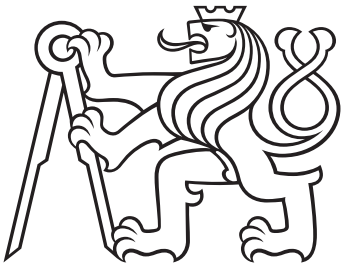
České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů
Obor: Softwarové inženýrství a technologie



Sémantické komponenty pro formuláře

Semantic-based form components

BAKALÁŘSKÁ PRÁCE

Vypracoval: Vojtěch Holub
Vedoucí práce: Mgr. Miroslav Blaško, Ph.D.
Rok: 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Holub** Jméno: **Vojtěch** Osobní číslo: **483550**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Sémantické komponenty pro formuláře

Název bakalářské práce anglicky:

Semantic-based form components

Pokyny pro vypracování:

SForms [1] je javascriptová knihovna pro zobrazování interaktivních formulářů ve frameworku ReactJS [2]. Narozdíl od obdobných knihoven je SForms založena na technologiích Sémantického webu, což umožňuje k otázkam přípojit jejich strojově zpracovatelný význam v podobě ontologií.
Cílem práce je vytvoření sady chytrých komponent, které umožní uživateli efektivněji vyplňovat formuláře v knihovne SForms pro vybrané významy otázek. Chytré komponenty jsou komplementární k více generickým komponentám v SForms. Kompatibilita chytrých komponent s otázkami formuláře bude vyhodnocena automaticky ze struktury formuláře a významu jeho otázek.
Instrukce:
1) seznamte se s sémantickými webovými technologiemi pro reprezentaci (OWL, RDF, JSON-LD) a validaci (SHACL) znalosti
2) prozkoumejte existující frameworky pro reprezentaci formulářů s důrazem na podporu pro popis sémantiky otázek
3) analyzujte požadavky chytrých komponent, případně rozšíření SForms
4) navrhnete a implementujte chytré komponenty včetně nutného rozšíření Sforms
5) otestujte implementovaný komponenty včetně uživatelského testování na alespoň 3 uživatelích

Seznam doporučené literatury:

[1] Ledvinka M., Blaško M., SForms (https://kbss.felk.cvut.cz/web/kbss/s-forms)
[2] Walke, Jordan. "React -- A JavaScript library for building user interfaces."; (2013).
[3] Wood, David. "What's New in RDF 1.1."; W3C Working Group Note (2014).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

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

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

Datum zadání bakalářské práce: **11.02.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

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

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze dne .....

.....
Vojtěch Holub

Poděkování

Děkuji Mgr. Miroslavu Blaškovi, Ph.D. za vedení této práce a za podnětné návrhy, které ji obohatily.

Vojtěch Holub

Název práce:
Sémantické komponenty pro formuláře

Autor: Vojtěch Holub

Studijní program: Softwarové inženýrství a technologie
Obor: Softwarové inženýrství a technologie
Druh práce: Bakalářská práce

Vedoucí práce: Mgr. Miroslav Blaško, Ph.D.

Abstrakt: Tato práce se zabývá návrhem a implementací chytrých znovupoužitelných kompoment pro JavaScriptovou knihovnu SForms, které umožní rychlejší a efektivnější vyplňování výsledných webových formulářů. Jejich zobrazování je vyhodnocováno automaticky na základě významu dat.
Klíčová slova: formuláře, sémantika, ontologie, chytré komponenty

Title:
Semantic-based form components

Author: Vojtěch Holub

Abstract: The thesis deals with design and implementation of smart reusable components for the JavaScript framework SForms, which will allow faster and more effective completion of resulting web forms. Selection and rendering of these components is evaluated based on form structure and semantics of data.
Key words: forms, semantics, ontology, smart components

Obsah

Seznam použitých zkratek

regex	Regulární výraz / Regular Expression
RDF	Resource Description Framework
RDFa	Resource Description Framework in Attributes
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
SHACL	Shapes Constraint Language
WCAG	Web Content Accessibility Guidelines

Úvod

Formuláře ke sběru dat mohou obsahovat desítky až stovky otázek, to může způsobit, že se v něm uživatelé při vyplňování budou ztrácet a zapomenou kontext některých otázek. Velké množství polí se také stává irelevantním po zodpovězení určitých otázek a působí rušivě. Kvůli těmto problémům se pak musí uživatelé vracet k předchozím, již vyplněným odpovědím, aby pochopili, čeho se následující pole ve formuláři týkají. To snižuje rychlost a efektivitu sběru dat od uživatelů a dává prostor pro chybně vyplněné odpovědi.

Technologie sémantického webu rozšiřují současný web a informacím přiděluje význam [1]. Tyto data využívá JavaScriptová knihovna SForms a na jejich základě zobrazuje formuláře. SForms ale poskytují pro uživatele pouze základní generické komponenty, které neumožňují co nejefektivnější vyplňování. Výsledné formuláře jsou členěny na velké množství sekcí a podsekcí, které nejsou uživateli prezentovány přehledně.

Cílem této práce je navrhnout a implementovat sadu chytrých komponent pro existující knihovnu SForms, která uživatelům umožní intuitivnější a efektivnější vyplňování údajů do formuláře a zpřehlední prezentaci otázek ve formuláři. Tyto komponenty budou pracovat se sémantickými daty a automaticky se zobrazovat na základě významu a struktury jednotlivých otázek. Komponenty budou rozšiřovat stávající generické prvky v knihovně SForms.

Technologie sémantického webu jsou představeny v kapitole 1. Kapitola 2 se zabývá srovnáním existujících řešení pro sběr dat s připojeným sémantickým významem. Samotná knihovna SForms je popsaná v kapitole 3.

Aktuální stav zobrazovaných formulářů je popsán v kapitole 4. V této kapitole jsou definovány i z toho vyplývající funkční a nefunkční požadavky pro jednotlivé chytré komponenty.

Návrh komponent zohledňující specifikované požadavky se nachází v kapitole 5, popis implementace vybraných navržených komponent je pak v kapitole 6.

Implementované komponenty byly podrobeny uživatelskému testování, zpětná vazba od uživatelů je popsaná v kapitole 7.

Kapitola 1

Sémantické webové technologie

V této kapitole je popsána myšlenka sémantického webu a některé aktuálně používané technologie, které se ji snaží naplnit.

1.1 Sémantický web

Sémantický web rozšiřuje současný web a informacím přiděluje význam, relace s ostatními informace a pravidla pro nakládání s těmito daty. To usnadňuje automatizované strojové zpracování [1][2].

1.2 RDF

Hlavním stavebním kamenem sémantického webu jsou propojená data [1], což jsou informace, které mezi sebou mají definovány určité vztahy [3].

RDF neboli Resource Description Framework je standardizovaný formát pro výměnu těchto dat. Základní strukturou je množina trojic prvků předmět-predikát-objekt (někdo má něco, někdo je něco, atd.). Jednotlivé prvky trojice můžou být unikátní identifikátory (takzvané IRI) nebo literály. Tato množina tvoří orientovaný RDF graf, kde jsou predikáty hrany, předmět a objekt vrcholy (uzly) [4].

Jedná se o abstraktní formát a pro jeho reprezentaci jsou používány další formáty jako je například Turtle, JSON-LD nebo RDFa [4].

1.2.1 RDF Schema

RDF Schema je ontologický modelovací jazyk, který sémanticky rozšiřuje RDF. Slovník připomíná objektově orientované programovací jazyky, definuje třídy `rdfs:Class`, dědičnost `rdfs:subClassOf`, vlastnosti `rdf:Property`, seznamy `rdf:List` a další [5].

1.3 OWL

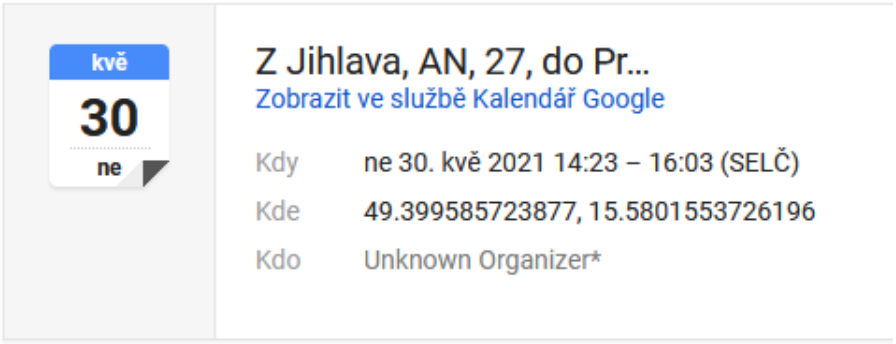
OWL je další z modelovacích jazyků. Narozdíl od RDF Schema je OWL expresivnější a umožňuje vytvářet průniky `owl:intersectionOf` a sjednocení `owl:unionOf`, kardinality `owl:maxCardinality`, disjunktnost `owl:disjointWith` a další. Aktuální verze jazyku je OWL 2 [6].

1.4 Schema.org

Schema.org je komunitní projekt, jehož cílem je spravovat slovník pro anotaci strukturovaných dat na webu.

Za vznikem stojí přední společnosti vyvíjející populární internetové vyhledávače. Slovník za pomoci HTML mikrodat (rozšíření HTML atributů) nebo JSON-LD rozšiřuje existující webové stránky a jednotlivým HTML prvkům přidává význam. Mezi nejčastější typy se řadí `Product` (produkt na internetovém obchodě), `Place` (místo), `Event` (jakákoliv událost) [7].

Vyhledávačům tak usnadňuje agregaci dat (např. průměrné hodnocení produktu). Emailová služba Gmail používá schema.org pro zvýrazňování rezervací letenek, zásilek nebo událostí [8]. Na následujícím obrázku je ukázka zobrazení kalendářové události, která byla formulována v těle emailu.



Obrázek 1.1: Zobrazení události ve službě Gmail

1.5 JSON-LD

Jedním ze způsobu reprezentace struktury RDF je JavaScript Object Notation for Linked Data, který je postavený na široce používaném formátu JSON a je pro lidi jednoduše čitelný bez nutnosti hlubší znalosti sémantických technologií [9].

```
{
  "@context": {
    "name": "http://schema.org/name",
    "description": "http://schema.org/description",
    "geo": "http://schema.org/geo",
    "latitude": {
      "@id": "http://schema.org/latitude",
      "@type": "xsd:float"
    },
    "longitude": {
      "@id": "http://schema.org/longitude",
      "@type": "xsd:float"
    },
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "name": "Prazsky hrad",
  "description": "Historicke sidlo",
  "geo": {
    "latitude": "50.091",
    "longitude": "14.401"
  }
}
```

Úryvek 1.1: Příklad historického sídla, reprezentovaného v JSON-LD

Syntaxe JSON-LD je navržena tak, aby umožňovala jednoduché nasazení na již existující systémy pracující s formátem JSON. Za pomoci objektu `@context` se jednotlivé termíny rozšíří na úplné identifikátory. `@id` označuje unikání identifikátor objektu a `@type` slouží k vytvoření hodnoty s datovým typem [10].

```
{
  "http://schema.org/description": [{"@value": "Historicke sidlo"}],
  "http://schema.org/geo": [
    {
      "http://schema.org/latitude": [
        {
          "@type": "http://www.w3.org/2001/XMLSchema#float",
          "@value": "50.091"
        }
      ],
      "http://schema.org/longitude": [
        {
          "@type": "http://www.w3.org/2001/XMLSchema#float",
          "@value": "14.401"
        }
      ]
    }
  ],
  "http://schema.org/name": [ { "@value": "Prazsky hrad" } ]
}
```

Úryvek 1.2: Příklad historického sídla, reprezentovaného ve formátu JSON-LD s rozšířenými atributy

Termínem `@graph` se označuje graf, který lze interpretovat do RDF trojic. K převodu mezi formátem znázorněným v předchozí ukázce do formátu v ukázce ná-

sledující se používají algoritmy pro zploštění, rozšíření a rámování [11].

```
{
  "@context": { ... },
  "@graph": [
    {
      "@id": "_:b0",
      "description": "Historicke sidlo",
      "geo": {
        "@id": "_:b1"
      },
      "name": "Prazsky hrad"
    },
    {
      "@id": "_:b1",
      "latitude": "50.091",
      "longitude": "14.401"
    }
  ]
}
```

Úryvek 1.3: Příklad historického sídla, reprezentovaného ve formátu JSON-LD, ve tvaru grafu

Předchozí ukázka se pak interpretuje do RDF trojic. RDF graf lze pak zpětně serializovat do formátu JSON-LD.

Předmět	Predikát	Objekt
`_:b0`	http://schema.org/description	Historické sídlo
`_:b0`	http://schema.org/geo	`_:b1`
`_:b0`	http://schema.org/name	Pražský hrad
`_:b1`	http://schema.org/latitude	50.091
`_:b1`	http://schema.org/longitude	14.401

Tabulka 1.1: Ukázka výsledných RDF trojic

1.6 SHACL

Shapes Constraint Language - SHACL je jazyk, který se používá k validaci existujících RDF grafů. SHACL také definuje vlastní slovník, který obsahuje, mimo jiné, termíny pro nastavení pravidel pro možné hodnoty jednotlivých RDF uzlů (mezi ně patří například `sh:equals`, `sh:minLength`, `sh:pattern`) [12].

Jednotlivá pravidla se seskupují do takzvaných tvarů, vůči kterým se validují data. V následující ukázce jsou znázorněna data a tvar k validaci - osoba Jan, kterému je 15 let a pravidlo, které určuje, jestli je dospělý.

```
ex:Jan
  a ex:Person ;
  ex:age 15 .

ex:JeDospelyTvar
  a sh:NodeShape ;
  sh:targetNode ex:Jan ;
  sh:property [
    sh:path ex:age;
    sh:minInclusive 18;
  ] .
```

Úryvek 1.4: Ukázka dat a SHACL tvaru ve formátu Turtle

Proces validace pomocí dat a předem určených tvarů vygeneruje hlášení, ve kterém je určeno, jestli data splňují veškerá validační pravidla [12].

```
[ a sh:ValidationReport ;
  sh:conforms false ;
  sh:result [
    a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:focusNode ex:Jan ;
    sh:resultPath ex:age ;
    sh:value 15 ;
    sh:sourceConstraintComponent sh:MinInclusiveConstraintComponent ;
    sh:sourceShape ex:JeDospelyTvar ;
  ]
] .
```

Úryvek 1.5: Ukázka vygenerovaného hlášení validace SHACL ve formátu Turtle

V příkladu můžeme vidět, že výsledkem je hlášení, které popisuje narušení pravidel - Jan nemá minimálně 18 let.

Kapitola 2

Existující řešení pro sběr sémantických dat na webu

Sémantické technologie umožňují k datům přiřadit jejich význam. Na základě významu dat lze zobrazovat specifické komponenty, které jsou přizpůsobené pro správné zobrazování těchto dat, a zefektivnit tak jejich sběr od uživatele.

Tato kapitola se zabývá popisem již existujících nalezených možností vykreslování webových formulářů na základě sémantických dat. Nástroji SForms je věnována samostatná Kapitola 3.

2.1 RDForm

RDForm[1] je plugin pro framework jQuery, který slouží k editaci RDF[2] dat za pomoci uživatelsky předpřipravených HTML+RDFa[3] šablon. Data samotná se do formuláře vkládají ve formátu JSON-LD[4], ve stejném formátu je také plugin vrací [13].

```
<form
    prefix="foaf http://xmlns.com/foaf/0.1/ rdfs
            http://www.w3.org/2000/01/rdf-schema#"
    base="http://example.com/">
    <legend>Clovek</legend>
    <div typeof="foaf:Person" resource="Person-{rdfs:label}">
        <label>Jmeno</label>
        <input name="rdfs:label" datatype="xsd:string" />
    </div>
</form>
```

Úryvek 2.1: Ukázková šablona RDFa

[1] https://github.com/simeonackermann/RDForm
[2] Resource Description Framework
[3] Resource Description Framework in Attributes
[4] JavaScript Object Notation for Linked Data

```
{
    "@id": "http://example.com/Person-Jan_Novak",
    "@type": [
        "http://xmlns.com/foaf/0.1/Person"
    ],
    "http://www.w3.org/2000/01/rdf-schema#label": [
        {
            "@type": "xsd:string",
            "@value": "Jan Novak"
        }
    ]
}
```

Úryvek 2.2: Ukázka exportovaných dat v JSON-LD

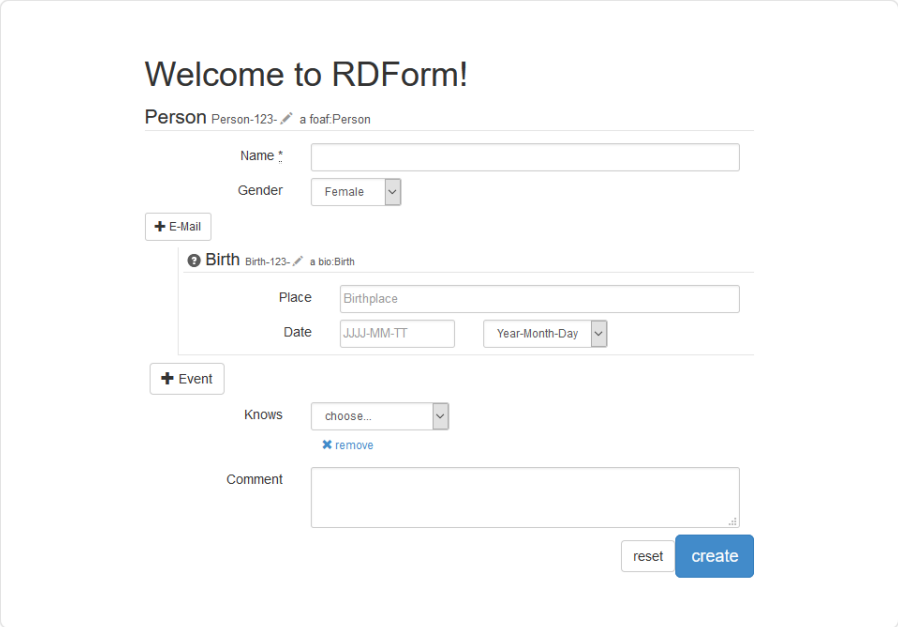
Kromě základních datových typů `string` a `boolean` umí zobrazit navíc pouze `date` a `select`. Datový typ `date` ovšem generuje do formuláře jako klasické textové vstupní pole, který poté validuje tak, aby odpovídalo formátu `YYYY-MM-DD`. Uživatelům nenabízí možnost zobrazení interaktivního kalendáře.

```
<label>Zna se s</label>
<input name="foaf:knows"
       type="resource"
       select
       select-options='{"Person-1":"Pavel Smutny",
                        "Person-2":"Petr Vesely"}'
       external>
```

Úryvek 2.3: Příklad reprezentace pole select

Výběr ze seznamu je ovšem limitující, daný seznam musí být specifikován v šabloně a nelze ho dynamicky zobrazit na základě existujících dat. Ve vývoji je podpora jazyku SHACL[5] pro definici šablon [14]. RDForms podporují různé jazykové varianty, ale musí být zvlášť specifikované v externím javascriptovém objektu, který je předán při inicializaci. Autor samotný nedoporučuje plugin k produkčnímu nasazení. Ukázka možného vygenerovaného formuláře se nachází na adrese `https://simeonackermann.github.io/RDForm/` [13].

[5]Shapes Constraint Language



Obrázek 2.1: Ukázka vygenerovaného formuláře pomocí RDForms

2.2 React JSONSchema Form

React-jsonschema-form[6] je knihovna pro framework React, která umožňuje generování interaktivních formulářů na základě JSONSchema[7], které popisuje sémantiku dat, a vlastní specifikace uiSchema[8], které popisuje, jak se mají jednotlivé datové typy zobrazovat [15].

```
{
  "title": "Priklad formulare",
  "description": "Popis tohoto formulare",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "title": "Jmeno",
      "default": "Jan"
    },
    "age": {
      "type": "number",
      "title": "Vek"
    }
  }
}
```

Úryvek 2.4: Příklad definice dat pro knihovnu react-jsonschema-form

[6] https://github.com/rjsf-team/react-jsonschema-form
[7] https://json-schema.org/
[8] https://react-jsonschema-form.readthedocs.io/en/docs/api-reference/uiSchema/

```
{
  "name": {
    "ui:autofocus": true
  },
  "age": {
    "ui:widget": "updown",
    "ui:title": "Vas vek",
    "ui:description": "Zadejte Vas vek"
  }
}
```

Úryvek 2.5: Příklad nastavení jednotlivých polí formuláře ve formátu uiSchema pro knihovnu react-jsonschema-form

Samotná data se do formuláře importují a exportují ve formátu JSON[9].

```
{
  "name": "Petr"
  "age": 30
}
```

Úryvek 2.6: Příklad výsledných hodnot vyplněných uživatelem

Generované formuláře nabízejí pokročilejší komponenty, na rozdíl od formulářů generovaných pluginem RDForms. Uživatelé mohou datum zadávat pomocí interaktivního kalendáře, formulář má možnost automaticky skrývat některé jeho části na základě vyplnění vybraných položek nebo přímo zobrazovat různé sekce dle uživatelem vybrané položky v select elementu. React-jsonschema-form nepracuje se standardem RDF. K dispozici je i interaktivní editor.[10]

2.3 SHACL Form

Shacl-form[11] je program napsaný v jazyce Python, který generuje HTML formuláře na základě vytvořených tvarů v jazyce SHACL ve formátu Turtle. Výsledná data v požadavku odeslaném uživatelem z formuláře je schopen převést zpět do Turtle formátu. Správně jsou také přiřazovány sémantické HTML atributy pro prvky `xsd:date`, `foaf:mbox` a `foaf:phone`, kde jsou výsledné vygenerované elementy `input` typu `date`, `email`, `tel`, pro které moderní prohlížeče umí zobrazit interaktivní prvky.

Podporovány jsou pouze vybrané definice ze slovníku SHACL `sh:`. Formulářové prvky se generují jenom na základě vlastností `sh:datatype` a `sh:nodeKind`. Shacl-form neobsahuje pokročilé widgety [16].

[9] JavaScript Object Notation

[10] `https://rjsf-team.github.io/react-jsonschema-form/`

[11] `https://github.com/CSIRO-enviro-informatics/shacl-form`

```
:PersonShape1
    a               sh:NodeShape ;
    sh:targetClass  schema:Person ;
    sh:property     [ sh:path          schema:givenName ;
                      sh:nodeKind      sh:Literal ;
                      sh:datatype      xsd:string ;
                      sh:description   "Krestni jmeno osoby" ;
                      sh:name          "Jmeno" ;
                      sh:group         :NameGroup1 ;
                      sh:order         1 ;
                      sh:maxLength     20 ;
                      sh:minCount      1 ;
                      sh:maxCount      1 ;
                      sh:defaultValue "Foo" ; ] ;
```

Úryvek 2.7: Úryvek definovaného tvaru pro shacl-form

```
:15ed5748-1ddc-45cf-b3d1-e7cda472b6e5 a schema:Person ;
    schema:address [ schema:postalCode 13999 ;
            schema:streetAddress "Adress 10"^^xsd:string ] ;
    schema:birthDate "2019-09-01"^^xsd:date ;
    schema:familyName "Novak"^^xsd:string ;
    schema:gender "male" ;
    schema:givenName "Jan"^^xsd:string ;
    foaf:mbox "jan.novak@test.cz" ;
    foaf:phone "12345" .
```

Úryvek 2.8: Úryvek vrácených dat z formuláře shacl-form

2.4 Generování pomocí SHACL+DASH

DASH je slovník, který představuje rozšiřující vlastnosti vhodné pro generování formulářů. Mezi tyto definice spadají možnosti podrobnějšího určení formátu (`dash:singleLine`, `dash:hasPattern`), rozšiřující slovník SHACL [17].

Form Generation using SHACL and DASH[12] je návrh specifikace, který se zabývá spojením tvarů SHACL a DASH k účelu generování formulářů. DASH také definuje vlastní formulářové widgety
(např. `dash:DateTimePickerEditor`, `dash:AutoCompleteEditor`, `dash:ImageViewer`, `dash:TextAreaWithLangEditor`, a další). Tyto widgety pak nahrazují elementární formulářová pole a přináší uživatelům vyšší míru interaktivity.

Widgety se zobrazují na základě detekce datového typu prvku a dalších specifikovaných vlastností tvaru prvku. Pro každý prvek obdrží jednotlivé widgety nezáporné bodové ohodnocení podle vhodnosti zobrazení. Widget s nejvyšším bodovým ohodnocením je pak vybrán k vykreslení do formuláře [18]. V následující tabulce jsou zachycena pravidla pro zobrazení komponenty `dash:TextAreaWithLangEditor`.

[12] `http://datashapes.org/forms.html`

podmínka	bodové ohodnocení
obsahuje vlastnost `dash:singleLine=true`	0
hodnota je typu `xsd:string` a zároveň `dash:singleLine=false`	20
hodnota je typu `xsd:string`	5
`xsd:string` se nachází mezi povolenými typy	2
jinak	0

Tabulka 2.1: Ukázka bodového ohodnocení pro widget dash:TextAreaEditor

2.5 Shrnutí

V předchozích sekcích byla představena některá existující řešení pro generování formulářů.

RDForm poskytuje mechanismus ke svázání předem připravených HTML+RDFa šablon a sémantických dat, knihovna komponenty nezobrazuje přímo ze struktury dat. V případě, že by byla potřeba vytvářet chytré komponenty, musela by se upravit jak samotná knihovna, tak by bylo třeba implementovat samostatný program, který by generoval šablony ze sémantických dat.

Podobně tomu tak je i u knihovny react-jsonschema-form, ta potřebuje upřesnit vzhled samotných otázek pomocí doplňujících dat ve formátu uiSchema. Komponenty se nevybírají ze struktury dat, ale jsou předem specifikovány například pomocí `ui:widget`.

Generování pomocí SHACL+DASH umožňuje narozdíl od SForms komplikovanější prioritizaci zobrazovaných komponent a to pomocí bodového skóre. SForms prvky zobrazuje na základě pravidel s daným pořadím. Pokud struktura otázky odpovídá nějakému pravidlu, tak další pravidla již nejsou zohledňována.

Ani jedna z výše uvedených řešení neumožňuje do dat připojit složitější stromově strukturované definice datových typů a následně je zobrazovat.

Kapitola 3

SForms

SForms[1] je nástroj napsaný ve frameworku React pro zobrazování webových formulářů za pomoci sémantických dat [19]. S těmito daty se pracuje ve formátu JSON-LD. Formulář je možné ukazovat celý najednou nebo ve formě vícekrokového průvodce.

Za pomoci vlastní ontologie je možné specifikovat například hierarchii otázek - vlastnost `has_related_question`, nebo třeba vzhled otázky samotné a to pomocí vlastnosti `has-layout-class`. Pořadí otázek se upřesňuje vlastností `has-preceding-question`. Otázky je možno seřadit i částečně, což umožňuje zachytit sémantiku dat bez vytváření umělých uspořádání.

Ontologie také představuje znaky pro omezení hodnot odpovědí ve formuláři a to například `requires-answer` pro označení povinné odpovědi na otázku, nebo `has-input-mask` k nastavení masky na textové pole. Jsou využívány i běžně používané jednoduché datové typy z XMLSchema[2] a intervaly pro numerické hodnoty `minInclusive` a `maxInclusive`.

V následujícím úryvku je vypsána část příkladu konfiguračního souboru specifikující otázky ve formuláři.

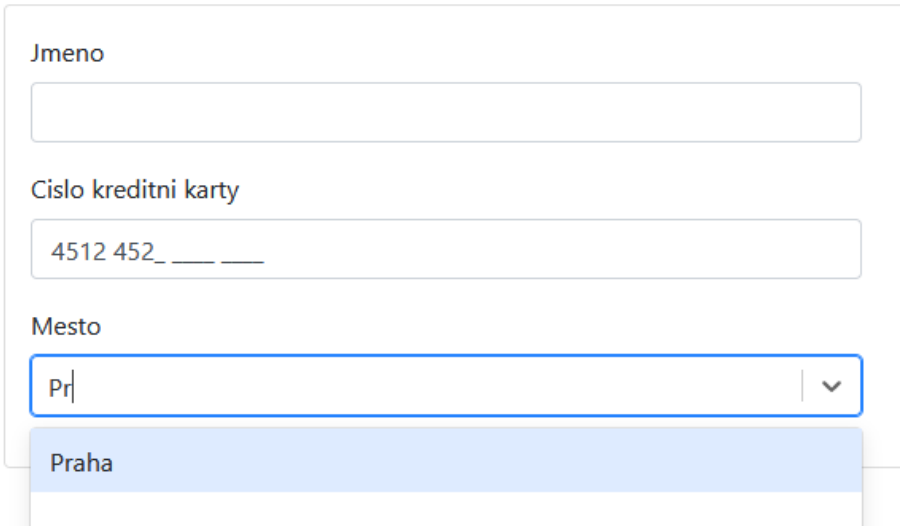
[1] `https://github.com/kbss-cvut/s-forms`

[2] `https://www.w3.org/TR/swbp-xsch-datatypes/`

```
{
  "@id": "cislo-kreditni-karty-2172",
  "@type": "doc:question",
  "has_related_question": [],
  "has-layout-class": "masked-input",
  "has-input-mask": "1111 1111 1111 1111",
  "has-preceding-question": "jmeno-9717",
  "label": "Cislo kreditni karty"
},
{
  "@id": "x:form-root",
  "@type": "doc:question",
  "has_related_question": [
    "jmeno-9717",
    "cislo-kreditni-karty-2172",
    "mesto-8771"
  ],
  "has-layout-class": "form",
  "label": "Root node"
},
{
  "@id": "jmeno-9717",
  "@type": "doc:question",
  "has_related_question": [],
  "has-layout-class": "text",
  "requires-answer": true,
  "label": "Jmeno"
}
```

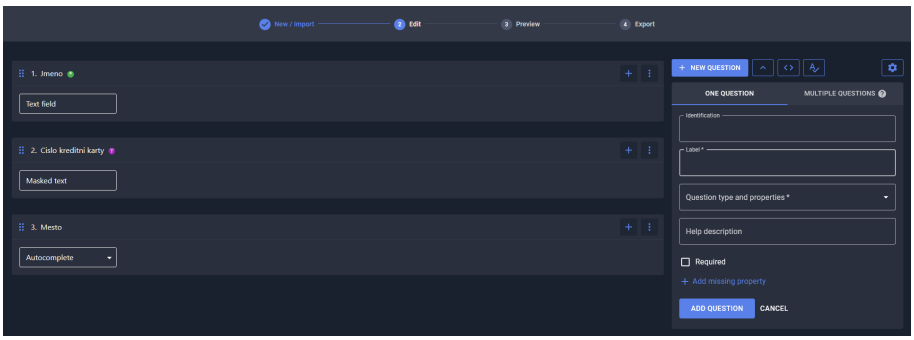
Úryvek 3.1: Úryvek konfiguračního souboru SForms ve formátu JSON-LD

SForms samotné nabízejí několik pokročilejších komponent pro efektivnější vyplňování formuláře jako textové pole s maskou, pro snadnější zadávání hodnot se standardním formátem (telefonní číslo, kreditní karta), či pole s našeptáváním předem určených hodnot. Na následujícím obrázku jsou tyto komponenty zobrazeny. Jak již bylo zmíněno výše, jednotlivé prvky lze hierarchicky seskupovat podle určitého významu například do sekcí.



Obrázek 3.1: Ukázka vygenerovaného formuláře pomocí SForms

S SForms je kompatibilní Semantic Form Editor[3], jenž poskytuje jednoduché uživatelské rozhraní pro vytváření konfiguračních souborů ve formátu JSON-LD.



Obrázek 3.2: Semantic Form Editor

3.1 Otevřené formální normy

Otevřené formální normy (zkráceně OFN) jsou technické specifikace vybraných datových sad, sjednocují tak formát dat pocházejících od různých poskytovatelů otevřených dat. Mezi konkrétní sady patří například definice turistického cíle nebo události [20].

OFN jsou udržovány v rámci projektu Evropské unie KODI[4].

3.2 Správce otevřených dat

Správce otevřených dat (ofn-record-manager) je vyvíjen s cílem vytvořit jednoduché uživatelské rozhraní pro vytváření a editaci dat specifikovaných dle OFN.

Pro sběr dat od uživatele jsou využívány formuláře, které jsou zobrazovány pomocí knihovny SForms. V následující kapitole v sekci 4.1 jsou tyto formuláře popsány.

[3] `https://tomasklima.now.sh/`

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

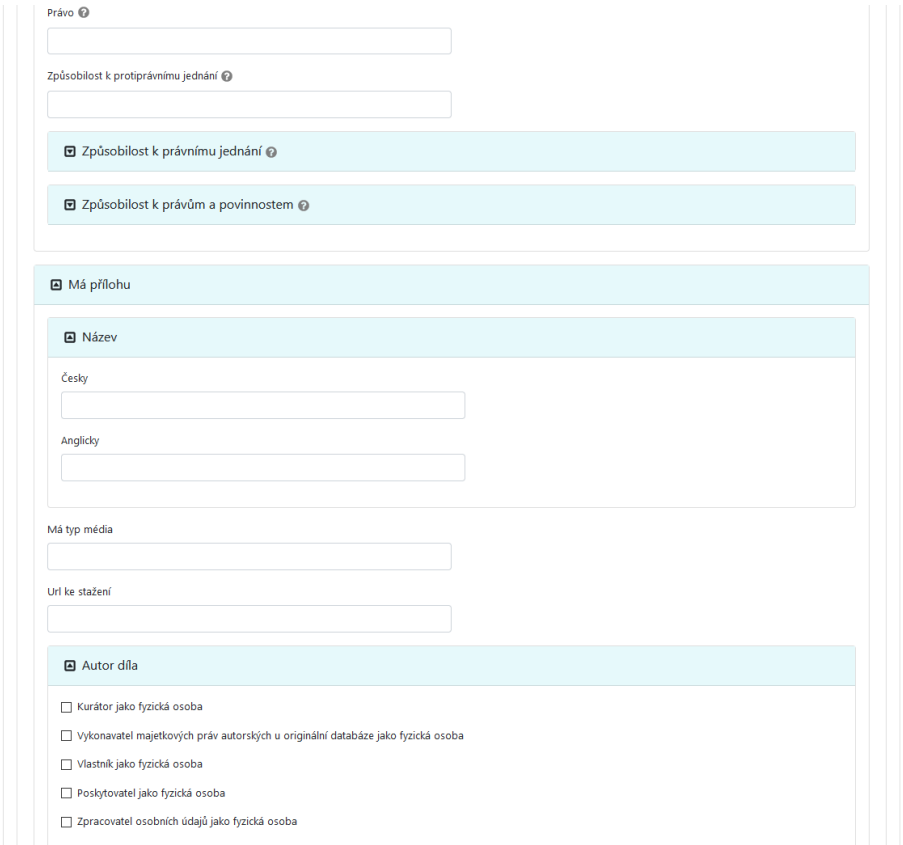
Kapitola 4

Analýza požadavků pro komponenty

Tato kapitola zahrnuje popis aktuálního stavu, problémů a z nich specifikuje funkční i nefunkční požadavky pro návrh chytrých komponent a dalších změn.

4.1 Aktuální stav

Formuláře ve správci otevřených dat mají velký počet otázek, například formulář pro záznam turistického cíle jich má více než 1000. Tyto otázky jsou pak členěny do sekcí a několikaúrovňových podsekcí. Na následujícím obrázku je zachycen stav, kdy jsou některé sekce otevřeny.



Obrázek 4.1: Ukázka otevřených sekcí ve formuláři pro turistický cíl

Na velké množství otázek není ve většině případů nutno odpovídat — nejsou důležité, takové otázky by se neměly uživatelům zobrazovat při každém průchodu formulářem.

Komplikované členění formuláře může způsobovat, že se uživatelé ve formuláři budou ztrácet. Zapomenutí kontextu dané otázky je příčinou návratu k předchozí otázce či sekci, což vede ke snížení rychlosti, jakou je uživatel schopen formulář vyplnit. Také může vést k nepochopení významu otázky a následnému nesprávnému vyplnění odpovědi.

Sekce formuláře sjednocují otázky s podobným významem, další jejich funkcí je možnost skrýt již vyplněné otázky. U vyplněných zavřených sekcí ale nelze na první pohled zjistit stav podotázek. Nelze také určit o čem odpovědi v sekci jednají.

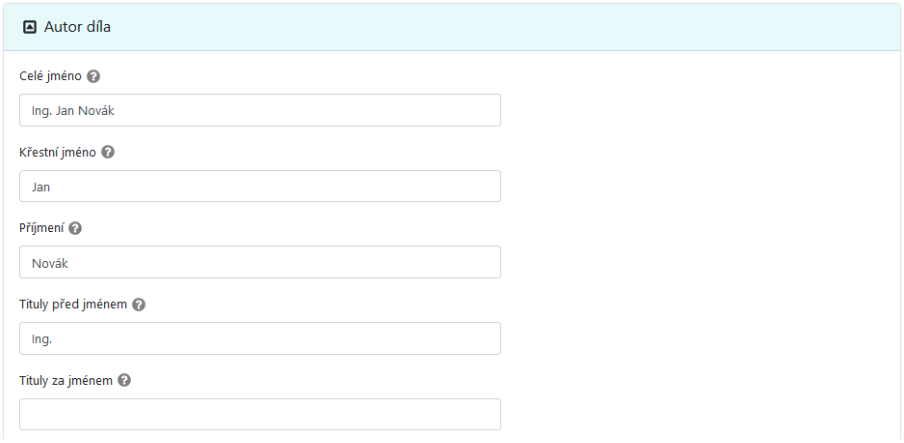
Velké množství sekcí je definováno ve tvaru „Má + předmět", obsahující podotázky upřesňující daný předmět, pokud ale entita předmět neobsahuje, podotázky se stávají nerelevantními.

Jako příklad si uvedeme sekci „Má přílohu", obsahující podotázky „Název" a „URL ke stažení". V případě, že subjekt nemá přílohu, jsou podotázky „Název" a „URL ke stažení" zbytečné a uživateli by se neměli zobrazovat k vyplnění. V opačném případě po vyplnění přílohy bude chtít uživatel sekci zavřít, ze zavřené sekce už ale nejde zjistit o jakou přílohu se jedná, to pak nutí uživatele sekci znovu otevřít, pokud by chtěl některou odpověď zpětně upravit.

Na následujícím obrázku je ukázka těchto zavřených sekcí.

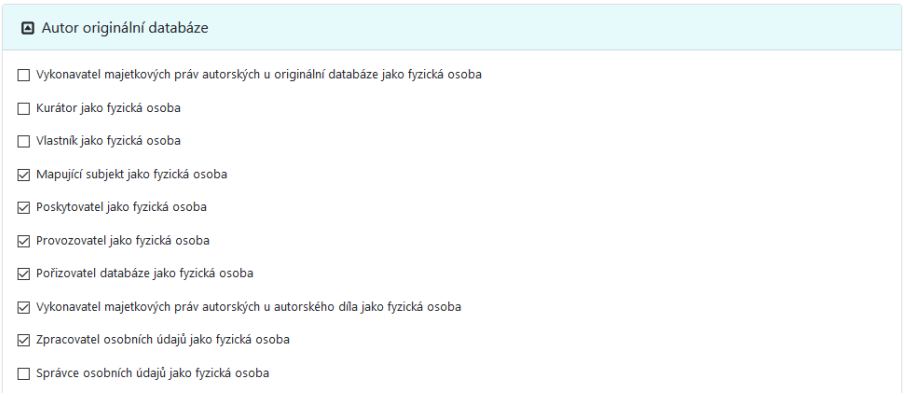
Obrázek 4.2: Ukázka zavřených sekcí ve formuláři pro turistický cíl

Formuláře obsahují množství redundantních otázek, které by mělo stačit vyplnit jenom jednou a tím urychlit sběr dat. Příkladem je otázka „Celé jméno" a s ní související „Jméno", „Příjmení" a další. Poté, co uživatel vyplní „Celé jméno", musí znovu jednotlivě vyplňovat jednotlivě položky celého jména. To platí i obráceně, jestli uživatel odpoví zvlášť na „Jméno" a na „Příjmení", je nucen znovu doplnit „Celé jméno". Tento příklad je ukázán na dalším obrázku.



Obrázek 4.3: Ukázka otázek na jméno ve formuláři pro turistický cíl

Otázky na typ jsou aktuálně řešeny seznamem zaškrtávacích polí, to ale není ideální, protože takovýto seznam může obsahovat velký počet položek, ve kterých se není snadné orientovat. Dalším problémem je nemožnost omezit maximální počet zvolených typů. Některé typy se také mohou navzájem vylučovat a při označení jednoho typu by nemělo být možné vybrat jiné disjunktní typy. Entity tvoří v datech stromovou strukturu, kterou formuláře aktuálně nedokáží uživateli prezentovat. Problém je ukázán v následujícím obrázku.



Obrázek 4.4: Ukázka otázek na typ ve formuláři pro turistický cíl

Zobrazení některých sekcí a otázek ve formuláři je podmíněno kompletním vyplněním určité sekce nebo specifickou odpovědí na předchozí otázku. Samotné zobrazení není doprovázeno animací ani není nikterak zvýrazněno. Uživatel může nově přidané otázky a sekce snadno přehlédnout, což vede k neúplnosti sbíraných dat.

Ve formuláři se také nachází otázky na číselnou hodnotu, doprovázenou zvlášť otázkou na jednotku, či měnu uvedené hodnoty (nejčastějším případem se jedná o peněžní částku s měnou). V rámci zjednodušení čitelnosti je lepší tyto dvě otázky reprezentovat uživateli jako jednu, jelikož spolu úzce souvisí. Prakticky by se otázky zobrazovaly na stejném řádku.

Šířka některých textových a numerických polí neodpovídá délce předpokládané odpovědi. Příkladem může být výše zmíněná peněžní částka.

4.2 Požadavky

Na základě aktuálního stavu popsaného v předchozí sekci byl vytvořen seznam požadavků. Samotné požadavky jsou rozděleny na funkční a nefunkční. Požadavky označené hvězdičkou (*) nebyly v rámci této práce splněny nebo byly splněny jenom částečně.

4.2.1 Funkční požadavky

FR1 Komponenty umožňují uživateli pohyb ve formuláři za pomoci klávesnice.

FR2 Komponenty umožňují snížit množství najednou zobrazovaných otázek.

FR3 Komponenty usnadňují uživateli rychlou orientaci ve formuláři.

FR4 Komponenty zamezí uživateli zobrazovat nerelevantní otázky.

FR5 Komponenty umožňují rozlišit mezi důležitými a méně důležitými otázkami.

FR6 Komponenty snižují množství redundantních otázek.

FR7 Komponenty umožňují zobrazovat hierarchické typy.

FR8 Komponenty podporují reprezentaci hierarchických typů vyjádřených pomocí relace `skos:broader` ze slovníku SKOS a `owl:disjointWith` ze slovníku OWL.

FR9 Komponenty umožňují zobrazit dvě otázky vedle sebe.

FR10 Komponenty umožňují konfigurovat šířku zobrazovaného textového pole na základě předpokládané délku vstupu.

FR11 Komponenty umožňují zvýraznit nově zobrazené otázky a sekce.

FR12 Komponenty umožňují jednoduchou integraci meta otázky.

Meta otázky zachycují aktuální stav formuláře a uživatelské předvolby.

FR12:1 Komponenty umožňují integraci meta otázky zobrazit více.

FR12:2* Komponenty umožňují integraci indikátoru dokončení sekce.

Tento požadavek nebyl splněn.

FR13 Knihovna umožňuje jednoduchou prioritizaci zobrazování jednotlivých komponent dle struktury dat.

FR14 Knihovna umožňuje vypínání jednotlivých komponent, v případě, že není vyžadováno jejich zobrazení.

4.2.2 Nefunkční požadavky

NFR1 Komponenty jsou funkční v aktuálních verzích prohlížečů Microsoft Edge, Mozilla Firefox a Google Chrome.

NFR2 Komponenty se správně zobrazují na obrazovkách s šířkou minimálně 1200 px.

NFR3 Komponenty jsou otestovány v uživatelském testování alespoň 3 uživateli.

NFR4 SForms jsou v případě dalších úprav snadno rozšiřitelné o další komponenty.

NFR5 Komponenty jsou znovupoužitelné ve stejném formuláři.

NFR6 Potřeby komponent nenarušují správnost sémantiky dat formuláře.

NFR7 Vypnutí komponent nenaruší funkčnost formuláře.

NFR8 Komponenty se zobrazují automaticky na základě předem definovaných pravidel podle aktuální struktury dat formuláře, datových typů odpovědí a dalších vlastností otázek.

NFR9 Komponenty pouze rozšiřují existující strukturu dat formuláře.

NFR10* Doba vykreslování změn ve formuláři je menší než 100 ms.

Tento požadavek byl splněn jenom částečně. Doba vykreslování změn ve formuláři v SForms roste s počtem zobrazovaných otázek, za to mohou určitá rozhodnutí při vývoji knihovny SForms. V případě většího množství zobrazených otázek doba překreslení může přesahovat 100 ms. Toto částečně adresuje FR2, který snižuje množství najednou zobrazených otázek. Odezva do 100 ms dává uživatelům pocit, že se jimi vyvolaná akce provedla instantně [21].

Kapitola 5

Návrh komponent

Tato kapitola se věnuje návrhu jednotlivých komponent a rozšíření pro SForms na základě předchozí analýzy, při které byly stanoveny funkční i nefunkční požadavky.

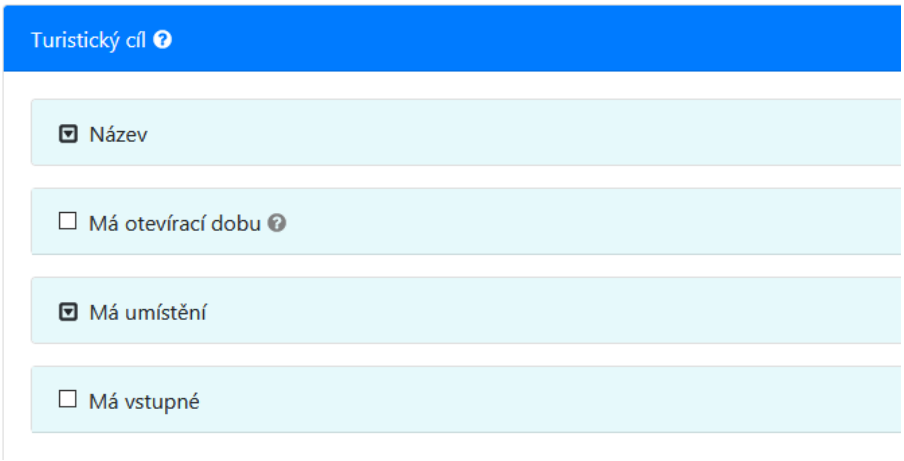
Návrh komponent se může odkazovat na specifikované požadavky, takový případ je znázorněn v závorce — například (FR1).

Pro samotnou implementaci je vyhrazena následující Kapitola 5.

5.1 Sekce s odpovědí

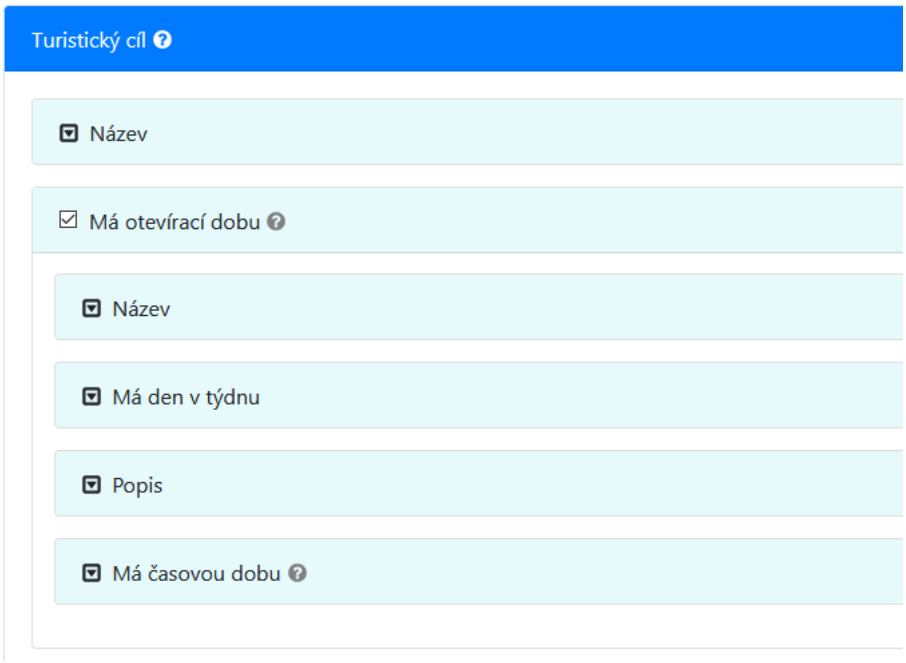
Sekce s odpovědí rozšiřuje funkcionalitu existujících sekcích v SForms o zaškrtávací pole.

Jako příklad si uvedeme Turistický cíl, který může mít přiřazenou otevírací dobu. V aktuálním stavu by byla zobrazena klasická sekce „Má otevírací dobu" se stejnojmennou podotázkou (zaškrtávacím polem), jež by určovala přítomnost otevírací doby. Komponenta umožňuje toto zaškrtávací pole zobrazit přímo v záhlaví sekce.



Obrázek 5.1: Sekce „Má otevírací dobu" s možností odpovědi

Sekce s odpovědí narozdíl od klasické sekce nejde otevřít bez zaškrtnutí odpovědi, protože by podotázky byly nerelevantní a uživateli se nemusí zobrazovat. Po zaškrtnutí odpovědi se sekce automaticky otevře a zobrazí tak případné relevantní podotázky (FR2, FR3, FR4), což je ukázáno na následujícím obrázku.



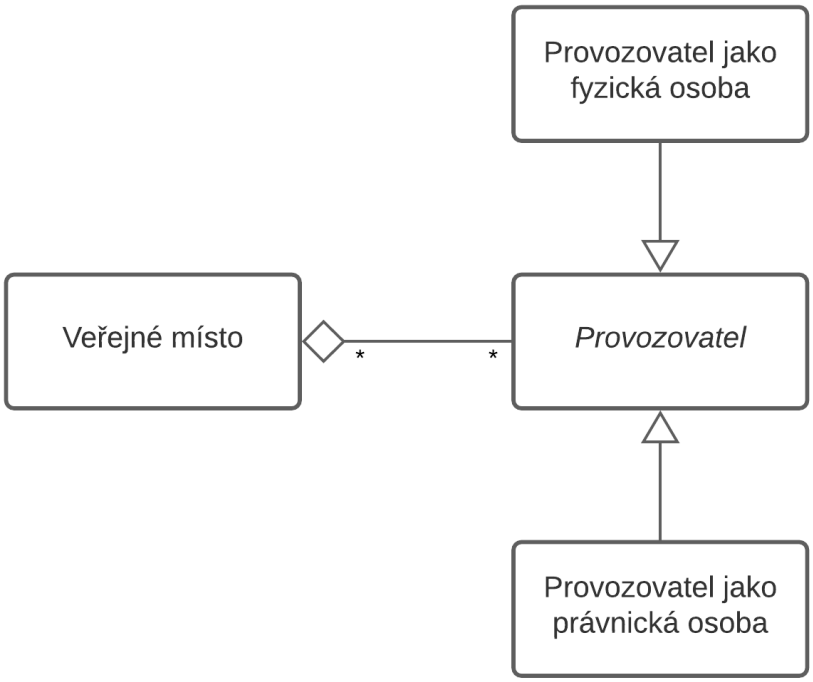
Obrázek 5.2: Sekce „Má otevírací dobu" se zaškrtnutým polem

Pokud je pole zaškrtnuté, sekci lze libovolně zavírat a otevírat jako standardní sekci. Po odškrtnutí se sekce automaticky zavře.

5.2 Filtrovací otázka na základě typu

Komponenta pracuje s otázkou, která obsahuje výběr z více hodnot uspořádaných do stromové struktury. Takovou položku je možné zobrazit několika způsoby - zaškrtávací pole, přepínače, rozbalovací seznam s volbou jedné či více možnosti. Při vykreslování ale záleží jestli se tyto hodnoty navzájem vylučují nebo ne. Komponenta dokáže detekovat, zdali jsou údaje disjunktní, pokud ano, umožní uživateli vybrat pouze jednu položku, v opačném případě nechá uživatele zaškrtnout více možností. Podle složitosti struktury pak zvolí vhodnou podobu pole.

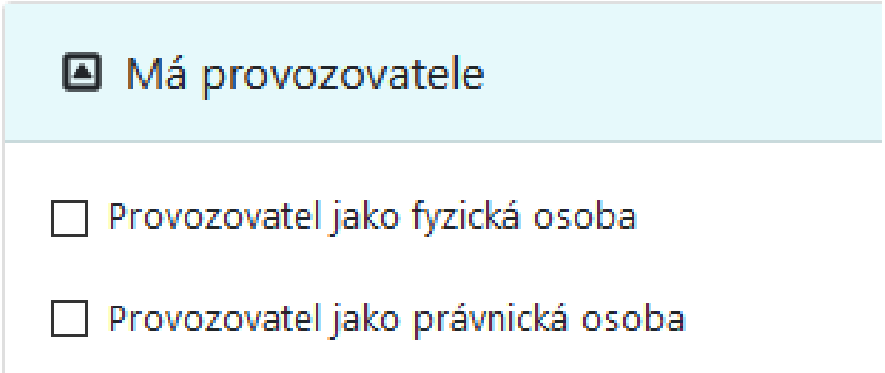
Uvažujme formulář ke sběru dat o veřejných místech, ve kterém můžeme specifikovat provozovatele. Provozovatelem je buď právnická osoba - firma, nebo osoba fyzická. Tyto dvě entity nemají žádné společné otázky. Fyzická osoba má jméno, příjmení a datum narození, zatímco u firmy se budeme ptát na právní formu a sídlo. Těžko budeme u právnické osoby vyplňovat datum narození.



Obrázek 5.3: Diagram veřejného místa s provozovatelem

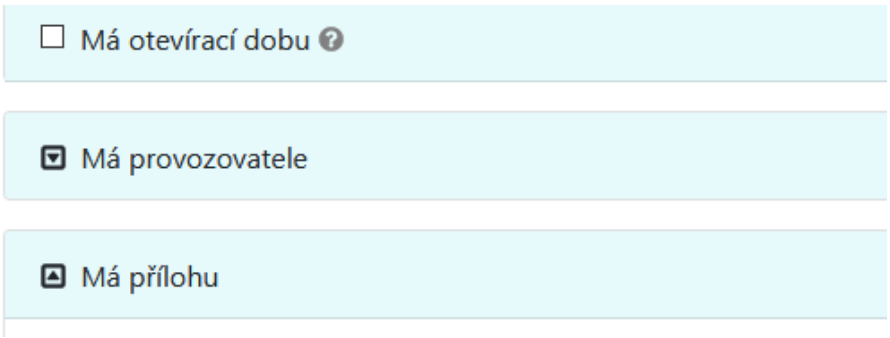
V naivním přístupu bychom mohli zobrazit všechna pole pod sebou. Takový formulář by ale byl velice nepřehledný a pro uživatele by nedával smysl. Kvůli tomu přidáme otázku, která se nejdříve zeptá na typ provozovatele. Je provozovatel fyzická nebo právnická osoba? Chytrá komponenta automaticky detekuje, že jsou druhy provozovatele disjunktní a nechá uživatele zvolit právě jeden. Na základě tohoto výběru pak lze skrýt nepotřebné otázky, které se netýkají daného typu. Výsledkem je přehlednější formulář s menším počtem otázek.

Aktuálně se otázka na základě typu zobrazuje jako série zaškrtávacích polí, které ale neumožňují kontrolu výlučnosti jednotlivých typů a uživatelům je umožněno zaškrtnout všechny typy. To může vést k nesprávně vyplněným údajům a nepřehlednosti. Uživatel vybere všechny typy a na základě toho se mu zobrazí veškeré možné otázky. Na následujícím obrázku je ukázka těchto polí.



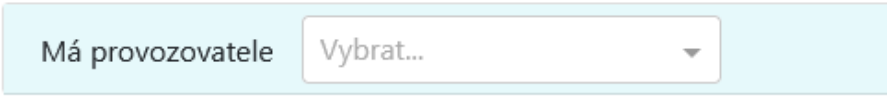
Obrázek 5.4: Aktuální stav otázky na základě typu

Dalším problémem je nejasnost, jestli provozovatel byl vůbec vyplněn. Pro kontrolu stavu je nutné otevřít zavírací sekci a zkontrolovat podotázky. Tuto sekci uživatel pravděpodobně co nejdříve zavřel. Na obrázku níže je tento stav znázorněn a všimněme si, že na první pohled nelze určit, zda má veřejné místo provozovatele.



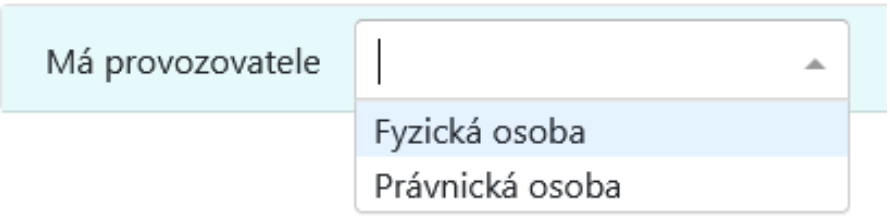
Obrázek 5.5: Aktuální stav otázky na základě typu - nelze na první pohled zjistit typ provozovatele

V rámci řešení byla použita již navržená komponenta sekce s odpovědí, která uživatelům nedovolí zobrazit podotázky, pokud neodpoví na typovou otázku Má provozovatele. Když veřejné místo nebude mít provozovatele, sekce se neotevře a bude jasně viditelné, že není vyplněn typ provozovatele. (FR2, FR3)



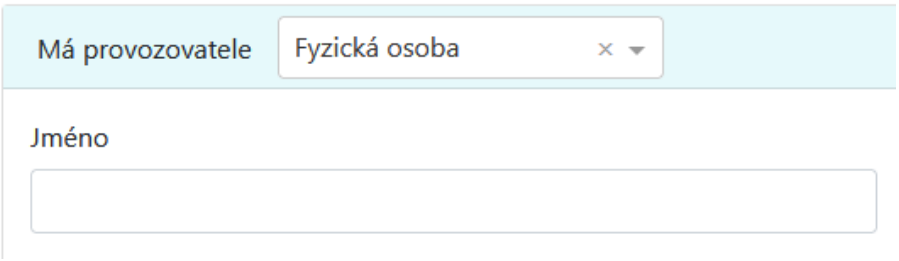
Obrázek 5.6: Řešení otázky na základě typu - lze jednoduše objevit nevyplněný typ provozovatele

Po kliknutí na vybrat se otevře nabídka s možností výběru. Komponenta detekovala, že se jedná o disjunktní typy a dovolí vybrat právě jeden.



Obrázek 5.7: Řešení otázky na základě typu - výběr z nabídky

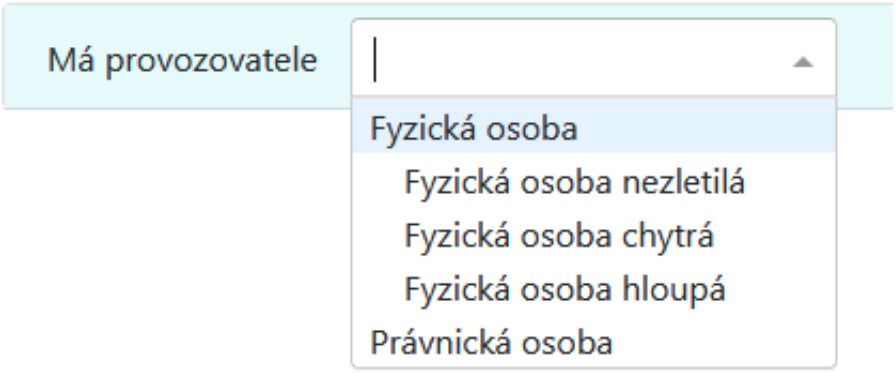
Po výběru se sekce otevře a uživateli se nabídnou pouze relevantní podotázky. V nabídce lze také vyhledávat.



Obrázek 5.8: Řešení otázky na základě typu - otevřená sekce po výběru typu

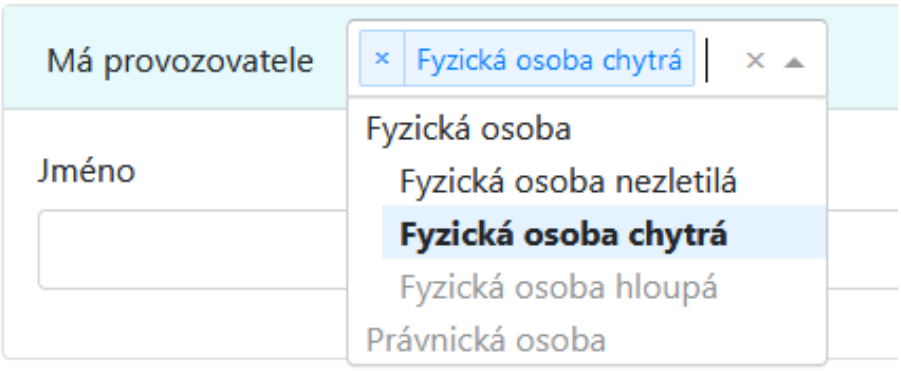
Vybraný typ provozovatele je vždy jasně viditelný a to i po zavření sekce.

V předchozím příkladu se pracovalo pouze se dvěma typy, které byly navíc navzájem disjunktní. Rozšiřme tedy typ Fyzická osoba o další tři vlastnosti - Fyzická osoba nezletilá, Fyzická osoba chytrá a Fyzická osoba hloupá. Vlastnosti chytrá a hloupá se navzájem vylučují (osoba nemůže být chytrá a hloupá zároveň), ovšem už se nevylučují s vlastností osoba nezletilá. Už takto komplikovanou strukturu je v aktuálním stavu prakticky nemožné uživateli reprezentovat. Výběr se v komponentě zobrazí jako strom, což ukazuje následující obrázek.

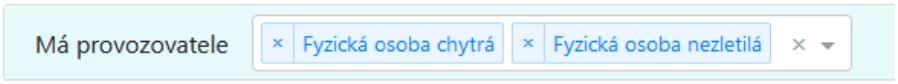


Obrázek 5.9: Řešení otázky na základě typu - zobrazení více vlastností jako strom

Komponenta detekuje, že typy nejsou úplně disjunktní a zobrazí se jako nabídka s možností výběru více položek. V případě, že uživatel zvolí některou z položek, položky, které se s danou vylučují, nepůjdou vybrat, dokud nebude zvolený typ odebrán. (FR7, FR8)



Obrázek 5.10: Řešení otázky na základě typu - vlastnosti, které se vylučují již nelze zvolit



Obrázek 5.11: Řešení otázky na základě typu - vybrané typy jsou viditelné i po zavření sekce

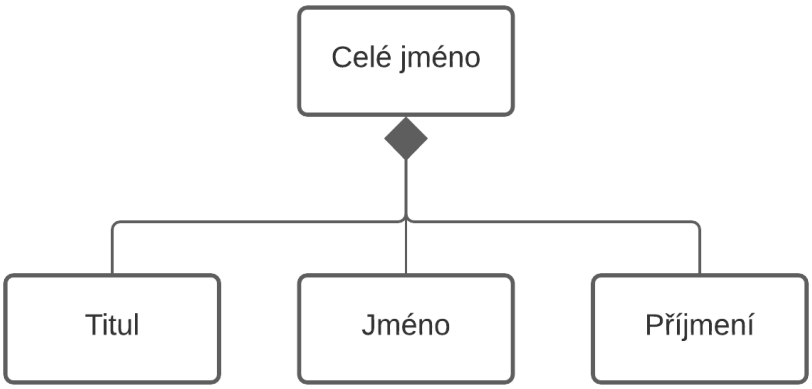
Komponenta na pozadí pracuje s určitým počtem podotázek, do kterých vyplňuje označené typy. Tyto podotázky jsou skryté a pro uživatele neviditelné. V případě vypnutí této komponenty jsou podotázky zobrazovány jako výběry s možností zvolit u každého právě jeden typ. Formulář tak zůstává nadále funkční i bez této komponenty. (NFR7)

5.3 Kompozitní otázka

Kompozitní otázka slučuje více podotázek (textových polí) do jedné a umožňuje automatické a efektivnější doplnění podotázek při zodpovězení sloučené otázky. (FR6)

Může se jednat o libovolný identifikátor, který má jasně definované části. Pokud uživatel rozumí takovému identifikátoru, je pro něho lepší ho vyplnit jako celek, části budou vyplněny automaticky. Když by vyplnění celku bylo sporné, uživatel by mu nerozuměl, je možné vyplnit části jednotlivě, potom bude doplněn celek.

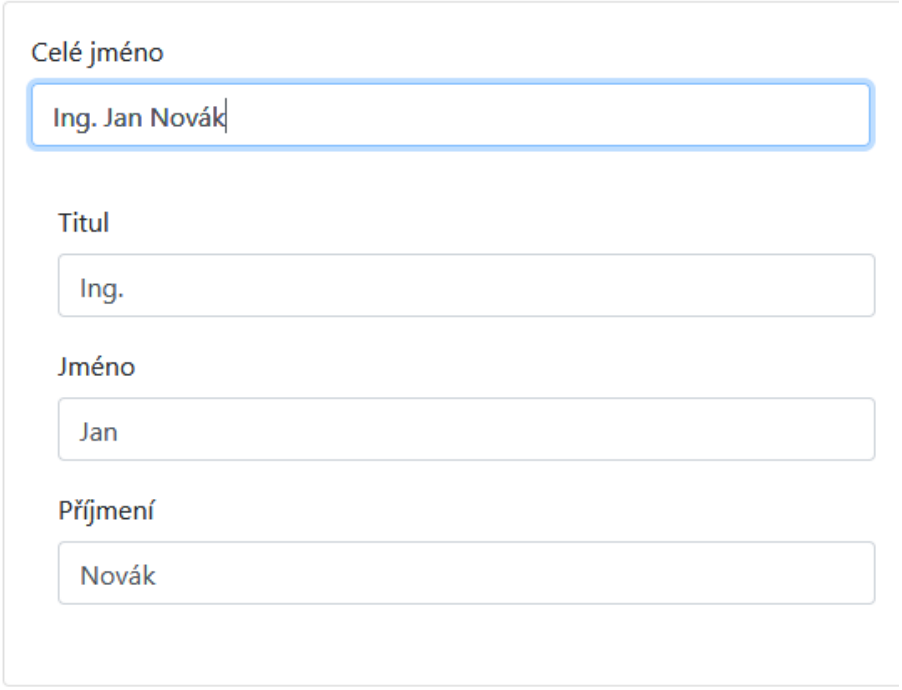
Jako nejlepší příklad lze použít jméno člověka. Celé jméno se skládá z titulu, jména a příjmení, tyto části jsou znázorněny na následujícím diagramu. Uživatel může vyplnit pouze jednu položku *celé jméno*, podotázky - části celého jména budou vyplněny automaticky.



Obrázek 5.12: Diagram struktury celého jména

Při automatickém vyplňování částí může dojít k chybám a sporným případům, kdy části vyplnit nelze. Příkladem u celého jména může být použití dvouslovných jmen a příjmení nebo vícera titulů[1], těchto variant je nespočet a nelze je všechny předem specifikovat. Automatické vyplňování částí proto probíhá v reálném čase při vyplňování celku, uživatel tak může při psaní kontrolovat doplňování podotázek a v případě špatného doplnění chybu opravit nebo části dopsat jednotlivě.

Na následujícím obrázku je ukázka stavu formuláře po vyplnění pouze pole Celé jméno. Ostatní pole byly doplněny automaticky.



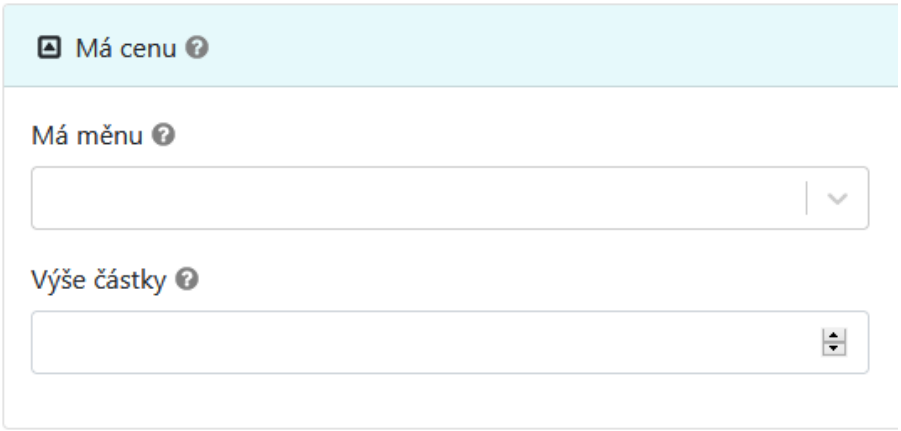
Obrázek 5.13: Kompozitní otázka Celé jméno

[1] `https://prirucka.ujc.cas.cz/?id=326`

5.4 Otázka s jednotkou

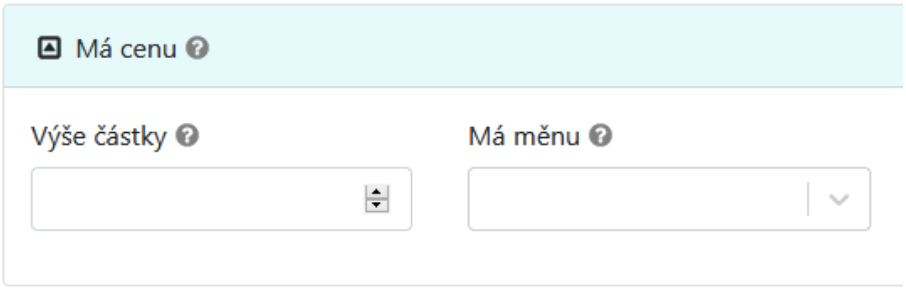
Komponenta dokáže zobrazit dvě otázky propojené vazbou „má jednotku" vedle sebe. Což by mělo vést ke zlepšení čitelnosti jednotlivých otázek, jelikož spolu úzce souvisí. Jednotky se uvádí například u vzdálenosti nebo u cenové hodnotě jako měna. (FR9)

Aktuálně se tyto dvě otázky zobrazují klasicky pod sebe.



Obrázek 5.14: Aktuální zobrazování otázky ohledně měny

A na následujícím obrázku je znázorněno zobrazování otázek vedle sebe.



Obrázek 5.15: Řešení zobrazování otázky ohledně měny

5.5 Přepínač zobrazit více

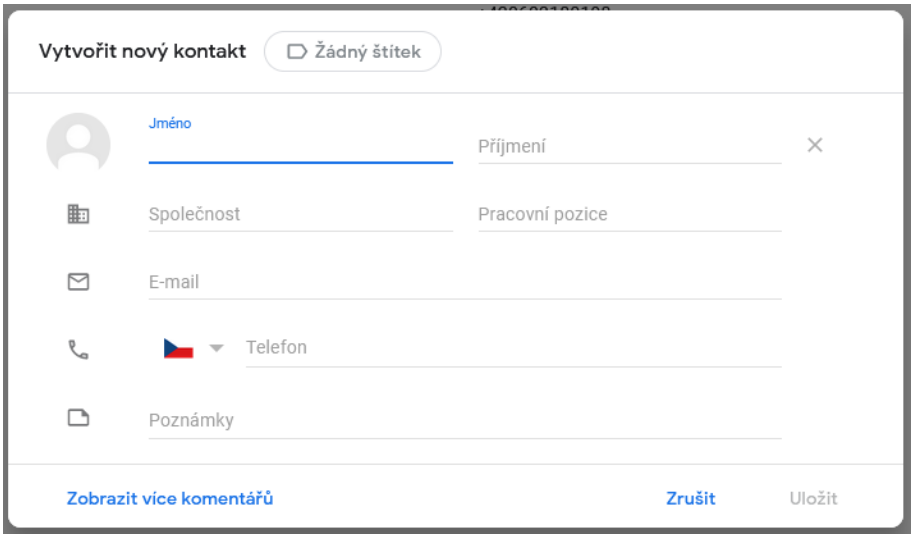
Formuláře pro sběr dat mohou obsahovat až stovky otázek, to může způsobit, že se uživatelé ve formuláři budou ztrácet a vyplní některá data špatně. Spousta otázek zůstává také ve většině případů nevyplněna. Cílem této komponenty je možnost skrýt málo vyplňované pokročilé otázky a zobrazovat je až poté, co uživatel přepne přepínač „Zobrazit více". (FR5)

5.5.1 Existující implementace

V této sekci jsou ukázány existující implementace zobrazování pokročilých otázek. Byly zvoleny aplikace kontakty, jelikož se jedná o běžné formuláře s větším množstvím polí.

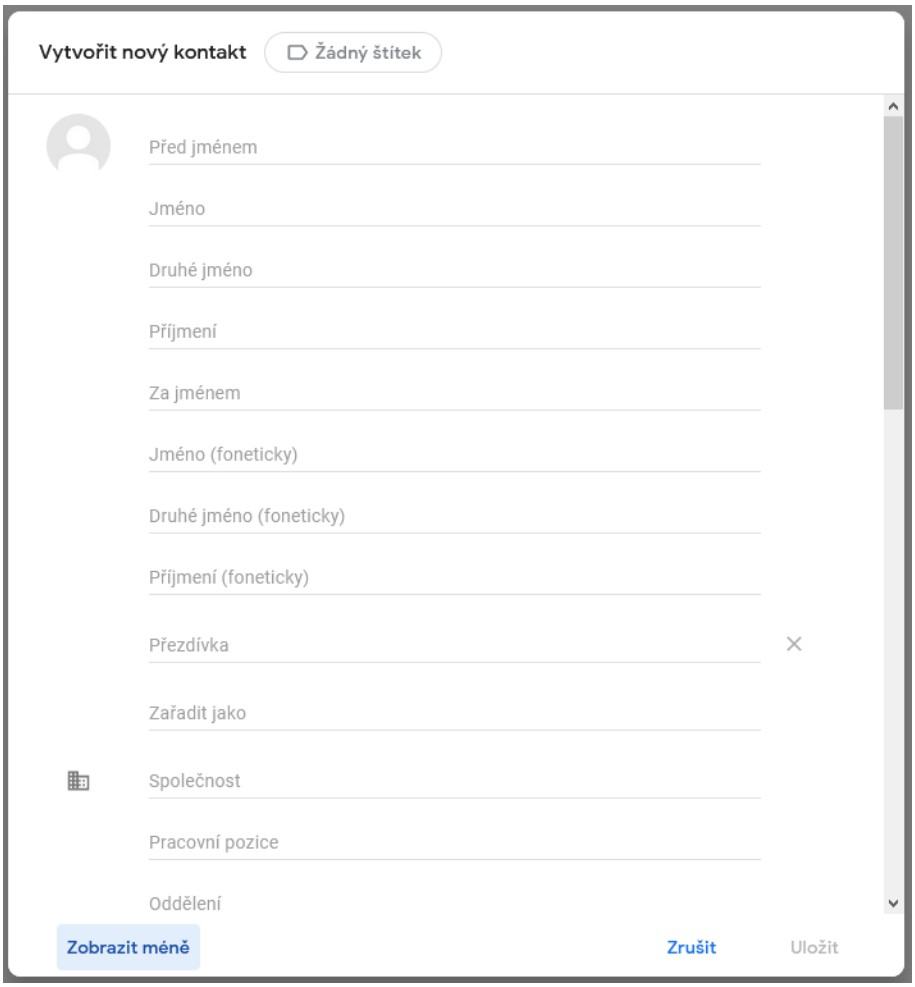
Google Kontakty

Jako prvním příkladem je webová stránka Google Kontakty, konkrétně formulář pro přidání nového kontaktu.



Obrázek 5.16: Google Kontakty - Formulář přidání kontaktu

Po stisku „Zobrazit více komentářů" se do formuláře přidají další pole k vyplnění. Tato akce není doprovázena žádnou animací. Akci lze navrátit tlačítkem „Zobrazit méně".

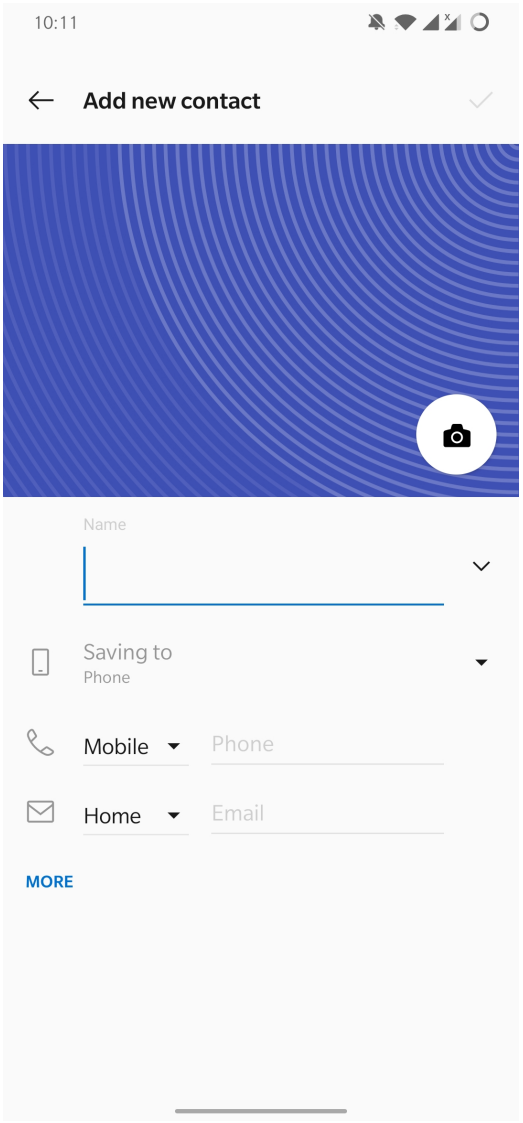


Obrázek 5.17: Google Kontakty - Formulář přidání kontaktu po stisku „Zobrazit více komentářů"

Pokročilá pole se po kliknutí na „Zobrazit méně" skryjí, přestože můžou zůstat některá vyplněna. Uživatel tak nemá přehled o vyplněnosti formuláře dokud znovu nevybere možnost „Zobrazit více komentářů".

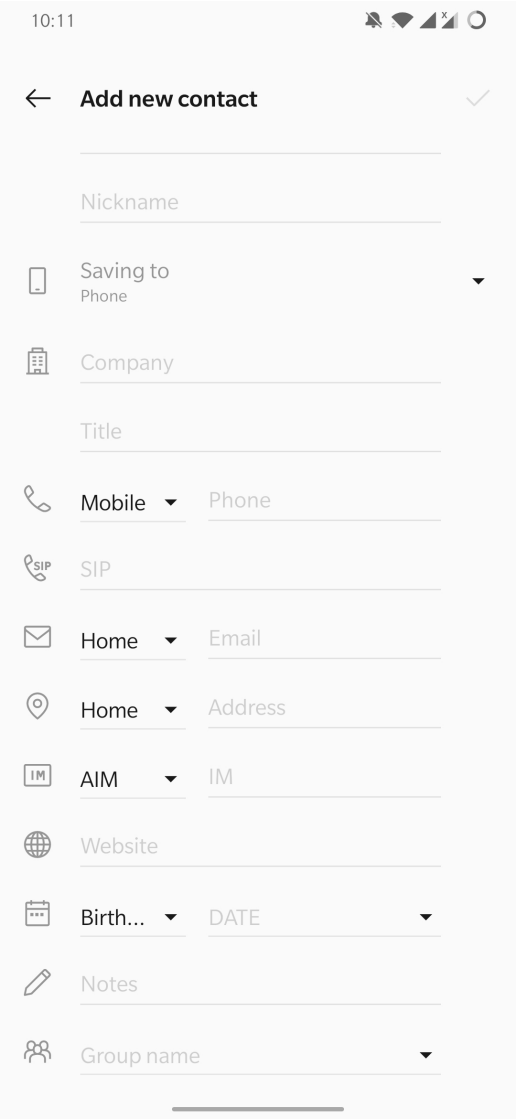
Aplikace Kontakty na platformě Android

Dalším příkladem je mobilní aplikace kontakty od společnosti OnePlus a formulář pro přidání kontaktu.



Obrázek 5.18: Aplikace Kontakty - Přidání kontaktu

Tlačítko Více vloží do formuláře další formulářová pole, což je doprovázeno animací.

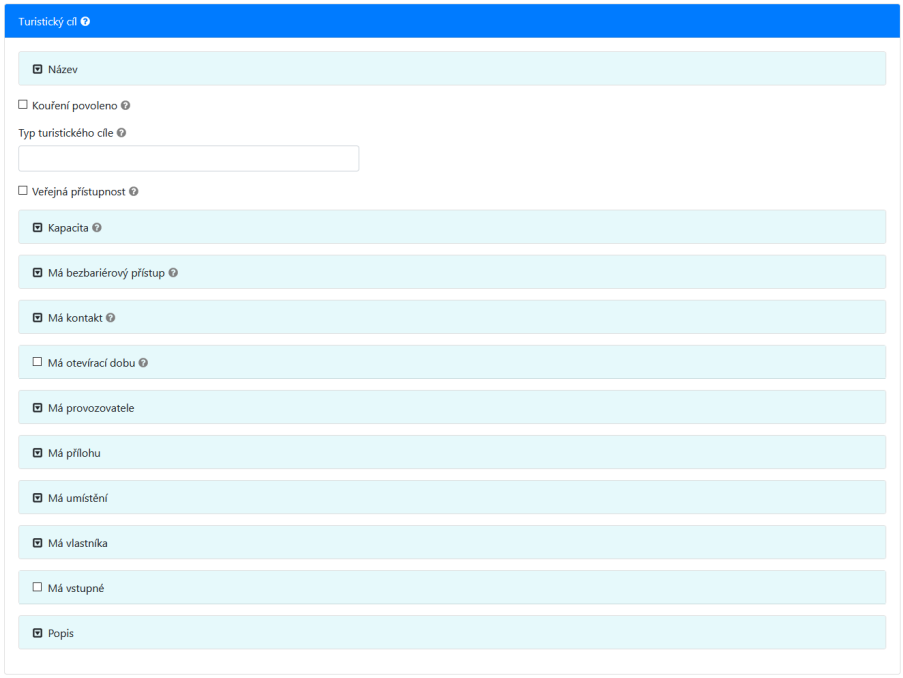


Obrázek 5.19: Aplikace Kontakty - Přidání kontaktu po stisku více („More")

Animace je krátká a nepůsobí rušivě, naopak podporuje rychlou orientaci v nově odkrytých polích. Akci nelze vrátit zpět, respektive nelze znovu skrýt pokročilá pole.

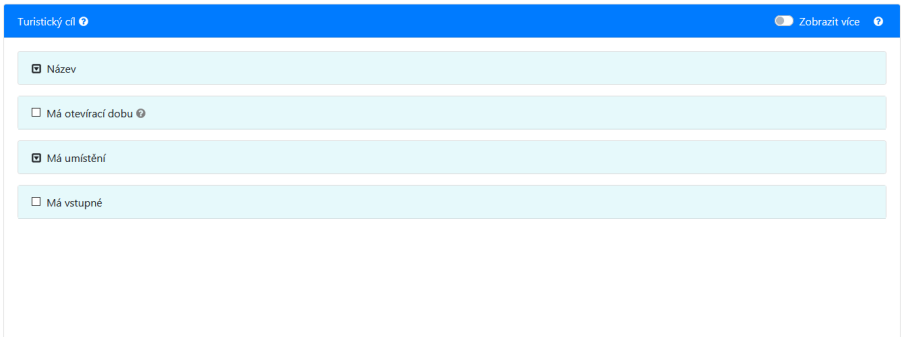
5.5.2 Vlastní návrh

Na následujícím obrázku je ukázka aktuálního stavu formuláře Turistický cíl. Je nutné upozornit na větší množství sekcí.



Obrázek 5.20: Aktuální stav formuláře Turistický cíl

Komponenta přidává přepínač „Zobrazit více" na pravou stranu záhlaví sekcí a celého formuláře. Uživatel si může skryté pokročilé otázky znovu zobrazit pomocí přepínače, ty se pak zobrazí na stejném místě jako před skrytím. Takto vypadá formulář po skrytí pokročilých otázek.



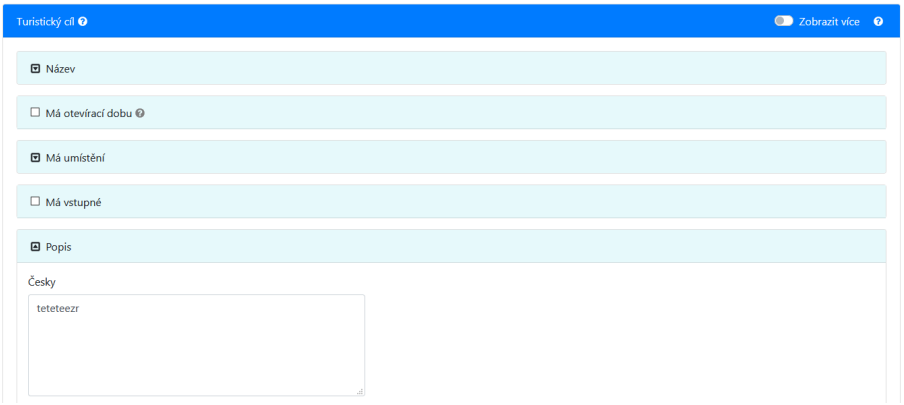
Obrázek 5.21: Formulář Turistický cíl po skrytí otázek

„Zobrazit více" se může ukazovat i na sekcích, pokud jsou otevřené.

Obrázek 5.22: „Zobrazit více" v záhlaví otevřené sekce

Jestli uživatel vyplní pokročilou otázku a poté pokročilé otázky skryje, vyplněná otázka zůstane zobrazena. Jinak řečeno pokročilé otázky s odpovědí zůstávají

viditelné. To je ukázáno na dalším obrázku, kde byl vyplněn popis - což je pokročilá otázka.



Obrázek 5.23: Vyplněná pokročilá otázka zůstává zobrazena

Zde se nabízí možné vylepšení serverové části poskytující data formuláře, ta by byla teoreticky schopna přizpůsobovat formulář jednotlivým uživatelům nebo skupinám uživatelů na základě předchozích odeslaných odpovědí a otázky dynamicky označovat jako pokročilé. Po prvním odeslání odpovědi na pokročilou otázku by již při dalším vyplňováním nebyla označena jako pokročilá.

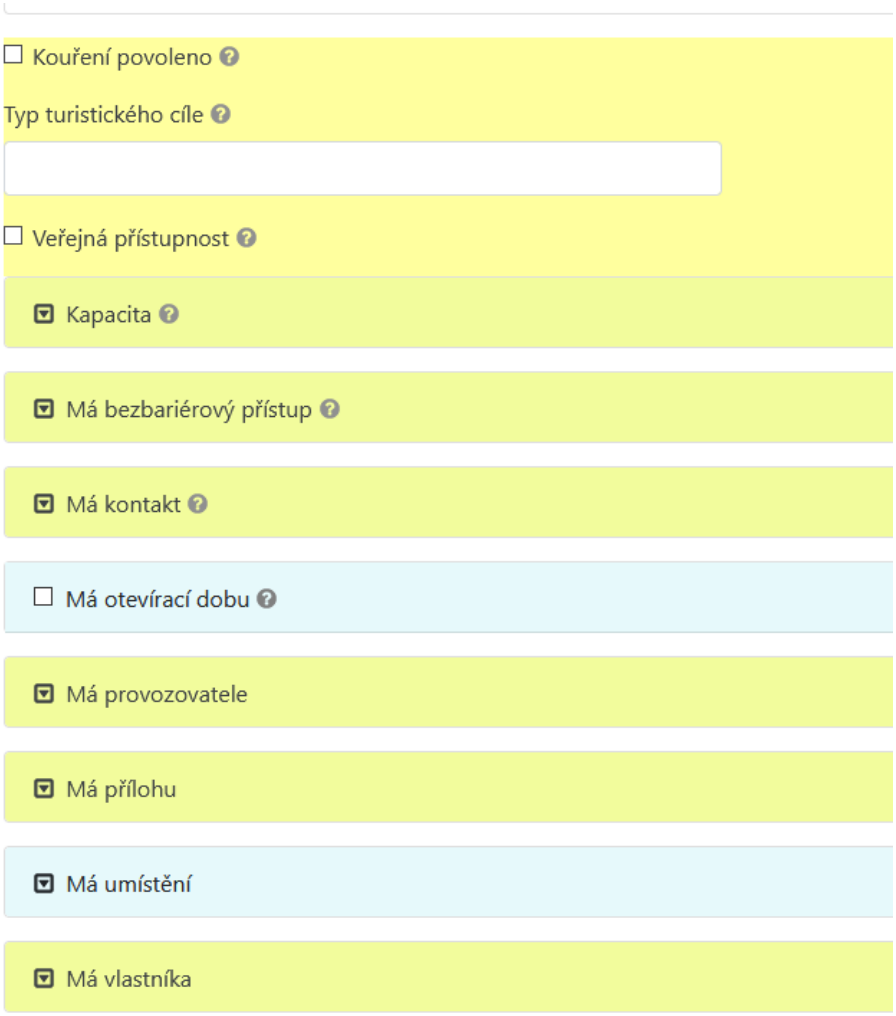
Samotná komponenta na pozadí používá otázku se zaškrtávacím polem, kterou skryje. Z této otázky převezme popis, nápovědu, a zobrazí ji jako přepínač v záhlaví sekcí. (FR12:1) Bez této komponenty bude skrývání pokročilých otázek fungovat, ale místo přepínače bude viditelné zaškrtávací pole jako normální otázka. (NFR7)

5.6 Zvýraznění otázek po zobrazení

V SForms se uživatel často dostane do situace, kde se po vyplnění jedné otázky zobrazí další doplňující otázky v jiné části formuláře - např. dokončení jedné sekce vytvoří sekci novou. Tyto změny nemusí být pro uživatele viditelné a nemusí tak pochopit důvod odkrytí nových prvků.

Zvýraznění funguje tak, že po odkrytí nových otázek či sekcí je na malou chvíli barevně podbarví. Uživateli viditelně zvýrazní případné nové prvky ve formuláři. (FR11)

Tato funkcionalita výborně doplňuje předchozí komponentu „Přepínač zobrazit více”, která je popsána v předchozí sekci. V následujícím obrázku je zachyceno toto dočasné barevné zvýraznění.

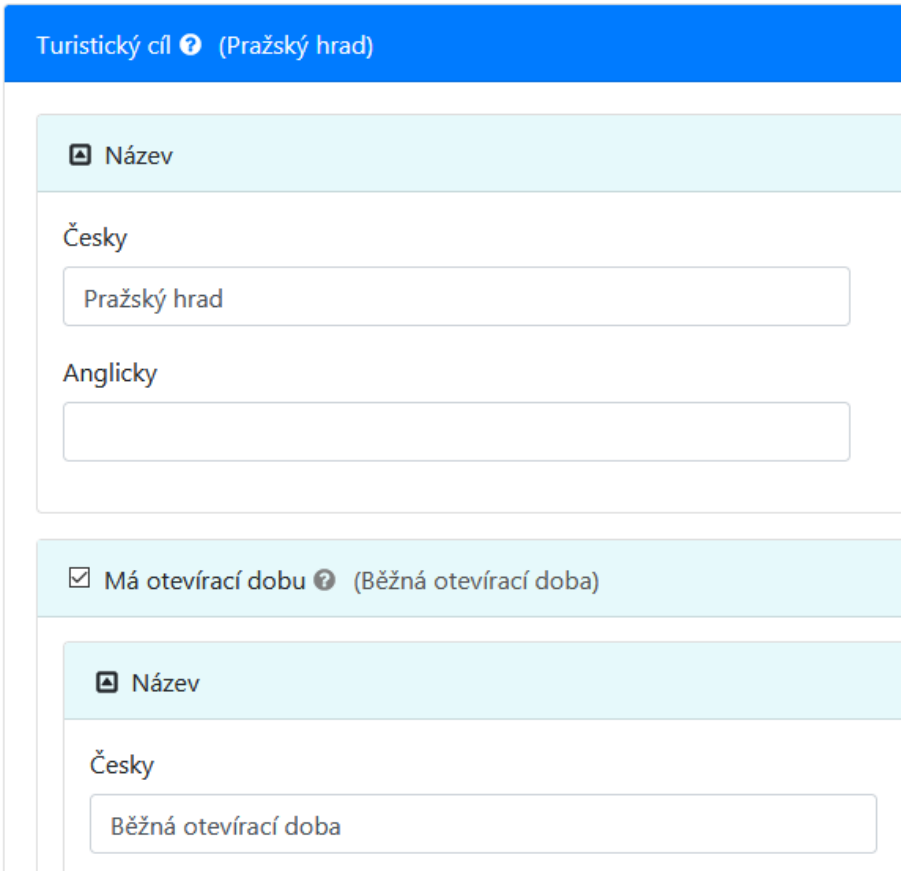


Obrázek 5.24: Ukázka podbarvení otázek a sekcí po vykreslení pokročilých otázek

5.7 Identifikátor sekcí

Formuláře jsou členěny na velké množství sekcí a podsekcí, uživatelé mohou při vyplňování zapomenout kontext jednotlivých otázek. Některé otázky proto lze označit jako identifikační pro danou sekci. Komponenta pak zajistí zobrazení odpovědí na tyto otázky v záhlaví sekce. Text se zobrazuje přestože je sekce zavřená, to usnadňuje orientaci ve formuláři - například když by se uživatel chtěl k nějaké otázce vrátit. (FR3)

Na dalším obrázku je ukázka dvou identifikačních otázek a zobrazení příslušných odpovědí v sekcích.

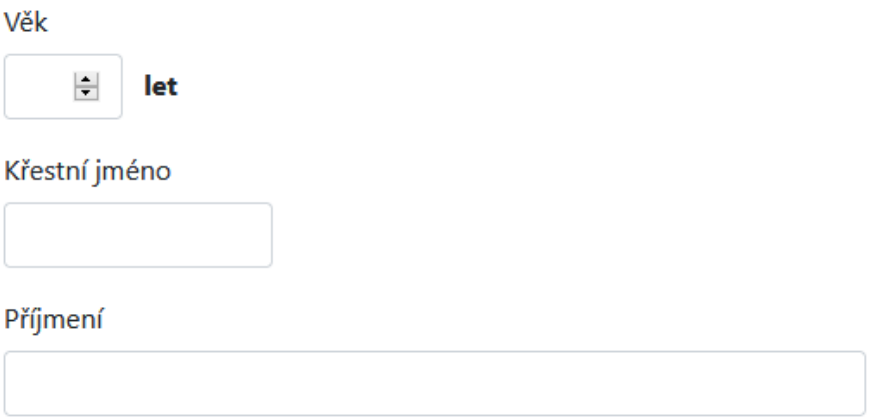


Obrázek 5.25: Ukázka zobrazení hodnoty identifikační otázky v záhlavích sekcí

5.8 Šířka polí

Funkcionalita přidává schopnost omezit maximální šířku textových polí definováním počtu znaků.

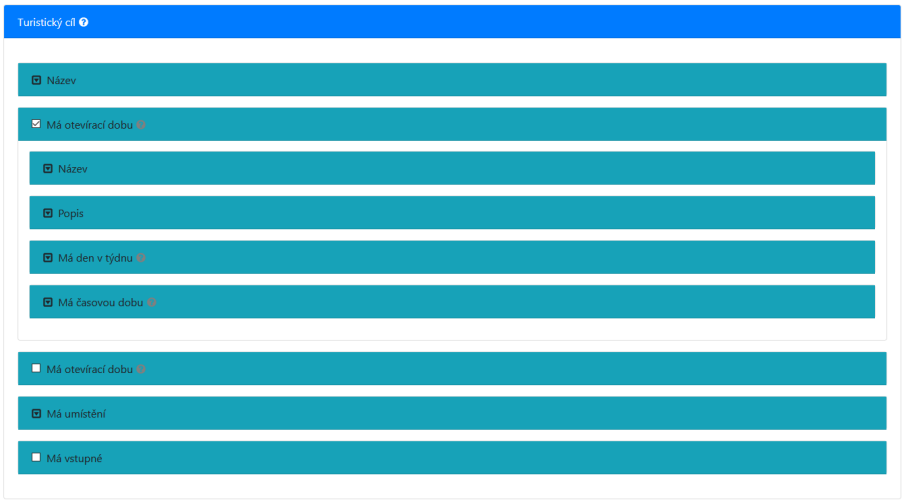
Větším počtem různých šířek polí by snadno mohlo dojít k znepřehlednění formuláře, proto by se nemělo používat více než 3 různé šířky - např. krátké pole pro otázky na věk, střední na křestní jména osob a standardní pro zbytek. (FR10)



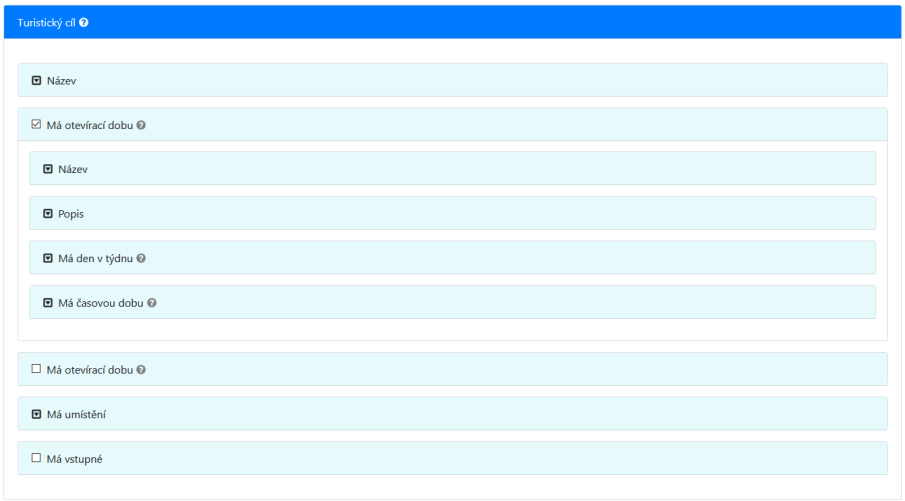
Obrázek 5.26: Ukázka nastavení různých délek polí

5.9 Úprava barev sekcí

Na základě zpětné vazby od uživatelů byly upraveny barvy sekcí, jelikož působily příliš rušivě. Na následujících obrázcích je možno srovnat změnu barev.



Obrázek 5.27: Barvy sekcí v aktuální implementaci



Obrázek 5.28: Barvy sekcí po změně

Barva sekcí byla upravena v souladu se standardy pro usnadnění přístupu k webovému obsahu WCAG[2], a výsledný kontrastní poměr mezi barvou textu a pozadí je alespoň 7:1 (úroveň AAA) [22].

[2]Web Content Accessibility Guidelines

Kapitola 6

Implementace komponent

Implementace se dělí na úpravu SForms a na vytvoření vlastní samostatné knihovny s-forms-smart-components. Úpravy jako je změna barev, sekce s odpovědí a zvýraznění otázek po zobrazení byly provedeny přímo v SForms, zbytek komponent se pak nachází právě ve zmiňované samostatné knihovně.

Všechny popsané návrhy z předchozí kapitoly byly implementovány. V následujících sekcích je popsána implementace vybraných změn.

6.1 Mapovací pravidla

Při inicializaci lze knihovně SForms nově předat pole obsahující objekty se třídou komponenty a mapovací funkcí. SForms při generování formuláře poté pro každou otázku zavolá tyto mapovací funkce, pokud některé z těchto funkcí vrátí hodnotu `true`, příslušná komponenta bude vybrána k vykreslení. V opačném případě se otázka vykreslí standardním způsobem.

Samotná mapovací funkce přijímá v parametru objekt s definicí otázky tak, jak je zapsaná ve zdrojovém JSON-LD formuláře a rozhoduje se na základě struktury otázky či na základě přítomností určitých vlastností. V následujícím úryvku se nachází zjednodušený příklad mapovacího pravidla pro fiktivní komponentu NameQuestion, která se má použít v případě, že datový typ odpovědi odpovídá termínu `foaf:name`.

```
{
  component: NameQuestion,
  mapRule: question => question['has-datatype'] === 'foaf:name'
}
```

Úryvek 6.1: Příklad mapovacího pravidla

Tato změna umožňuje do budoucna další jednoduché rozšiřování SForms bez nutnosti zasahovat do samotné knihovny. Vývojáři si také sami mohou specifikovat, jaké chytré komponenty budou chtít používat. (FR13, FR14, NFR4)

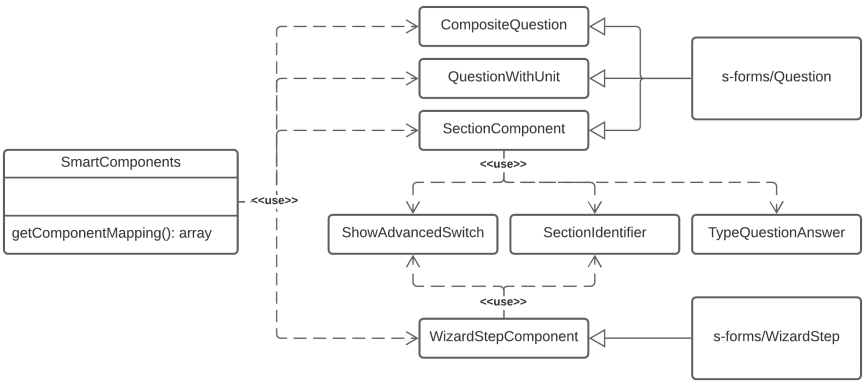
6.2 SForms Smart Components

Chytré komponenty byly implementovány jako samostatná knihovna s-forms-smart-components. Knihovna samotná je napsaná v programovacím jazyce JavaScript

a je, stejně jako SForms, založená na frameworku React[1] a používá prvky z knihovny React Bootstrap[2]. Kvůli kompatibilitě byly verze těchto knihoven zvoleny tak, aby se shodovaly s využitými verzemi v SForms.

K sestavení projektu je využíván Babel[3] a Parcel[4]. Součástí je i testovací aplikace s formulářem, která sloužila především pro vývoj knihovny.

Na obrázku dále je znázorněn zjednodušený diagram tříd.



Obrázek 6.1: Zjednodušený diagram tříd v knihovně s-forms-smart-components

Funkce `getComponentMapping()` ve třídě `SmartComponents` vrací pole s definovanými mapovacím pravidly pro komponenty. Toto pole pak stačí předat knihovně SForms. Princip mapovacích pravidel je popsán v předchozí sekci.

6.3 Filtrovací otázka na základě typu

Nejprve je třeba definovat strom typů, ze kterých bude nabízen výběr, k tomu je využíván termín `skos:broader` ze slovníku SKOS[5]. Tímto termínem se specifikuje hierarchická struktura jednotlivých typů.

K označení navzájem disjunktních typů se používá termín `owl:disjointWith` ze slovníku OWL.[6] V následujícím úryvku je příklad části konfigurace formuláře, která popisuje právnickou a fyzickou osobu tak, jak bylo zmíněno v minulé kapitole s návrhem.

[1] `https://reactjs.org/`
[2] `https://react-bootstrap.github.io/`
[3] `https://babeljs.io/`
[4] `https://v2.parceljs.org/`
[5] `https://www.w3.org/TR/skos-reference/`
[6] `https://www.w3.org/TR/owl-guide/`

```
{
  "@id": "pravnicka-osoba",
  "label": "Pravnicka osoba"
},
{
  "@id": "fyzicka-osoba",
  "label": "Fyzicka osoba",
  "owl:disjointWith": [
    {"@id": "pravnicka-osoba"}
  ]
},
{
  "@id": "fyzicka-osoba--nezletila",
  "label": "Fyzicka osoba nezletila",
  "skos:broader": { "@id": "fyzicka-osoba" },
  "owl:disjointWith": [
    { "@id": "pravnicka-osoba" }
  ]
},
{
  "@id": "fyzicka-osoba--chytra",
  "label": "Fyzicka osoba chytra",
  "skos:broader": { "@id": "fyzicka-osoba" },
  "owl:disjointWith": [
    { "@id": "pravnicka-osoba" }
  ]
},
{
  "@id": "fyzicka-osoba--hloupa",
  "label": "Fyzicka osoba hloupa",
  "skos:broader": { "@id": "fyzicka-osoba" },
  "owl:disjointWith": [
    { "@id": "pravnicka-osoba" },
    { "@id": "fyzicka-osoba--chytra" }
  ]
}
```

Úryvek 6.2: Úryvek konfigurace typů pro filtrovací otázku ve formátu JSON-LD

Po definici typů je již možné používat filtrovací otázky. K zobrazení komponenty se kontroluje přítomnost položky `type-question` ve vlastnosti `has-layout-class`, která se v SForms používá pro úpravu vzhledu otázky.

Existující vlastnost `has-possible-value` SForms používá k nastavení možností pro standardní formulářovou nabídku. Komponenta tuto vlastnost používá ke stejnému účelu — do seznamu se vloží odkazy na předem popsané typy, které se mají uživateli nabízet. Tento přístup má několik výhod. Typy je nutné definovat pouze jednou, což je užitečné v případě, že formulář obsahuje více podobných otázek. Další výhodou je možnost ze složité struktury typů vybrat pouze určitou část.

Nová vlastnost `has-type-question`, rozšiřující ontologii SForms, je určena pro specifikování podotázek, do nichž se na pozadí budou vkládat uživatelem vybrané typy. Tyto podotázky jsou pro uživatele neviditelné. SForms neumožňuje jedné otázce přiřadit více odpovědí. Počet podotázek v `has-type-question` určuje maximální počet vybraných typů.

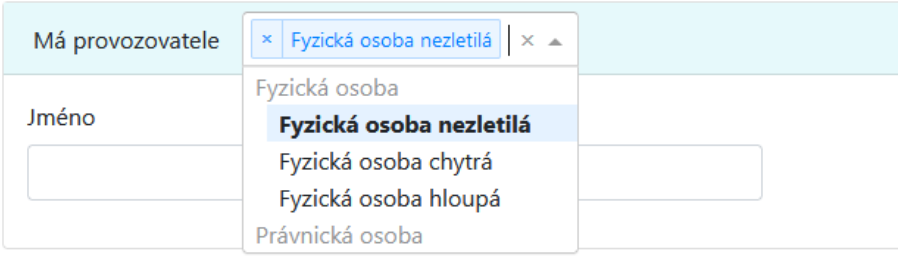
Filtrovací otázka také umožňuje specifikovat, které typy nebude možné v otázce

vybrat, a to díky nové vlastnosti `has-non-selectable-value`. Možnosti ale v nabídce budou stále viditelné.

```
{
  "@id": "provozovatel-section",
  "@type": "doc:question",
  "has_related_question": [
    "ps-type-1",
    "ps-type-2",
    "provozovatel-jmeno"
  ],
  "has-type-question": [
    "ps-type-1",
    "ps-type-2"
  ],
  "has-layout-class": [
    "answerable",
    "section",
    "type-question"
  ],
  "has-possible-value": [
    "fyzicka-osoba",
    "pravnicka-osoba",
    "fyzicka-osoba--nezletila",
    "fyzicka-osoba--chytra",
    "fyzicka-osoba--hloupa"
  ],
  "has-non-selectable-value": [
    "fyzicka-osoba"
  ],
  "label": "Ma provozovatele"
}
```

Úryvek 6.3: Příklad filtrovací otázky ve formátu JSON-LD

Výsledná otázka může vypadat následovně.



Obrázek 6.2: Výsledné vykreslení otázky Má provozovatele

Pro vykreslování samotné nabídky se stromem typů byla použita knihovna intelligent-tree-select.[7]

[7] `https://github.com/lecbyjak/intelligent-tree-select`

6.4 Kompozitní otázka

Implementaci je možné rozložit do dvou hlavních částí. První je schopnost rozdělit kompozitní otázku do částí, druhá je spojení částí do celku, pokud se uživatel rozhodne vyplnit raději části.

K rozdělování kompozitní otázky je používán předem definovaný regex[8], jehož dílčí části slouží ke správnému přiřazování do podotázek. Tento regex lze definovat vlastností `has-pattern`. Pro vkládání do podotázek ve správném pořadí se používá seznam `has-composite-variables`, který obsahuje odkazy na specifické podotázky. Kvůli univerzálnosti se nepoužívá existující seznam `has_related_question`, ten je určen především k definici hierarchie otázek. Je nutné rozšířit ontologii, kterou používají formuláře SForms o tyto nové definice. Použití kompozitní otázky se detekuje právě přítomností vlastnosti `has-composite-pattern`.

Kompozitní otázka *Celé jméno* pak může vypadat následovně.

```
{
  "@id": "sectionfoo-1592",
  "@type": "doc:question",
  "has_related_question": [
    "title-7183",
    "first-name-9402",
    "last-name-6610"
  ],
  "has-composite-variables": [
    "title-7183",
    "first-name-9402",
    "last-name-6610"
  ],
  "has-layout-class": "name",
  "has-datatype": "foaf:name",
  "has-composite-pattern": "?1 ?2 ?3",
  "has-pattern": "^(?:([A-Za-z]{1,4}\\.) )?(.+) (.+)$",
  "requires-answer": true,
  "label": "Cele jmeno"
}
```

Úryvek 6.4: Definice otázky Celé jméno ve formátu JSON-LD

Pro rozdělení celého jména je použit regex
`^(?:([A-Za-z]{1,4}\\.) )?(.+) (.+)$`, který lze kvůli jednoduchosti aplikovat pouze na nejběžnější jména. Zahrnuje tituly maximální délky 4, které končí tečkou.

Pro implementaci byla rozšířena třída Question z knihovny SForms. Úpravou existující funkce `onAnswerChange` je možné zachytit změny v kompozitní otázce. Pak je možné na hodnotu aplikovat regex a části přiřadit podotázkám.

Pokud nemůže být regex aplikován, podotázky se vyprázdní a uživatel je tak donucen, aby případně ručně vyplnil části.

Druhou částí implementace je sloučení částí do celku. K tomu je v kompozitní otázce definována vlastnost `has-composite-pattern`. Do tohoto textového řetězce nahrazovány za otazníky s indexem hodnoty příslušných podotázek. V případě celého jména se jedná jednoduše o řetězec `?1 ?2 ?3`, kde `?1` symbolizuje první podotázku - titul, další pak jméno a poslední příjmení. K pořadí se používá již dříve popsaná

[8] Regulární výraz / Regular Expression

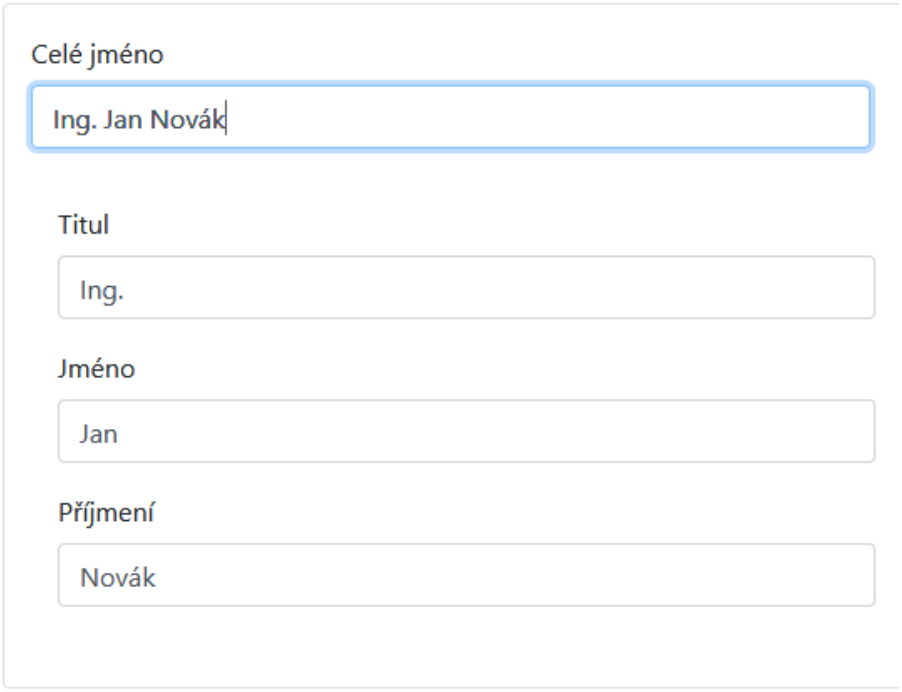
vlastnost `has-composite-variables`. Rozšířením funkce `onSubQuestionChange` lze detekovat změny v podotázkách. Pak už zbývá jenom do řetězece `has-composite-pattern` nahradit jednotlivé podotázky.

Kompletní kompozitní otázka *Celé jméno* pak může být definována takto:

```
{
  "@id": "section-cele-jmeno-1592",
  "@type": "doc:question",
  "has_related_question": [
    "title-7183",
    "first-name-9402",
    "last-name-6610"
  ],
  "has-composite-variables": [
    "title-7183",
    "first-name-9402",
    "last-name-6610"
  ],
  "has-layout-class": "name",
  "has-datatype": "foaf:name",
  "has-composite-pattern": "?1 ?2 ?3",
  "has-pattern": "^(?:([A-Za-z]{1,4}\\.) )?(.+) (.+)$",
  "requires-answer": true,
  "label": "Cele jmeno"
},
{
  "@id": "title-7183",
  "@type": "doc:question",
  "has_related_question": [],
  "has-layout-class": "text",
  "label": "Titul"
},
{
  "@id": "first-name-9402",
  "@type": "doc:question",
  "has_related_question": [],
  "has-layout-class": "text",
  "has-preceding-question": "title-7183",
  "label": "Jmeno"
},
{
  "@id": "last-name-6610",
  "@type": "doc:question",
  "has_related_question": [],
  "has-layout-class": "text",
  "label": "Prijmeni"
}
```

Úryvek 6.5: Definice kompozitní otázky Celé jméno

Výsledný formulář je pak zobrazen na následujícím obrázku. Vyplněno bylo pouze pole celé jméno, zbylé vstupy byly správně doplněny automaticky.



Obrázek 6.3: Formulář s kompozitní otázkou celé jméno

6.5 Otázka s jednotkou

K detekci otázky s jednotkou se používá nový termín `has-unit-of-measure-question`, jehož hodnota odkazuje na otázku určující jednotku. Tyto dvě otázky se pak vykreslí do formuláře vedle sebe, musí být ovšem na stejné úrovni (stejná rodičovská otázka).

6.6 Přepínač zobrazit více

Komponenta je závislá na již existující funkci SForms `is-relevant-if`, ta se stará o podmíněné zobrazení otázek. Díky této vlastnosti lze otázku vykreslovat například pouze v případě, že jiná otázka má specifickou odpověď. Toho bylo využito při implementaci přepínače zobrazit více.

Přepínač na pozadí využívá podotázku označenou jako `show-advanced-question: true`, samotnou otázku uživateli skrývá. Z definice otázky se pak přebírá název (případně i nápověda), ten se pak zobrazuje vedle přepínače. Při kliknutí na přepínač se do této podotázky vloží na pozadí odpověď s hodnotou `true` nebo `false`. Z odeslaných odpovědí tak lze vyčíst stav přepínače. (FR12:1)

Pokročilé otázky, které se mají skrývat za zobrazit více, mají určené pravidlo `is-relevant-if`, odkazující na podotázku přepínače. To pak zaručí vykreslení jenom v případě, že má přepínač hodnotu `true`.

V následujícím úryvku je příklad sekce s přepínačem.

```
{
  "@id": "parent-section-1590",
  "@type": "doc:question",
  "has_related_question": [
    "show-advanced-3887"
  ],
  "has-layout-class": "section",
  "label": "Sekce"
},
{
  "@id": "show-advanced-3887",
  "@type": "doc:question",
  "has_related_question": [],
  "has-layout-class": "checkbox",
  "label": "Zobrazit vice",
  "description": "Napoveda",
  "show-advanced-question": true
}
```

Úryvek 6.6: Sekce s přepínačem zobrazit více ve formátu JSON-LD

Předchozí příklad se vykreslí následovně:

Obrázek 6.4: Ukázka přepínače zobrazit více

6.7 Zvýraznění otázek po zobrazení

Přidáním nové položky `emphasise-on-relevant` do seznamu `has-layout-class` se při vykreslování formuláře HTML elementu s otázkou nastaví CSS třída, která má nastavenou animaci při zobrazení. V následujícím úryvku se nachází definice této CSS třídy a samotné animace.

```
.emphasise-on-relevant {
  animation: emphasiseOnRelevant 1.2s ease-in;
}

@keyframes emphasiseOnRelevant {
  0% {
    display: none;
    opacity: 0;
  }

  1% {
    display: block;
    opacity: 0;
    background-color: yellow;
  }

  25% {
    display: block;
    opacity: 0;
  }

  75% {
    display: block;
    opacity: 1;
  }

  100% {
    display: block;
    opacity: 1;
    background-color: transparent;
  }
}
```

Úryvek 6.7: CSS animace pro zvýraznění otázek po zobrazení

Tato změna byla provedena přímo v SForms.

6.8 Identifikátor sekcí

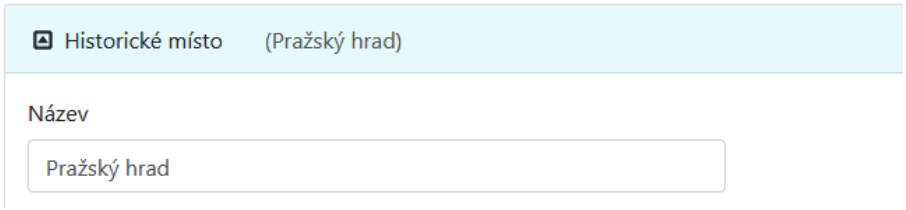
Identifikátor sekce se zobrazí v případě, že daná sekce má novou vlastnost `has-identifying-question` obsahující odkaz na otázku, z níž se bude vepsaná odpověď zobrazovat v záhlaví sekce.

V následujícím úryvku je příklad sekce Historické místo s identifikační otázkou Název.

```
{
  "@id": "historicke-misto-section-1590",
  "@type": "doc:question",
  "has_related_question": [
    "nazev-3887"
  ],
  "has-identifying-question": [
    "nazev-3887"
  ],
  "has-layout-class": "section",
  "label": "Historicke misto"
},
{
  "@id": "nazev-3887",
  "@type": "doc:question",
  "has_related_question": [],
  "has-layout-class": "text",
  "label": "Nazev"
}
```

Úryvek 6.8: Sekce historické místo s identifikační otázkou ve formátu JSON-LD

Předchozí příklad se po vyplnění názvu uživatelem vykreslí následovně:



Obrázek 6.5: Příklad identifikátoru sekce Historické místo

6.9 Šířka polí

Funkcionalita pro nastavení šířky polí byla implementována přímo do SForms. Nově přidanou vlastností `initial-input-length` lze počtem znaků specifikovat šířku pole ve formuláři. Počet znaků se přímo přenese do stylu textového pole jako šířka s jednotkou `ch`, což je relativní jednotka jejíž základem je šířka znaku `0` (nula) [23]. Pro zjednodušení je `1ch` bráno jako průměrná šířka znaku.

Kapitola 7

Uživatelské testování

Tato kapitola se zabývá uživatelským testováním formulářů SForms a implementovaných chytrých komponent. Cílem tohoto testování je ověřit si funkčnost, celkovou uživatelskou přívětivost těchto komponent a jestli řeší nalezené problémy ve formulářích.

7.1 Testování 1

První testování proběhlo již při samotném vývoji jednotlivých komponent. V době testu byla implementována pouze komponenta sekce s odpovědí.

Testování prováděli 4 uživatelé — odborníci, kteří jsou s danou problematikou seznámeni, navrhli i vlastní způsob řešení některých nalezených problémů.

Na základě zpětné vazby od uživatelů byly dodefinovány některé funkční a nefunkční požadavky. Jedná se především o NFR10, týkající se doby vykreslování změn. Dlouhou dobu vykreslení zmiňovali všichni uživatelé.

Některé připomínky se shodují s problémy popsanými v předchozích kapitolách.

Jeden z uživatelů měl připomínku k implementované komponentě sekce s odpovědí — zobrazení této sekce je podle něj nekonzistentní s již existujícími sekcemi.

7.2 Testování 2

Druhé testování testování se uskutečnilo po dokončení implementace chytrých komponent a probíhalo podle předem specifikovaných scénářů, které jsou k nalezení v Příloze A.

V prvním scénáři (v Příloze A.1) uživatel vyplní formulář ukázkovými daty, ve druhé scénáři (v Příloze A.2) pak uživatel zhodnocuje funkce formulářů a implementovaných komponent.

Testování proběhlo online přes platformu Microsoft Teams a podíleli se na něm 3 uživatelé, z nichž se v dané problematice orientoval pouze jeden. Relevantní zpětná vazba k jednotlivým scénářům se nachází v Příloze B.

Jednou z výtek je nepřehledně umístěné tlačítko „Zobrazit více”, uživatelé si ho ze začátku vůbec nevšimli. Jako velice přívětivá změna je otázka na základě typu. Zpětná vazba obsahuje i návrhy k budoucímu zlepšení či další úpravě komponent.

Závěr

Cílem této práce bylo navrhnout a implementovat chytré komponenty. Podařilo se mi navrhnout 7 znovupoužitelných komponent a další 2 funkcionality rozšiřující možnosti generování formulářů v knihovně SForms. Tyto komponenty se zobrazují na základě významu dat a struktury jednotlivých otázek.

Samotná knihovna SForms byla modifikována tak, aby umožňovala do budoucna jednoduché rozšíření o další chytré komponenty bez nutnosti zásahu do SForms, a to za pomoci specifikace takzvaných mapovacích pravidel, které vyhodnocují význam otázky.

Chytré komponenty byly navrženy na základě funkčních i nefunkčních požadavků, které vyplývaly z analýzy aktuálního stavu zobrazovaných formulářů a popisu nalezených problémů.

Tyto komponenty pak byly podrobeny uživatelskému testování, které proběhlo jak v průběhu vývoje, tak na jeho konci. V uživatelském testování byla ověřena funkčnost a celková uživatelská přívětivost nově zobrazovaných komponent.

Část implementovaných komponent se již používá ve Správci otevřených dat, který byl vytvořen v rámci projektu Evropské unie KODI.

V teoretické části práce byly představeny některé populární technologie pro sémantický web. Dále byly popsány již existující řešení pro sběr sémantických dat na webu.

SForms nabízí prostor pro další vylepšení a návrh nových komponent, jako třeba prvek pro zadávání GPS polohy pomocí mapy, nebo prvek podporující vícejazyčné odpovědi.

Za zmínku stojí potřeba zrychlit vykreslování formuláře po uživatelem provedené akci, které je v případě většího množství zobrazovaných otázek příliš pomalé.

Výstupem práce je samostatná knihovna s chytrými komponentami s-forms-smart-components. Zabalený zdrojový kód knihovny je přiložený k této práci, v příloze se také nachází odkaz na aktuální repozitář a návod ke zprovoznění.

Osobním přínosem práce jsou pro mě nově nabyté vědomosti, týkající se především sémantických webových technologií, a prohloubení znalosti programovacího jazyku JavaScript včetně frameworku React.

Seznam použitých zdrojů

1. *W3C Semantic Web Frequently Asked Questions* [online]. W3C, 2009 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/2001/sw/SW-FAQ.
2. *Main Page* [online]. W3C, [2009] [cit. 2021-05-16]. Dostupné z: https://www.w3.org/2001/sw/wiki/Main_Page.
3. *Linked Data* [online]. W3C, c2015 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/standards/semanticweb/data.
4. CYGANIAK, Richard; WOOD, David; LANTHALER, Markus. *RDF 1.1 Concepts and Abstract Syntax* [online]. W3C, 2014 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/Overview.html.
5. BRICKLEY, Dan; GUHA, R.V. *RDF Schema 1.1* [online]. W3C, 2014 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/TR/rdf-schema/.
6. BAO, Jie; KENDALL, Elisa F.; MCGUINNESS, Deborah L.; PATEL-SCHNEIDER, Peter F. *OWL 2 Web Ontology Language Quick Reference Guide (Second Edition)* [online]. W3C, 2012 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/.
7. *Welcome to Schema.org* [online]. Schema.org Community Group, [2021] [cit. 2021-05-16]. Dostupné z: https://schema.org.
8. *Gmail for Developers Email Markup* [online]. Google, 2020 [cit. 2021-05-16]. Dostupné z: https://developers.google.com/gmail/markup/overview.
9. *JSON for Linking Data* [online]. PaySwarm [cit. 2021-05-16]. Dostupné z: https://json-ld.org/.
10. KELLOGG, Gregg; CHAMPIN, Pierre-Antoine; LONGLEY, Dave. *JSON-LD 1.1* [online]. W3C, 2020 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/TR/json-ld11/.
11. KELLOGG, Gregg; LONGLEY, Dave; CHAMPIN, Pierre-Antoine. *JSON-LD 1.1 Processing Algorithms and API* [online]. W3C, 2020 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/TR/json-ld11-api/.
12. KNUBLAUCH, Holger; KONTOKOSTAS, Dimitris. *Shapes Constraint Language (SHACL)* [online]. W3C, 2017 [cit. 2021-05-16]. Dostupné z: https://www.w3.org/TR/shacl/.
13. *RDForm* [online]. GitHub, c2020 [cit. 2020-11-18]. Dostupné z: https://github.com/simeonackermann/RDForm.
14. *Implement SHACL as template language* [online]. GitHub, c2020 [cit. 2020-11-18]. Dostupné z: https://github.com/simeonackermann/RDForm/issues/18.

15. *React-jsonschema-form* [online]. Read the Docs, [2020] [cit. 2020-11-11]. Dostupné z: https://react-jsonschema-form.readthedocs.io/en/latest/.

16. *Shacl-form* [online]. GitHub, c2020 [cit. 2020-12-01]. Dostupné z: https://github.com/CSIRO-enviro-informatics/shacl-form.

17. KNUBLAUCH, Holger. *DASH Constraint Components* [online]. TopQuadrant, c2020 [cit. 2020-12-02]. Dostupné z: http://datashapes.org/constraints.html.

18. KNUBLAUCH, Holger. *Form Generation using SHACL and DASH* [online]. TopQuadrant, c2020 [cit. 2020-12-02]. Dostupné z: http://datashapes.org/forms.html.

19. *SForms* [online]. GitHub, c2021 [cit. 2021-01-05]. Dostupné z: https://github.com/kbss-cvut/s-forms.

20. *Otevřené formální normy (OFN)* [online]. Portál otevřených dat, [2020] [cit. 2021-05-21]. Dostupné z: https://data.gov.cz/ofn/.

21. NIELSEN, Jakob. Website Response Times. *Nielsen Norman Group* [online]. 20. 6. 2010 [cit. 2021-05-21]. Dostupné z: https://www.nngroup.com/articles/website-response-times/.

22. KIRKPATRICK, Andrew; CONNOR, Joshue O; CAMPBELL, Alastair; COOPER, Michael. *Web Content Accessibility Guidelines (WCAG) 2.1* [online]. W3C, 2018 [cit. 2021-05-18]. Dostupné z: https://www.w3.org/TR/WCAG21/.

23. ATKINS, Tab; FANTASAI. *CSS Values and Units Module Level 3* [online]. W3C, 2019 [cit. 2021-05-18]. Dostupné z: https://www.w3.org/TR/css-values-3/.

Přílohy

A Testovací scénáře

A.1 Scénář 1

1. Přihlaste se do formulářové aplikace:
   `https://kbss.felk.cvut.cz/ofn-record-manager/`

2. Otevřete následující webovou adresu a vyčkejte na zobrazení formuláře:

   ```
   https://kbss.felk.cvut.cz/ofn-record-manager/records/create
   ?formTemplate=
   https://slovn%C3%ADk.gov.cz/datov%C3%BD/
   turistick%C3%A9-c%C3%ADle/form-template
   ```

3. Vyplňte ve formuláři příklad jednoduchého turistického cíle, jehož popis najdete na této adrese:

   ```
   https://ofn.gov.cz/turistick%C3%A9-c%C3%ADle/2020-07-01/
   #p%C5%99%C3%ADklady-jednoduch%C3%BD-turistick%C3%BD-c%C3%ADl
   ```

   (a) Změřte čas vyplnění dat do formuláře

   (b) Zaznamenejte si, které položky jste nedokázali vyplnit

4. Doplňte ve formuláři příklad turistického cíle o další položky, které najdete popsané na této adrese:

   ```
   https://ofn.gov.cz/turistick%C3%A9-c%C3%ADle/2020-07-01/
   #p%C5%99%C3%ADklady-komplexn%C3%AD-turistick%C3%BD-c%C3%ADl
   ```

   (a) Změřte čas vyplnění dat do formuláře

   (b) Zaznamenejte si, které položky jste nedokázali vyplnit

A.2 Scénář 2

Ohodnoťte následující formulářové prvky a posuďte jestli byla práce s nimi intuitivní.

1. Identifikátor sekce
2. Přepínač zobrazit více
3. Zobrazení hodnoty s jednotkou
4. Zobrazení otázky na základě typu
5. Komponenta pro vyplnění celého jména

B Výsledky testování

B.1 Výsledky scénáře 1

1. Jaký byl čas vyplnění kroku 3 a které položky nešlo vyplnit?
   (a) 4m 40s, všechny položky šly vyplnit
2. Jaký byl čas vyplnění kroku 4 a které položky nešlo vyplnit?
   (a) více než 10 minut
   (b) nešlo vyplnit hodně věcí
3. Popište problémy, které jste měli s pochopením zadání?
   (a) žádné
4. Popište ostatní nedostatky formulářů (např. vzhled).
   (a) příliš vnořených sekcí, několikrát jsem se ztrácel
   (b) v sekci Má bezbariérový přístup je hodně polí za sebou, bylo by lepší rozdělit do sekcí
   (c) nějaký čas trvalo najít tlačítko zobrazit více
5. Máte nějaký nápad na vylepšení?
   (a) v některých sekcích (např. Název) seskupovat prvky ne pomocí sekcí, ale umístit je například vedle sebe nebo je barevně odlišit, případně přidat rámeček
6. Další poznámky
   (a) všechno bylo pochopitelné, rychle jsem se zorientoval a vyplnil všechna požadovaná pole, design je dobrý a je pohodlné s tím pracovat
   (b) nějaký čas mi zabralo najít tlačítko zobrazit více a kvůli velkému počtu vnořených sekcí nešlo formulář rychle vyplnit, možná by bylo lepší seřadit pole podle priorit

B.2 Výsledky scénáře 2

1. Identifikátor sekce
   (a) uživatelé neměli námitky a komponentu chválili
2. Přepínač zobrazit více
   (a) tlačítko jsem hledal dlouho
   (b) hodně problému pomohl, stejně je ale těžké najít hledanou otázku, možná přidat nějaké vyhledávání
3. Vícenásobnost sekcí
   (a) na první pohled matoucí, po interakci se dá lehce pochopit
4. Zobrazení hodnoty s jednotkou
   (a) pochopitelné a uživatelsky přívětivé
5. Zobrazení otázky na základě typu
   (a) o hodně lepší a intuitivnější
   (b) šipka vpravo evokuje single-select, možná změnit na "+"
6. Komponenta pro vyplnění celého jména
   (a) komponenta je intuitivní, možná by bylo lepší zvýraznit Celé jméno nebo jinak upravit design
   (b) komponenta ne vždy vyplňuje správně, což může vést ke špatně vyplněným datům, pokud si chyby uživatel nevšimne

C Repozitář s-forms-smart-components

Repozitář se zdrojovým kódem knihovny se nachází na adrese `https://gitlab.fel.cvut.cz/holubvo3/sforms-smart-components`

D Změny v knihovně SForms

Změny byly do repozitáře SForms zahrnuty pomocí několika pull requestů. Následující odkaz tyto pull requesty zobrazí.
`https://github.com/kbss-cvut/s-forms/pulls?q=is%3Apr+author%3Aholubv+created%3A%3C2021-05-21`

E Návod na instalaci s-forms-smart-components

1. Nainstalujte program Node.js[1] verze 14.
2. Přiložený zdrojový kód rozbalte, nebo naklonujte GIT repozitář `https://gitlab.fel.cvut.cz/holubvo3/sforms-smart-components`.
3. Přejděte do složky se zdrojovým kódem.
4. Pomocí příkazu `npm install` nainstalujte veškeré závislosti.
5. Příkazem `npm run dev` spustíte ukázkovou aplikaci.
6. V prohlížeči přejděte na adresu `localhost:8080`.

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