod, proč by bylo potřeba zasahovat do původní stylizace.

11.3.1 Adresa

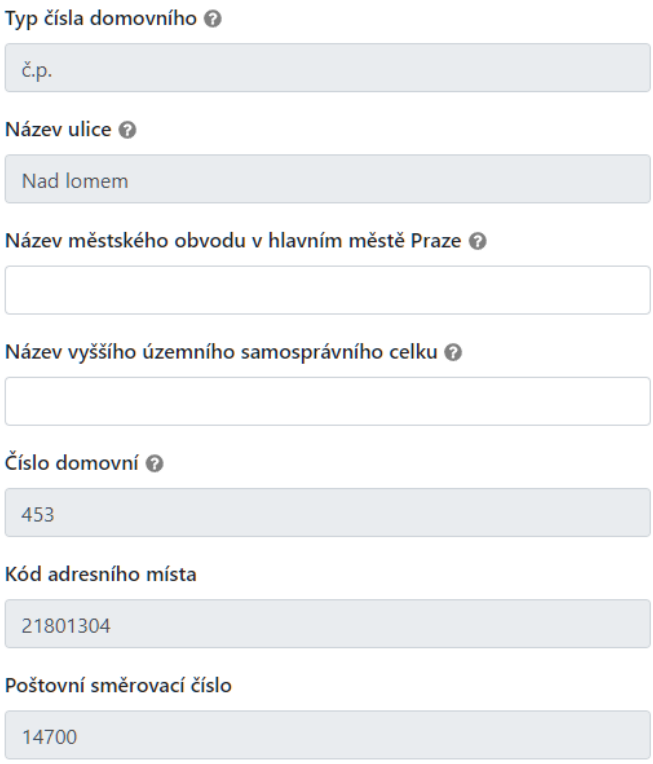
Jak jsem již zmiňoval, nebudu zasahovat do vizuální podoby komponent, která vychází z knihovny SForms.

Aby se udržovala datová konzistence v rámci celého formuláře, je nutné kontrolovat uživatelovy vstupy při každé změně. V rámci geo komponenty se nicméně vyplatí zvolit přístup jiný, jelikož je počet změn, které vyvolávají změny další, omezený. Poněvadž spolu mají komunikovat tři logické komponenty (adresa, mapa, souřadnice), tak by se kontrola datové konzistence při každé změně hodnoty jakéhokoliv formulářového pole stala značně komplexní a matoucí operací. Pro celkové zjednodušení využiji funkčnosti SForms, která se týká uzamčení (anglicky disabling) formulářových polí. To je znázorněno zešedivěním pole a zároveň prohlížeč nedovolí další změnu hodnoty v daném poli. Toto vše lze využít pro jednoduché splnění požadavku FR1.4.

Tento typ návrhu lze použít u situace, když má uživatel vybrané adresní místo z mapy. Nastíněný postup je následující:

1. Výběr adresního místa z mapy
2. Uzamčení formulářových polí v sekci *Adresa*

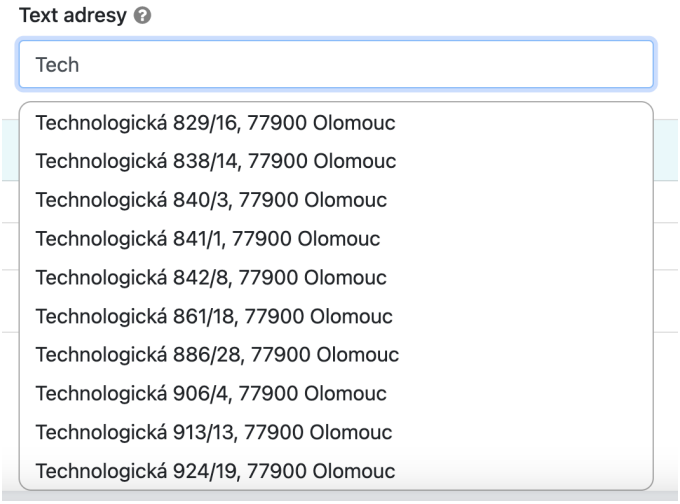
Na tuto situaci naváži v další podsekci o mapě.



Obrázek 11.3: Ukázka uzamčených (disabled) polí

Našeptávací pole

U pole s našeptáváním se inspiruji běžnou praxí při návrhu webových aplikací. Nejběžnějším postupem je zobrazit seznam nejpřesnějších výsledků po zadání vstupu. Aby nedocházelo ke zbytečnému hledání, uvažuji našeptávání až pro řetězce délky vetší než 3 znaky. Ze stejného důvodu volím maximální počet návrhů textu adresy na 10.



Obrázek 11.4: Ukázka pole s našeptáváním

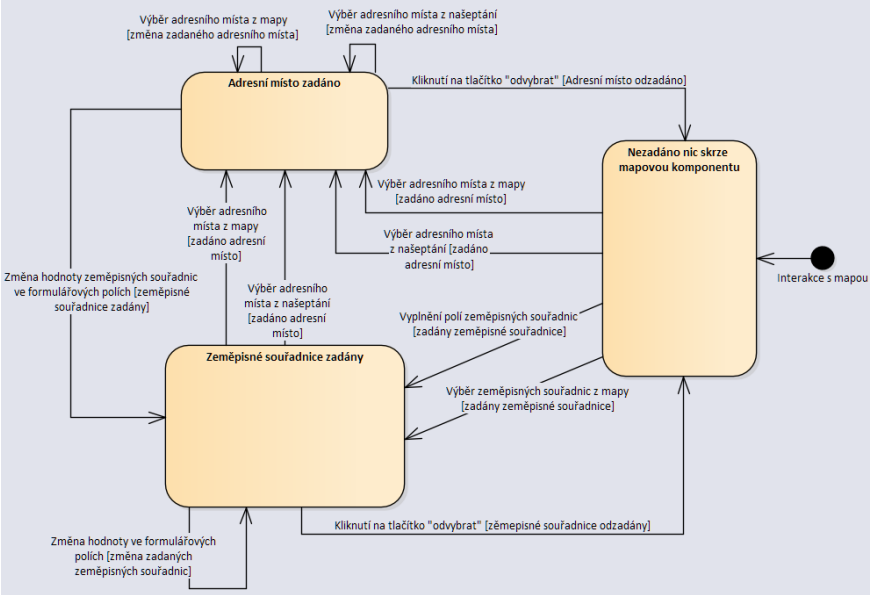
11.3.2 Mapa

Mapová komponenta je úplně novým přídavkem do ekosystému SForms. Spojují se zde informace o zeměpisných souřadnicích a adresních místech. Co se vizuální stránky týká, ponechám dlaždice mapy ve výchozím stylu React leaflet. Rozšíření výchozí mapové komponenty je jednoznačně potřeba pro plnění požadavků efektivně a uživatelsky přívětivě.

Pro splnění požadavku FR1.7 a FR1.11 je nutné umožnit uživatelům najít svou vlastní polohu a znázornit její přesnost. Zde není potřeba vymýšlet nic nového. Ikonka GPS je u všech mapových služeb velice podobná. A pro naznačení přesnosti použiji, také masově používanou, kruhovou oblast v mapě, jejíž průměr závisí na přesnosti lokalizace.

Mapa to středobod, ze kterého lze rychle a přesně určit zadané souřadnice nebo adresní místo. Pro tyto účely je zásadní, aby i zde se udržovala konzistence zadaných dat. Nejnázornějším zachycením situací, kdy se mění data ve formuláři, je stavový diagram na obrázku 11.5.

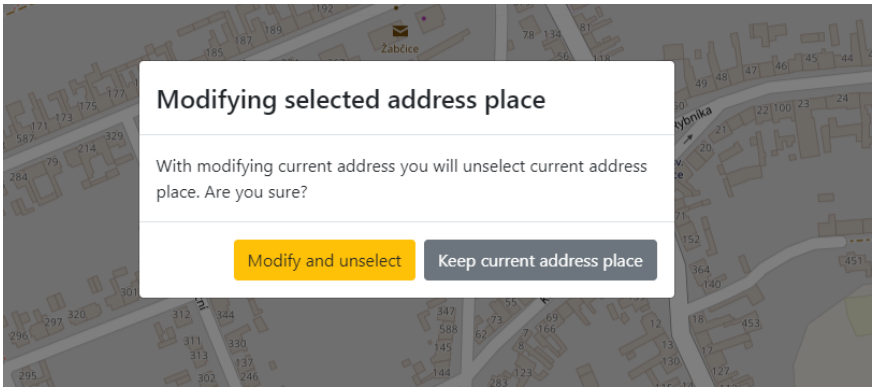
Pro udržování konzistence navrhuji jednoduché řešení, které sice o něco zpomalí vyplňování formuláře, což je sice prioritou této práce, ale zase to mnohonásobně zpřehlední práci pro uživatele, a přehlednost v této situaci považuji za přednější. Rychlost vyplňování formuláře se použitím geo komponenty i tak razantně zvýší. Tímto řešením je při všech přechodech ze stavu *Adresní místo zadáno* do stavu jiného **vynulovat formulářová pole adresy**.



Obrázek 11.5: Stavový diagram mapové komponenty

Společně s uzamykáním polí adresy (zmíněném v podsekci 11.3.1 Adresa) tím odpadne potřeba kontrolovat integritu zvoleného adresního místa s hodnotami ve formulářových polích v sekci Adresa. Formulářová pole s hodnotami zadaného adresního místa tedy nepůjde modifikovat, budou uzamčená. Pro změnu musí uživatel nejdřív adresní místo vynulovat, nebo zadat adresní místo jiné.

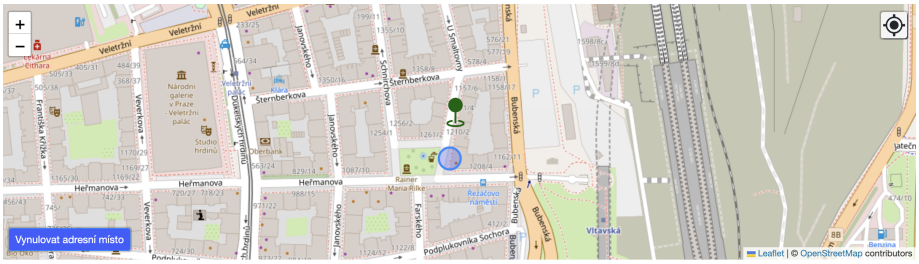
Pro přechod ze stavu *Adresní místo zadáno* bude uživatel muset svou volbu potvrdit, aby se nestalo, že uživatel nějakou chybou vynuloval svůj vyplněný formulář. Nejběžnější cestou, jak od uživatele vyžadovat informované potvrzení, je vyskakovací okno (anglicky popup). To chytne uživatelovu pozornost a ten se již rozhodne, jak pokračovat dále.



Obrázek 11.6: Vyskakovací okno s informativním textem a tlačítky

Požadavek FR1.2 lze efektivně přiřadit další uživatelské interakci - vynulování formuláře (odvybrání adresního místa / souřadnic). Nabízí se tuto

funkcionalitu realizovat tlačítkem. Taktickým umístěním v levém dolním rohu, kde se ještě žádná vizuální komponenta nenachází, poskytne mapa uživatelům další funkcionalitu, a zároveň zobrazením tlačítka ve stavech *Adresní místo zadáno*, nebo *Zeměpisné souřadnice zadány*, upozorní uživatele, že se stav aplikace změnil.



Obrázek 11.7: Ukázka návrhu mapy

11.3.3 Zeměpisné souřadnice

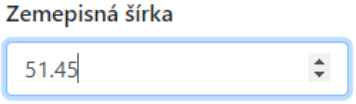
Formulářová pole zeměpisných souřadnic nevyžadují moc změn oproti knihovně SForms. Pro potřeby splnění požadavků FR1.2 a FR1.3 je nutné přidat výchozím prvkům kontrolu a omezení vstupu. Omezení jsou následující:

- vstupy musí být čísla
- čísla musí být v rozmezí hodnot validních zeměpisných souřadnic

K tomuto lze nejjednodušeji přistoupit pomocí použití atributu s hodnotou `type="number"` na základním HTML prvku `<input>`. A pro omezení intervalu hodnot lze využít atributů `min` a `max`. Výsledný prvek by tedy mohl vypadat takto:

```
<input type="number" id="lat" name="lat" min="-90" max="90">
```

Vykreslené inputové pole v knihovně React-Bootstrap lze vidět na obrázku níže.



Obrázek 11.8: Formulářové pole s omezeními

Kapitola 12

Implementace komponent

V této kapitole se věnuji implementaci webových komponent a technologiím, které jsem za tím účelem použil. Implementované řešení je udržováno v repozitáři[1] na GitHubu.

12.1 TypeScript

K vývoji je použit TypeScript[2]. To je open-source programovací jazyk udržovaný firmou Microsoft. TypeScript využívám i přesto, že původní knihovna SForms je stále převážně JavaScriptová. Pro použití s komponentou SForms to ničemu nebrání a pro vývojáře může být práce s TypeScriptem velice jednodušší a příjemnější než s JavaScriptem. TypeScript totiž představuje například statické typování. Tím umožňuje kontrolu typů, takže vývojář má větší přehled o proměnných, se kterými pracuje a o jejich datovém typu.

12.2 Použité technologie

Framework použitý pro tvorbu komponent je React (ver. 17.0.2, kvůli kompatibilitě s SForms). Další použité knihovny jsou následující:

- jsonld
  - pro práci s daty ve formátu JSON-LD
- react-leaflet
  - mapová komponenta postavená nad knihovnou leaflet
- leaflet
  - potřebná k fungování knihovny react-leaflet
- axios
  - tvorba HTTP requestů a konzumace webových API

[1] https://github.com/VojtechLunak/s-forms-geo-components
[2] https://www.typescriptlang.org

- babel
  - kompilace kódu
- react-bootstrap
  - knihovna nastylovaných UI komponent
- microbundle
  - balíčkovač aplikace
- a další pomocné knihovny a knihovny, na které jsou výše zmíněné knihovny závislé

12.3 Mapovací pravidlo geo komponenty

Prvním krokem k použití jakékoliv rozšiřující komponenty je najít způsob, jak ji namapovat na formulář SForms. K tomu slouží mapovací pravidla (viz sekce 10.2 Mapovací pravidla). V průběhu implementace jsem narazil na několik problémů, které vyžadovali konzultace s vedoucím práce a někdy nevedly k úplně jasnému závěru jak postupovat, z důvodu velké komplexnosti problému jednoznačnosti a obecnosti. V aktuálním stavu je použito mapovací pravidlo nastíněno v kapitole Mapování geo komponenty. To znamená, že prozatím funguje mapování minimálně pro existující formulář nad daty *Turistický cíl*.

Mapovací pravidlo, procesované knihovnou SForms, je vydefinováno ve třídě `GeoComponents`. Přistup k němu je pomocí funkce **getComponentMappings()**.

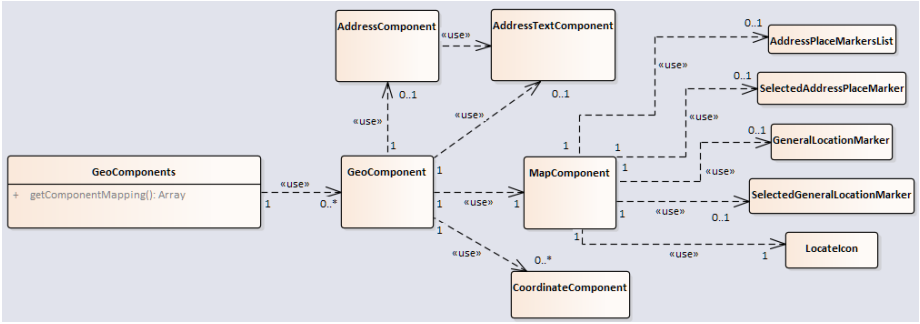
12.4 Geo komponenta

S odkazem na návrh komponent je realizována i implementace. Po odpovědi true mapovacího pravidla je pro daná data použita geo komponenta. Ta si nalezne data společná pro danou lokaci (pokud taková data jsou přítomná). Struktura geo komponenty pak vypadá takto:

- lokální data
  - cache otázek spojených s danou lokací
  - reference React leaflet mapové komponenty, která je součástí mé mapové komponenty
- callback funkce
  - onUserLongitudeInput(longitude)
  - onUserLatitudeInput(latitude)
  - onMarkerLocationChange(latitude, longitude)

  - onAddressPlacePicked(addressPlace)
  - onAddressPlaceSuggestionClick(addressCode)
  - onAddressPlaceTextModified(onChangeEvent)
  - onCoordinateValueModifiedWhenAddressPlaceIsSelected(onChangeEvent)
- pomocné funkce vracející boolean pro rozhodnutí, o jaký typ geo komponenty se jedná
  - isAddressComponentQuestion - určuje, zdali je vstupem do komponenty JSON-LD objekt adresy
  - locationContainsAddress
  - locationContainsCoordinates
- render
  - `AddressComponent`
  - `AddressTextComponent`
  - `MapComponent`
  - `CoordinateComponent` (dvakrát - zem. šířka + zem. délka)
  - + logika rozhodování, co a za jakých podmínek v jakém pořadí zobrazit

Na diagramu tříd níže je znázorněná kompletní struktura knihovny. Je vidět, že větší množství komponent a logiky spravuje mapová komponenta. To je logické, protože právě v mapové komponentě se spojuje stav související jak s adresou, tak se souřadnicemi.



Obrázek 12.1: Diagram tříd knihovny s-forms-geo-components.

Do komponent, které se renderuji, pak vstupují callback funkce v rámci React props. Tyto funkce se pak ve specifické chvíle volají, a spouští logiku v rodičovské geo komponentě. Tímto způsobem probíhá modifikace dat formuláře na jednom místě a data jsou vždy aktuální pro jiné komponenty.

12.5 Mapa

V mapové komponentě, která je hlavním přínosem geo komponenty, dochází především k propagaci uživatelovy interakce do nadřazené geo komponenty. Mapa je prostředníkem, který poskytuje uživatelské rozhraní pro jednodušší výběr souřadnic a adresních míst, a na pozadí nezpracovává velké množství logiky. Většinu rozhodování přenechává pro specifičtější komponenty:

- `AddressPlaceMarkersList` - obstarává vykreslení adresních míst v mapě a logiku výběru adresního místa
- `SelectedAddressPlaceMarker` - vykresluje zvolené adresní místo ve formuláři
- `GeneralLocationMarker` - obstarává logiku výběru zeměpisných souřadnic bez adresního místa
- `SelectedLocationMarker` - vykresluje zvolené zeměpisné souřadnice bez adresního místa
- `LocateIcon` - implementuje lokalizaci zařízení uživatele

Při interakcí s mapou má uživatel volnější omezení, než v případě interakce s formulářem přímo. S odkazem na návrh mapy (podsekce 11.3.2) lze z mapy vybírat souřadnice i adresní místa, zároveň ale mapa zobrazuje vyplněná data z formuláře (pokud jsou validní) - souřadnice nebo adresní místo. Pro udržení konzistence dat jsem implementoval uzamykání formulářových polí při zvoleném adresním místě. Mapa má ale volnější omezení, a dovolí uživateli měnit hodnoty (zvolené adresní místo / souřadnice) interakcí s mapou, bez vyžadování potvrzení o změnách. Toto potvrzení formou vyskakovacího okna by uživatel musel poskytnout, pokud by chtěl do již vyplněných hodnot přistupovat přímo.

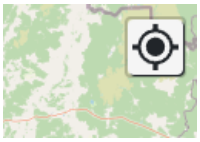
12.5.1 Lokalizace

Pro implementaci požadavku FR1.7 jsem zvolil běžnou variantu - Geolocation API. Geolocation API umožňuje uživatelům poskytnou jejich lokaci webovým aplikacím, pokud si tak přejí. Z důvodu ochrany soukromí je uživatel vyzván, zdali chce opravdu sdílet informace o své poloze.[Cor23b]

Geolocation API nabízí velké množství informací o lokaci - souřadnice, přesnost, rychlost pohybu, nadmořskou výšku. Pro moje využití stačí pouze informace o souřadnicích a přesnost.

Do pravého horního rohu mapy jsem implementoval tlačítko `"Najdi mou polohu"`, které má podobu běžného `"GPS"` tlačítka. Logiku lokalizace realizuje `MapComponent` po kliknutí na komponentu `LocateIcon`.

Atribut přesnosti z Geolocation API použiji pro implementaci požadavku FR1.11. Po kliknutí na `LocateIcon` se prohlížeč uživatele dotáže, jestli chce opravdu sdílet údaje o své poloze (mechanismus je zabudovaný v prohlížeči) a



Obrázek 12.2: Implementované tlačítko `"GPS"`.

pokud souhlasí, `MapComponent` využije poskytnutá data a vycentruje a přiblíží mapu na uživatelovu polohu. Musel jsem ovšem přidat jedno omezení. Jelikož stolní počítače nemají GPS čip, jejich lokalizace probíhá analýzou jejich internetového připojení. Jenže tento způsob nemůže zaručit, že nalezené souřadnice odpovídají skutečné poloze zařízení. V některých případech Geolocation API vracela tak moc malou přesnost, že se kruhová oblast reprezentující přesnost zobrazila přes velká území. Pro tyto případy jsem upravil nastavování přiblížení mapy po povolení sdílení polohy, aby velký zoom uživatele nenaváděl k tomu, že je poloha přesná.

12.5.2 Manipulace s adresními místy

Komponenta `AddressPlaceMarkersList` realizuje zobrazování adresních míst v mapě a logiku výběru adresního místa.

Zobrazení míst probíhá efektivně - implementoval jsem dvě pravidla, jak k zobrazování přistupovat:

- **úroveň přiblížení** - pokud je hodnota atributu `zoom` React leaflet komponenty `MapContainer` rovna, nebo větší než 19, adresní místa se zobrazí
- **počet zobrazených adresních míst** - zobrazují se pouze adresní místa taková, která jdou vidět v mapě přiblížené na úroveň 19 a více

12.6 Zeměpisné souřadnice

Zeměpisné souřadnice spravuje jednoduchá komponenta `CoordinateComponent`. Je to jednotná komponenta, která se vykresluje dvakrát a pracuje s rozdílnými daty - zeměpisná šířka a zeměpisná délka. Abych splnil FR1.2 a FR1.3 dle návrhu (podsekce 11.3.3), musel jsem modifikovat formulář pro *Turistický cíl*. To není dlouhodobé řešení, ale řešit nekorektnost atributů některých JSON-LD objektů není v zadání této práce.

Knihovna SForms umí rozlišovat z JSON-LD struktur typy objektů a omezení jejich hodnot, jeli to definováno. Tyto typy a omezení pak přenese do základních React-Bootstrap komponent. Pro opravení zeměpisných souřadnic stačilo k otázkám zeměpisných souřadnic přidat tyto páry:

```
"has-layout-class": "number",
"has-datatype": "xsd:int"
```

Abych realizoval i intervalová omezení hodnot, přidal jsem tyto řádky

```
"xsd:maxInclusive": 180,
"xsd:minInclusive": -180
```

pro zeměpisnou délku a tyto

```
"xsd:maxInclusive": 90,
"xsd:minInclusive": -90
```

pro zeměpisnou šířku.

12.7 Adresa

Komponenta `AddressComponent` má nejméně přidané logiky. Její hlavní význam spočívá v zapouzdření otázky *Adresy* a udržování integrity dat ve formuláři. Hlavním středobodem je rozlišit, kdy zadaná data společně definují specifické adresní místo, nebo kdy jsou data manuálně plněná uživatelem. V ideálním případě by uživatel pro plnění formuláře vždy používal mapovou komponentu, ale pro uživatelskou použitelnost je nutné zachovat možnost manuálního vyplnění, například pro pokročilé uživatele.

S touto komponentou souvisí i zamykání formulářových polí při vybrání adresního místa z mapy. Jsou to totiž právě pole *Adresy*, která se uzamykají před dalšími změnami. SForms podporuje zamykání polí. Pokud se přidá do JSON-LD objektu ke klíči **"http://onto.fel.cvut.cz/ontologies/form-layout/has-layout-class"** hodnota `"disabled"`, SForms tuto hodnotu propaguje až k nízkoúrovňové React Bootstrap inputové komponentě a nastaví ji na `disabled`. Toho si potom ještě na nižší úrovni všimne prohlížeč a pole zešediví a nepůjde modifikovat.

Hodnota `has-layout-class` pak bude pro většinu polí adresy následující:

```
"has-layout-class" : ["text", "disabled"]
```

Po vynulování zvoleného adresního místa se samozřejmě hodnota `disabled` odebere a pole jsou znovu klasicky modifikovatelná.

12.8 Text adresy - fulltextové vyhledávání

Komponenta `AddressTextComponent` realizuje našeptávání nad textem adresy. Toto našeptávání je fulltextové - nad celým vstupním řetězcem. Datovou strukturou, se kterou komponenta pracuje je otázka *Text adresy* patřící pod otázku *Adresa*. Zde vyvstává mírný problém. Jedné struktuře *Umístění* náleží dle OFN jedna *Adresa*. Nicméně *Adresu* procesuje komponenta `AddressComponent`. Dochází tedy ke kolizi, kdy dvě komponenty pracují nad stejnými daty. To zvyšuje komplexnost a snižuje čitelnost. Po rozmyšlení jsem se touto cestou vydal. Přístup byl následující:

1. `AddressTextComponent` vykreslí nové formulářové pole pod mapou.

   a. Pokud chce uživatel vyplnit text adresy přímo, použije pole v `AddressComponent`.

2. Hodnota v obou komponentách bude identická.

Tento přístup zajistí, že konzistence dat bude zachována. Také stále umožní uživatelům manuálně zasáhnout do hodnoty pole adresy, ale zároveň zde bude možnost poskytnou asistenci našeptáváním. Řeší to tedy problém sdílených dat, ale může to být složitější k použití. Východisko poskytne až výstup z uživatelského testování komponent.

Funkcionalitu našeptání poskytuje aplikace RUIAN-search. V tomto odstavci shrnu jen implementaci na frontendové části.

RUIAN-search poskytuje aplikační rozhraní pro použití externími systémy. Pro jeho konzumaci používám knihovnu **axios**. Axios je HTTP klient založený na Promise pro node.js a prohlížeče.[Sar23]

V aplikaci volám po zadání 4 a více znaků HTTP GET dotaz na endpoint aplikace RUIAN-search `"/api/suggestions/suggest"` s parametrem `searchString=[uživatelům vstup]`. V praxi je pak dotaz volán například takto: `"/api/suggestions/suggest?searchString=Technicka"`.

12.9 Nasazování

Pro automatizaci procesu nasazování a zjednodušení vystavení náhledu aplikace formou Storybook[3] jsem využil zabudovanou funkcionalitu v GitHubu. Po propojení repozitáře se službou Netlify[4] lze nastavit, že po každém pushnutí změn do určité větve se na prostředí Netlify webová aplikace zbuildí a spustí. U mě se toto dělo po každém pushnutí do větve *master*. Tento typ automatického nasazování přináší mnoho výhod. Netlify sám správce repozitáře informuje, pokud se build nepodaří. Dále toto nasazení zprostředkovává náhled na produkční verzi aplikace tak, že je možné nové změny hned vidět ve scénářích Storybook.

[3] https://storybook.js.org
[4] https://www.netlify.com

Kapitola 13

Implementace fulltextového našeptávání

V této kapitole se zaměřím nejenom na implementačních detaily, ale i na nutné změny v konfiguracích. Svým způsobem bude tato kapitola charakteristická tím, že pouze rozšiřuji již vytvořenou a funkční aplikaci. Můj fork původního řešení je udržován v GitHub repozitáři[1]. Všechny mnou provedené změny jsou zřetelné z porovnání původního repozitáře oproti mému forku[2].

13.1 Rozdělení implementace

Postup implementace je rozdělen na dvě části. V první části musím upravit indexovací pravidla a konfiguraci vyhledávací open-source platformě Apache Solr. V části druhé musím rozšířit API rozhraní Java aplikace, která je postavená na frameworku Spring boot. Po finalizaci implementace je nutné služby správně nastavit tak, aby komunikovali mezi sebou a aby Java aplikace byla schopná poskytovat data externím službám - primárně pro React komponentu `AddressTextComponent` (sekce 12.8), která zastřešuje logiku našeptávání textu adresy.

13.2 Apache Solr

Aby služba Apache Solr uměla fulltextově vyhledávat nad textem adresy, je potřeba nahrávaná data přetransformovat tak, aby měla služba k dispozici správně strukturovanou adresu v csv souborech z RÚIAN. Tato transformace probíhá sekvenčně pomocí Java utility vyvinuté v rámci RUIAN-search, která se nazývá **csvmodifier**. Tato utilita prochází stažené csv soubory a modifikuje jejich strukturu. Je potřeba přidat sloupec, který bude obsahovat text adresy. *csvmodifier* zpracuje originální data, která poskytuje RÚIAN a přidá sloupec, ve kterém složí text adresy dle zákona č. 111/2009 Sb. Modifikace probíhá takto:

[1] https://github.com/VojtechLunak/ruian-search

[2] https://github.com/letomas/RUIAN-search/compare/master...VojtechLunak:ruian-search:master

```
String textAddress = "";
//data[10] reprezentuje ulici
//data[8] reprezentuje nazev casti obce
//identification reprezentuje cislo popisne/evidencni
if (!data[10].isEmpty())
    textAddress += data[10] + " " + identification;
else
    textAddress += data[8] + " " + identification;

//data[15] reprezentuje cislo orientacni
textAddress += ", " + data[15] + " ";
//data[6] reprezentuje nazev obvodu Prahy
//data[2] reprezentuje nazev obce
if (!data[6].isEmpty())
    textAddress += data[6];
else
    textAddress += data[2];

//csvWriter prida na konec souboru dalsi sloupce
csvWriter.write(row + ";"
        + identification + ";"
        + transformedCoordinates + ";"
        + textAddress + "\n");
```

Listing 13.1: Ukázka modifikace csv souborů

Tímto se csv soubory modifikují a do Apache Solr se nahrají v dalším kroku s přidanými sloupci.

13.2.1 Konfigurace indexace a dotazování

Aby služba Apache Solr věděla, s jakými daty pracuje, je nutné do schématu, který definuje interní strukturu dat, přidat pravidla, s jakými má daná data zpracovat a pravidla, která má použit při dotazování na data.

Do souboru schématu *managed-schema* jsem přes webové rozhraní Apache Solr přidal pole text adresy a pravidlo (typ) pro text adresy:

```
<field name="Text_adresy" type="text_adresy"
    uninvertible="true" indexed="true" stored="true"/>

<fieldType name="text_adresy" class="solr.TextField">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.ASCIIFoldingFilterFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
    </analyzer>
</fieldType>
```

Listing 13.2: Nastavení typu a pravidel pro Text adresy

Pravidlo platí jak pro indexaci (nahrávání dat), tak i pro dotazování. Apache Lucene, vyhledávací engine pod Apache Solr, data přečte po jednotlivých tokenech (slovech), odstraní diakritiku a převede na malá písmena. Takto jsou data uložená. Při dotazu se s parametrem textu adresy nakládá identicky. Například pro hodnotu dotazu **Technická 6, Praha Dejvice** si Apache Solr převede text do struktury po tokenech: **technicka 6 praha dejvice**. A s

touto hodnotou pak hledá shodu s naindexovanými daty a vrací nejpodobnější text adresy. Úroveň podobnosti už řeší interně Apache Lucene.

Po ozkoušení jsem došel k poznatku, že pokud použiji pokročilé zpracování dotazu přes takzvaný eDismax Query Parser[3], který bere v úvahu i různé ocenění vstupních tokenů, tak jsou výsledky přesnější, než při použiti výchozího parsování dotazu. To i přesto, že já manuálně žádné ohodnocení tokenům nepřiřazuji. Toto chování bude chtít blíže prozkoumat, ale to už vybočuje ze zadání této práce.

13.3 Java aplikace

Hlavním rozšířením Spring boot aplikace je přidání nového kontroleru pro zpracování requestů ohledně fulltextového našeptávání. S tím souvisí i přidání nové našeptávací logiky. I když aplikace již v sobě našeptávání obsahuje, nejedná se o fulltextové vyhledávání nad kompletním textem adresy, ale pouze o našeptávání nad jednotlivými částmi textu adresy, viz podsekce 7.4.6 RUIAN-search.

13.3.1 SuggestionController, AddressService a AddressCustomRepository

Do `SuggestionController` jsem přidal metodu zpracovávající HTTP GET dotaz s parametrem na endpoint **/api/suggestions/suggest**. Parametrem je textový řetězec od uživatele, pro který chce našeptat nejpodobnější hodnoty.

Daná hodnota je propagována přes `AddressService` až do **AddressCustomRespository**. To je třída anotovaná `@Repository`, a je konfigurovaná tak, aby komunikovala přímo se službou Apache Solr. Pro implementaci eDismax dotazu jsem musel rozšířit základní springovské `SimpleQuery`. Tomuto dotazu nastavuji atributy a následně posílám do Apache Solr.

Dotaz je strukturován takto:

```
1 EdisMaxQuery query = new EdisMaxQuery();
2 //pridavam wildcard k retezci dotazu
3 query.addCriteria(new SimpleStringCriteria(searchString+"*"));
4 //chci vratit pouze kod adresniho mista a text
5 query.addProjectionOnFields("K_d_ADM", "Text_adresy");
6 //chci pouzit edismax parser
7 query.setDefType("edismax");
8 //hledam hodnotu Text_adresy
9 query.addQueryField("Text_adresy")
```

Listing 13.3: Ukázka tvorby dotazu do Apache Solr

13.4 Dockerizace

Úprava souboru docker-compose je nutná, protože v původní variantě nestačilo služby pouze spustit, ale uživatel musel ještě použít další podpůrné skripty

[3] https://solr.apache.org/guide/8_7/the-extended-dismax-query-parser.html

pro uvedené aplikace RUIAN-search do funkčního stavu s aktuálními daty.

Můj zásah byl nutný v image Apache Solr. Zde jsem musel přidat logiku, aby se při spuštění image kontrolovala dostupnost a aktuálnost dat dle postupu v návrhu - viz podsekce 11.2.3.

Pro kontrolu existence dokumentu (jádra) ve službe Apache Solr používám výchozí zabudouvanou kontrolu formou skriptů Apache Solr. Hlavní přídavek se týkal kontroly aktuálnosti dat. K tomuto problému jsem přistoupil tak, že po stažení datové sady z RÚIAN se nebude mazat zip soubor s definicemi, ale zůstane v image uložený a podle jeho existence se bude rozhodovat, zdali je dostupný novější soubor, nebo zdali je vůbec nějaký archiv v image k dispozici. Název zip souborů datových sad RÚIAN totiž obsahuje datum vydání v názvu. Díky tomu jsem schopen z názvu vyčíst, ze kdy datová sada pochází, a lze to brát jako opěrný bod při rozhodování o aktuálnosti souboru.

```
#!/bin/bash

csv_data_process() {
  if find "$newDir" -name "*.zip"; then
    data_created=$(find "$newDir"/*.zip | head -1 | grep -Po '[0-9]{8}(?=_)' )
    url=$(wget -q -O - https://nahlizenidokn.cuzk.cz/StahniAdresniMistaRUIAN.aspx | grep 'id="ctl00_bodyPlaceHolder_linkCR"' | sed -r 's/^.+href="([^"]+)".+$/\1/')
    csv_archive=${url##*/}
    last_update=$(echo "$csv_archive" | grep -Po '[0-9]{8}(?=_)')
    if [ "$data_created" != "$last_update" ]; then
      echo "Newer data available. Updating solr..."
      rm -rf "$newDir"/*.zip
      /opt/ruian-solr-scripts/update.sh
    else
      echo "Data are present and up-to-date."
    fi
  else
    echo "No data found. Updating solr..."
    /opt/ruian-solr-scripts/update.sh
  fi
}

echo "Starting initial update"
dir="/opt/solr-8.3.1/data/"
newDir="/tmp/ruian-update/"

# downloads CSV files from ruian if they are not downloaded yet

if [ -d "$newDir" ]; then
  csv_data_process
else
  /opt/ruian-solr-scripts/update.sh
fi
echo "Initial update check done"
```

Listing 13.4: Skript pro kontrolu aktuálnosti dat

Image RUIAN-solr je publikovaný jako Github Package[4].

Takto upravená služba běží společně se Spring boot Java aplikací na serveru výzkumné skupiny znalostních a softwarových systému. Vstupní bod Java aplikace je na adrese https://kbss.felk.cvut.cz/ruian-search.

[4]https://github.com/VojtechLunak/ruian-search/pkgs/container/ruian-search%2Fruian-solr

Část IV

Evaluace

Kapitola 14
Uživatelské testování

V této kapitole vydefinuji testovací scénáře pro uživatele SForms se zaměřením především na vytvořenou geo komponentu v rámci této práce. Cílem uživatelského testování je ověření, zda komponenta opravdu přináší žádoucí změny - zrychlení vyplnění Turistického cíle, konkrétně pak adresy a zeměpisných souřadnic. Pak zanalyzuji výstupy testování a odvodím, jestli komponenta opravdu zefektivňuje a zpříjemňuje vyplňování formulářů.

14.1 Metodika testování

Cílem testování geo komponenty s uživateli je odhalit nedostatky v implementovaných rozšířeních, nebo naopak potvrdit jejich smysluplnost.

Hlavním kritériem vyplňování je **rychlost**. To bude hlavní metrikou v rámci všech scénářů. Dále se testování zaměřuje na uživatelskou použitelnost, přívětivost. Rychlost budu přímo porovnávat proti vyplňování formuláře v základní verzi SForms.

Aby se testování zaměřovalo pouze na moje změny, bude probíhat v prostředí, kde bude geokomponenta izolovaná - formulář bude vykreslen jen pro objekt *Umístění*. Testovací scénáře se budou zaměřovat na nejčastější použití. Dle nejčastějšího použití byly i prioritizovány požadavky - viz kapitola 8 Specifikace požadavků.

Testování bude probíhat ve webových prohlížečích na počítačích nebo noteboocích. Neberu v potaz hardwarové rozdíly mezi zařízeními uživatelů.

14.2 Evaluační scénáře

Testovacích scénářů je 8. Scénáře jsou pojmenovány dle vzoru:
Scénář <číslo><a/b> - název akce, kterou uživatel provádí

Číslo značí pořadí scénáře. Písmeno *a* znamená, že průchod probíhá s nově implementovanou **s-forms-geo-components** knihovnou. Naopak písmeno *b* znamená, že se průchod odehrává v prostředí základní knihovny **SForms**.

Každý scénář obsahuje tři části:

- **Před zahájením průchodu** - instrukce, na co se má uživatel soustředit,

a jaké akce potřebuje provést před samotným průchodem (například připravit si stopky)

- **Průchod** - samotné kroky průchodu
- **Po průchodu** - sběr zpětné vazby ihned po dokončení průchodu - otázky se liší scénář od scénáře

Uživatel bude postupovat tak, že realizuje jeden scénář a poskytne ihned zpětnou vazbu k danému scénáři. A takto se to opakuje pro každý scénář.

Dokument s kompletními informacemi o scénářích a s kompletní zpětnou vazbou je dostupný v dokumentech google[1]. V textu bakalářské práce představím scénáře a shrnu zpětnou vazbu od uživatelů.

14.2.1 Scénář 1a - zadávání úplné adresy - našeptání

Scénář 1a se zaměřuje na otestování komponenty s fulltextovým našeptáváním. Uživatel zadává text adresy do pole s našeptáváním a vybere žádoucí adresu. Kontroluji, zda se uživatelům pracovalo s mapou intuitivně a jestli jim našeptávání poskytlo očekávané texty adresy.

14.2.2 Scénář 1b - zadávání úplné adresy - bez našeptání - SForms

Scénář 1b simuluje zadávání adresy do formuláře bez našeptávání. Uživatel musí správně zadat části adresy do formuláře. Zde mě hlavně zajímalo, zda je varianta s našeptáváním rychlejší, nebo ne.

14.2.3 Scénář 2a - zadávání zeměpisných souřadnic z mapy

Scénář 2a realizuje výběr souřadnic z mapové komponenty. Abych průchod zobecnil, použil jsem jako polohu, kterou chci zadat bydliště uživatele. Tímto způsobem bude každý uživatel mít "jiný"průchod, ale zároveň se stále testuje stejná funkcionalita.

14.2.4 Scénář 2b - zadávání zeměpisných souřadnic - SForms

Scénář 2b se zaměřuje na zadávání zeměpisných souřadnic bez mapové komponenty. Uživatel se snaží zadat souřadnice svého bydliště, ale SForms mu k tomu žádnou pomocnou mapu nevykreslují. Testuji, jakým způsobem uživatelé budou souřadnice hledat a jestli přítomnost mapové komponenty ve scénáři 2a čas průchodu zkrátí.

[1] https://docs.google.com/document/d/10ArzZdk7d7hwpu4oeQx2Bc12HiR16zTqNoHNd8bqAU/edit?usp=sharing

14.2.5 Scénář 3a - zadávání adresního místa výběrem z mapy

V tomto scénáři se znovu zadává adresa (adresní místo), ale pouze pomocí interakcí s mapou. Cílem je otestovat, zda se mapa chová dle očekávání uživatelů.

14.2.6 Scénář 4a - zadávání aktuální polohy - GPS

Ve scénáři 4a jsem se zaměřil na případ, kdy uživatel musí zadat souřadnice a adresu své aktuální polohy. Proto je podmínkou toho scénáře to, aby se uživatel nacházel v místě s adresou. S mapovou komponentou by se mělo jednat o rychlý a jasný průchod, kdy uživatel povolí využití lokalizace zařízení a mapa se vycentruje na uživatelovu polohu. Zde jsem musel navíc zjistit, na jakém zařízení (počítač, nebo notebook) uživatel průchod prováděl, abych mohl porovnat výsledky.

14.2.7 Scénář 4b - zadávání aktuální polohy - SForms

Scénář 4b simuluje situaci, kdy uživatel chce zadat souřadnice a adresu své aktuální polohy, ale nemá k dispozici vykreslenou mapovou komponentu. Soustředím se především na postup, který uživatel použije ke zjištění souřadnic a rozdíl času splnění průchodu oproti scénáři 4a.

14.2.8 Scénář 5 - komplexní průvod

Ve scénáři 5, který nijak neporovnávám s SForms, testuji, jestli se implementovaná komponenta chová intuitivně i při několika změnách výběru adresního místa i zeměpisných souřadnic.

14.3 Výsledky testování

Testování se zúčastnili čtyři lidé. Všichni byli z výzkumné skupiny znalostních a softwarových systémů a v problematice OFN a sémantických dat se orientují.

Po shromáždění výsledků testování od uživatelů jsem odhalil jeden opakující se jev, který částečně znehodnotil časové porovnání mezi **s-forms-geo-components** a základním **SForms**. Polovina uživatelů využívala ve scénářích s písmenem *b* hodnoty souřadnic a adresy nalezených ze scénářů s písmenem *a*. Například v průchodu scénáře 4a našel tester zeměpisné souřadnice a adresu aktuální polohy, a využil tato data i při průchodu scénáře 4b. To bylo pravděpodobně způsobeno tím, že jsem dostatečně nevymezil způsoby, kterými uživatel může hledat zeměpisné souřadnice a adresu. Ale i tak byly průchody v s-forms-geo-components rychlejší než v SForms až o 30 sekund, protože uživatele nezdržovalo nadbytečné překlikávání mezi okny a kopírování hodnot. U dvou uživatelů, kteří hodnoty pro scénáře v SForms hledali jinde než v předchozích scénářích se potvrdilo, že vyplňování s geokomponentou bylo **rychlejší** až **pětinásobně** - Scénář 4a vs Scénář 4b.

Při testováni narazili dva uživatelé na situaci, kdy jim našeptávací pole vrátilo pro hodnotu **Budějovická 1523/9a** nejpřesnější shodu jako text **Budějovická 340, 40747 Varnsdorf**. Ostatní uživatelé asi zadávali adresu zkráceněji a zvolili správně našeptané pole dříve. Toto nepřesné našeptání je nejspíš způsobeno lomítkem v textu vyhledávané adresy, na které není Apache Solr správně nastaven.

Výstupy z testování jsou shrnuty v odrážkách níže:

- Každý tester souhlasil s tím, že je mapová komponenta intuitivním vylepšením základní knihovny SForms.
- Všechny průchody se podařilo dokončit - aplikace nespadla.
- Našeptávání poskytuje nepřesné výsledky, když vstupní text k našeptání obsahuje lomítko.
- Ikony zvolených souřadnic a adresních míst by mohly být výraznější (1 ze 4 uživ.).

Jeden tester uvedl poznámku k obecné realizaci testování. Ocenil by použití formulářových služeb (například google forms) tak, aby průchody a otázky k nim náležející byly kousek pod sebou.

Kapitola 15

Závěr

Výstupem práce je funkční knihovna React komponent, která rozšiřuje javascriptovou knihovnu SForms. Komponenty umí zobrazit formulář s dynamickou strukturou otázek. Pro formulář se tím poskytují funkcionality našeptávání adresních míst, vyhledání aktuální polohy uživatele, výběr adresního místa z mapy a automatické vyplnění závislých položek formuláře. Pro našeptávání jsem použil dockerizovanou externí aplikaci, která se automaticky inicializuje. Aktualizaci datových sad adresních míst z RÚIAN lze jednoduše provést spuštěním skriptu. V konečném stavu je uživatelské rozhraní v českém jazyce, ale díky externalizaci našeptávácí služby není problém v budoucnu knihovnu napojit na jiný zdroj pro našeptávání. Nasazování knihovny je automatizováno a náhled na nejnovější verzi komponent je k dispozici ve formě Storybook scénářů[1]. Implementovaná knihovna komponent nabízí řešení všech požadavků, které byly metodou MoSCoW prioritizovány M - *must have*, až na požadavek FR1.2 zmíněný v následující sekci Budoucí práce.

Knihovnu jsem podrobil uživatelským testům. Z testování vyplynulo, že je knihovna výrazným vylepšením základního SForms. I přesto nastaly situace, kdy uživatelé nějaké výhrady měli. Ty se týkaly drobných nesrozumitelností v použití komponenty a nedokonalého našeptávání při použití lomítka při vyhledávání.

V průběhu práce vyvstaly různé překážky (složité mapování, změna struktury dat formuláře), ale celkově jsem splnil zadání práce. Důvodem pro nesplnění některých požadavků je jejich nižší priorita a soustředění na prioritnější požadavky.

Výstupem je React knihovna komponent v TypeScriptu udržována v repozitáři[2] na GitHubu.

[1] https://s-forms-geo-components.netlify.app/?path=/story/sforms-geo-component–tourist-destination-with-geolocation

[2] https://github.com/VojtechLunak/s-forms-geo-components

Budoucí práce

Po implementování nejdůležitějších požadavků dle priority jsem v rámci bakalářské práce nesplnil požadavky v seznamu dále. Seznam je seřazen podle důležitosti, kterou jsem odvodil z dopadu absence implementace požadavku vůči finálnímu řešení.

Jedná se o požadavky:

- M - FR1.2 - Okamžité informování uživatele o stavu vyplnění
  - po domluvě s vedoucím práce se tento požadavek bude implementovat přímo v knihovně SForms společně s komplexní validací vstupů
- S - NFR1.5 - Půjde zvolit i anglický jazyk pro uživatelské rozhraní komponent.
- C - FR1.5 - Přepínání mezi souřadnicovými systémy
- S - NFR1.4 - Implementované komponenty budou validovány dle metodiky RAIL.
- S - FR3.1 - Tříhodnotový boolean
- C - FR1.6 - Přepínání mezi satelitním pohledem
- S - FR3.2 - Oprava výběru časového rozpětí

Další prací do budoucna je i optimalizace našeptávání, které se nechovalo v krajních situacích dle očekávání uživatelů při testování.

Přílohy

Příloha A

Literatura

[AC23] s.r.o. AION CS, *Zákon č. 106/1999 sb., zákon o svobodném přístupu k informacím*, https://www.zakonyprolidi.cz/cs/1999-106#p3a-3, 2023, Online; Navštíveno: 25.05.2023.

[Aga23] Volodymyr Agafonkin, *Leaflet - an open-source javascript library for mobile-friendly interactive maps*, https://leafletjs.com, 2023, Online; Navštíveno: 25.05.2023.

[AP19] s.r.o ARCDATA PRAHA, *Vyhledávací služba geocodesoe - popis rozhraní*, https://geoportal.cuzk.cz/Dokumenty/Rozhrani_GeocodeSOE.pdf, 2019, Online; Navštíveno: 25.05.2023.

[BLH+23] Miroslav Blaško, Martin Ledvinka, Vojtěch Holub, Yan Doroshenko, Max Chopart, and Tomáš Klíma, *Sforms*, https://github.com/kbss-cvut/s-forms, 2023, Online; Navštíveno: 25.05.2023.

[Cc22] Paul Le Cam and contributors, *React leaflet*, https://react-leaflet.js.org, 2022, Online; Navštíveno: 25.05.2023.

[Cor23a] Mozilla Corporation, *Client-side form validation*, https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation, 2023, Online; Navštíveno: 25.05.2023.

[Cor23b] ———, *Geolocation api*, https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API, 2023, Online; Navštíveno: 25.05.2023.

[Fou23] The Apache Software Foundation, *Apache lucene*, https://lucene.apache.org, 2023, Online; Navštíveno: 25.05.2023.

[Kl1] Bc. Tomáš Klíma, *Editor sémantických webových formulářů*, http://hdl.handle.net/10467/92722, 2021, Online; Navštíveno: 25.05.2023.

[Kl3] Jakub Klímek, *Propojená data*, https://ofn.gov.cz/propojená-data/draft/#principy-propojených-dat, 2023, Online; Navštíveno: 25.05.2023.

[Le20] Bc. Tomáš Le, *Aplikace usnadňující vyhledání adresních míst v rúian*, http://hdl.handle.net/10467/88341, 2020, Online; Navštíveno: 25.05.2023.

[M21] Ing. Michal Med, RNDr. Jakub Čurda, and Mgr. Veronika Kůsová, *Informace o stahovacích službách pro inspire téma adresy*, Sekce centrální databáze ČÚZK, https://services.cuzk.cz/doc/inspire-ad-download.pdf, 2021, Online; Navštíveno: 25.05.2023.

[Ne3] Martin Nečaský, *Sémantický slovník pojmů*, https://ofn.gov.cz/sémantický-slovník-pojmů/draft/#úvod, 2023, Online; Navštíveno: 25.05.2023.

[Sar23] John Jakob "Jake"Sarjeant, *Axios*, https://axios-http.com/docs/intro, 2023, Online; Navštíveno: 25.05.2023.

[SLK+23] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin, and Niklas Lindström, *Json-ld 1.1*, https://www.w3.org/TR/json-ld11/#the-context, 2023, Online; Navštíveno: 25.05.2023.

[tea23] Chrome DevRel team, *Measure performance with rail model*, https://web.dev/rail/, 2023, Online; Navštíveno: 25.05.2023.

[V20] topografický a kartografický Výzkumný ústav geodetický, *Slovník tk ČÚzk*, http://ww.slovnikcuzk.eu/index.php?jazykova_verze=cz, 2020, Online; Navštíveno: 25.05.2023.

[v3a] Ministerstvo vnitra ČR, *Adresní místo*, https://slovník.gov.cz/prohlížíme/pojem?iri=https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/adresní-místo, 2023, Online; Navštíveno: 25.05.2023.

[v3b] ———, *Otevřené formální normy - adresa*, https://slovník.gov.cz, https://slovník.gov.cz/prohlížíme/pojem?iri=https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/adresa, 2023, Online; Navštíveno: 25.05.2023.

[v3c] ———, *Otevřené formální normy (ofn)*, https://data.gov.cz, https://data.gov.cz/ofn, 2023, Online; Navštíveno: 25.05.2023.

[7] Český statistický úřad, *Základní pojmy*, https://www.czso.cz/csu/rso/zakladni_pojmy, 2017, Online; Navštíveno: 25.05.2023.

Příloha B

Návod na instalaci s-forms-geo-components

1. Nainstalujte program Node.js[1] verze 18.16.0.
2. Přiložený zdrojový kód rozbalte, nebo naklonujte GIT repozitář https://github.com/VojtechLunak/s-forms-geo-components.
3. Přejděte do složky se zdrojovým kódem.
4. Pro ukázku funkční aplikace použijte
   a. příkaz npm run dev.
   b. příkaz docker-compose up - pro spuštění kompletního řešení (geo-komponenta s RUIAN-search). Může trvat až 30 minut.
5. V prohlížeči přejděte na adresu localhost:6006.

[1] https://nodejs.org/en