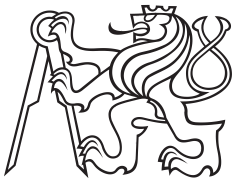
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Mobilní aplikace pro sběr formulářových dat v terénu

Bc. Dominik Prokš

Vedoucí: Mgr. Miroslav Blaško, Ph.D.
Obor: Otevřená informatika
Studijní program: Softwarové inženýrství
Květen 2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Prokš** Jméno: **Dominik** Osobní číslo: **483840**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Mobilní aplikace pro sběr formulářových dat v terénu

Název diplomové práce anglicky:

Mobile application for in situ data collection

Pokyny pro vypracování:

Cílem práce je vytvořit mobilní aplikaci v Javascriptu pro sběr formulářových dat v rámci terénního průzkumu. Umožní identifikovat dotčený objekt pomocí aktuálních souřadnic GPS. Odpovědi na otázky formulářů mohou být doplňovány fotografiemi pořízenými integrovaným fotoaparátem. Pro sběr dat bude aplikace využívat knihovnu pro definici interaktivních formulářů – SForms [1]. Aplikace umožní sběr dat dle různých typů formulářů, které budou staženy ze simulované serverové aplikace. Formuláře bude možné vyplnit i v offline režimu. Aplikace bude validována na scénáři sběru dat pro dokumentaci budov projektu MONDIS [2,3,4].
Postup:
Seznamte s technologiemi Sémantického webu pro reprezentaci (OWL, RDF, JSON-LD) a dotazovaní znalostí (SPARQL)
Analyzujte aplikace pro sběr formulářových dat a definujte požadavky a scénáře pro terénní průzkum, včetně základních scénářů aplikace MONDIS
Navrhněte a implementujte prototyp aplikace pro terénní průzkum včetně komponent pro vybrané scénáře
Implementovaný prototyp otestujte na vybraných scénářích pomocí tří uživatelů

Seznam doporučené literatury:

[1] Sforms, online na https://github.com/kbss-cvut/s-forms
[2] Project MONDIS, online na http://www.mondis.cz/
[3] MONDIS mobile application, online na http://www.mondis.cz/web/portal/mobile
[4] Cacciotti, R.; Valach, J.; Kuneš, P.; Čerňanský, M.; Blaško, M.; Křemen, P. Introduction to an Ontology-Driven Documentation System of Damages to Cultural Heritage.
[5] Manola, Frank, Eric Miller, and Brian McBride. "RDF primer." W3C recommendation 10.1-107 (2004): 6.

Jméno a pracoviště vedoucí(ho) diplomové práce:

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

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

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

Platnost zadání diplomové 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)

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

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

. _____ _____

Datum převzetí zadání Podpis studenta

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Poděkování patří hlavně panu Mgr. Miroslavu Blaškovi, Ph.D. za to, že mi během projektu byl přístupný a ochotný s čímkoliv pomoct.

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, 25. května 2023

Abstrakt

Tato práce se zabývá analýzou a návrhem prototypu mobilní aplikace pro sběr formulářových dat. Aplikace byla navržena tak, aby uživatelům umožnila sbírat a zaznamenávat údaje o různých entitách v jejich přirozeném prostředí, jako jsou budovy, stavby a krajina. Jádro aplikace tkví ve vykreslení formulářů založených na Sémantickém webu, přesněji OWL. Uživatelé této aplikace mohou formuláře doplnit o různé multimediální soubory jako jsou obrázky nebo audio nahrávky.

Klíčová slova: OWL, RDF, mobilní aplikace, SForms, sběr dat v terénu

Vedoucí: Mgr. Miroslav Blaško, Ph.D.

Abstract

This thesis deals with the analysis and design of a prototype mobile application for form data collection. The application was designed to allow users to collect and record data about different entities in their natural environment such as buildings, structures and landscapes. The core of the application lies in the rendering of forms based on the Semantic Web, more specifically OWL. Users of this application can supplement the forms with various multimedia files such as images or audio recordings.

Keywords: OWL, RDF, mobile application, SForms, in-situ data collection

Title translation: Mobile application for in situ data collection

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

1.1 Úvod do problému

Sběr dat v terénu je účinná metoda sběru informací, která zahrnuje sběr dat v přirozeném prostředí nebo v situaci, kdy se přirozeně vyskytují. Tento proces se obvykle provádí pomocí různých technologií, jako jsou mobilní zařízení, GPS, kamery a další. Tento přístup ke sběru dat může poskytnout přesné a autentické údaje. Sběr dat v terénu se hojně využívá v oborech, jako je zdravotnictví, environmentální studie a sociální vědy. Navrhování a implementace aplikací pro sběr dat v terénu však může být náročné, zejména v kontextu mobilních zařízení.

MONDIS je jedna z aplikací, která měla za cíl sběr dát v terénu. Její použití však spadalo pouze do domény stavebnictví a památkové péče. Aplikace sloužila k hodnocení poškození budov a památek. Aktuálně je aplikace již zastaralá a je třeba ji přetvořit do nových technologií a obohatit o nové funkcionality. Hlavním nedostatkem, který je potřeba odstranit, je nemožnost využití aplikace k řešení/vyplňování generických formulářů. Na základě aplikace MONDIS se zaměříme na vývoj aplikace, která uživatelům umožní sbírat data v terénu pomocí mobilního zařízení, i když jsou offline. K vytvoření modelu reprezentace znalostí, který umožní standardizaci a interoperabilitu dat, využijeme jazyky OWL a RDF. Práce se bude zabývat několika oblastmi, včetně návrhu a vývoje funkce sběru dat v terénu, testování a vyhodnocení aplikace a posouzení uživatelské zkušenosti. Výsledná aplikace by měla být užitečným nástrojem pro výzkumné pracovníky, odborníky a všechny ostatní, kteří potřebují sbírat data v terénu. Použití jazyků OWL a RDF zajistí,

že shromážděná data budou standardizovaná a interoperabilní, což usnadní integraci a analýzu dat.

Cílem této diplomové práce je navrhnout a implementovat mobilní aplikaci pro sběr dat v offline režimu. Mobilní aplikace bude pracovat s knihovnou SForms. SForms je knihovna založená na ontologii a slouží k vykreslení sémantických formulářů, které staví na technologiích OWL a RDF. OWL (Web Ontology Language) a RDF (Resource Description Framework) jsou sémantické webové technologie, které poskytují standardizovaný způsob reprezentace a sdílení znalostí. Pomocí OWL a RDF můžeme vytvořit společný slovník pro reprezentaci dat shromažďovaných naší aplikací. To může usnadnit sdílení a integraci dat, což umožní získat z dat významnější poznatky.

Kapitola 2

Technologické pozadí

2.1 Sémantický web

Sémantický web je vizí a rozšířením stávajícího World Wide Webu, kde informace dostávají přesně definovaný význam a strukturu, což umožňuje počítačům a lidem efektivněji spolupracovat. Hlavní myšlenkou sémantického webu je učinit data na webu strojově čitelnými, což umožní lepší porozumění, interoperabilitu a integraci mezi různými aplikacemi, službami a zdroji dat.[1]

2.1.1 RDF

Resource Description Framework je webový standard vytvořený organizací W3C. Hlavním důvodem jeho vytvoření byla výměna mezi organizacemi dat na webu. Datový model RDF se skládá z takzvaných trojic. Trojice lze rozčlenit na subjekt, predikát a objekt. Dejme si za příklad následující ukázku: „(Auto) (má barvu) (červenou)“. Z této ukázky lze vyjádřit subjekt (auto), predikát (má bravu) a také objekt (červenou). Tímto jednoduchým schématem se řídí všechna data v RDF napsaná. Mimo jiné má RDF speciální vlastnost, která umožňuje spojování dat, i když si základní schéma dat neodpovídají. Díky této vlastnosti není třeba upravovat všechny datové konzumenty.[3]

[1] `https://www.ontotext.com/knowledgehub/fundamentals/what-is-the-semantic-web/`

2.1.2 JSON-LD

JSON-LD je jednou z mnoha existujících serializací propojených dat definovaných v abstraktním jazyce RDF. Propojená data[2] definují principy, jakými propojovat data napříč webovými stránkami nebo dokumenty. JSON-LD je jednoduchou syntaxí pro serializaci RDF dat v datové reprezentaci JSON, jehož design nám dovoluje interpretovat existující JSON dokumenty jako propojená data s minimálními změnami. JSON-LD obsahuje všechny vlastnosti JSON a navíc přidává své vlastní. Mezi přidané vlastnosti patří:

- univerzální identifikátor pro JSON objekty pomocí IRI
- způsob jakým rozlišit klíče napříč různými JSON dokumenty pomocí mapování na IRI za pomoci kontextu
- mechanismus jakým hodnota JSON objektu může odkazovat na zdroj na jiné webové stránce
- schopnost anotovat řetězce v jejich jazyce
- způsob jakým asociovat datové typy s jejich hodnotami (jakožto datum a čas)
- způsob vyjádření jednoho či více orientovaných grafů, jakožto např. sociální síť, v jednom dokumentu

Syntax JSON-LD je vytvořen tak, aby nenarušoval již nasazené systémy pracující s formátem JSON, ale poskytoval hladký přechod z JSON na JSON-LD. Jelikož forma dat uschovaných ve formátu JSON se může značně lišit, JSON-LD disponuje mechanismy ke změně dokumentu do deterministické struktury, která zjednodušuje jejich zpracování.

JSON-LD se skládá ze dvou základních částí, kterými jsou context a graf. Context obsahuje mapování termínů[3] na IRI. Jinými slovy by se dalo říci, že slouží ke zjednodušení a zpřehlednění grafové části JSON-LD. Graf reprezentuje uzly a jejich vztahy, které jsou spojené orientovanými hranami. Uzly mohou být reprezentovány jako subjekty, objekty a vlastnosti, zatímco orientované hrany ukazují směr vztahu mezi uzly. Každý uzel musí mít identifikátor v podobě IRI.[4]

JSON-LD poskytuje několik stylů serializace dat. Jimi jsou:

[2] `https://www.w3.org/standards/semanticweb/data`
[3] `https://www.w3.org/TR/json-ld11/#terms`

- Expandovaná (Expanded)
  - Tento druh serializace odstraňuje prvek context.
  - Všechny termíny z contextu a zkrácené verze IRI jsou expandovaný na normální IRI.
  - Všechny hodnoty jsou vyjádřeny v podobě pole.
  - Ukázka JSON-LD souboru v expandované serializaci lze nalézt na ukázce kódu 1.
- Kompaktní (Compacted)
  - Na rozdíl od expandované verze obsahuje prvek context.
  - Je jednodušší na čtení a porozumění lidmi.
  - Zkracuje IRI na termíny a zkrácené verze IRI.
  - Ukázka JSON-LD souboru v kompaktní serializaci lze nalézt na ukázce kódu 2.
- Zploštělá (Flattened)
  - Stejně jako expandovaná verze obsahuje prvek context.
  - Zajišťuje deterministickou podobu dat.
  - Zploštělá verze se stará o to, aby všechny vlastnosti uzlu byly v jednom uzlovém objektu a prázdné uzly byly označeném identifikátorem prázdného uzlu.
  - Ukázka JSON-LD souboru v kompaktní serializaci lze nalézt na ukázce kódu 3.

2.1.3 SForms

Tato sekce analyzuje knihovnu SForms, která je vytvořena KBSS ČVUT[4]. Knihovna je využívána k vykreslování chytrých formulářů založených na ontologii. Knihovna přijímá a zpracovává formuláře v podobě JSON-LD, které jsou definovány pomocí RDF. Knihovna je koncipována na modelu[5] otázka-odpověď. Daný model, který knihovna používá, dovoluje přiřadit k prvku typu Question další neomezený počet prvků Question. Každý prvek Question může obsahovat pouze jednu odpověď.

[4] `https://kbss.felk.cvut.cz/`
[5] `https://kbss.felk.cvut.cz/gitblit/summary/s-forms-model.git`

Aktuální stav

Aktuálně SForms podporují formulářové komponenty popsané v tabulce 2.1. Dále je možné formulář rozdělit na 2 druhy. Prvním druhem je klasický formulář. Druhým je formulář typu Wizard. Formulář typu Wizard se skládá z kroků (záložek). Tyto záložky lze postupně přepínat pomocí navigace, kterou je možné vykreslit buď horizontálně nebo vertikálně.

Komponenta	**Poznámka**
Input	Input je konvertován na komponentu Textarea v případě, že obsahuje více než 50 znaků.
Textarea	Jedná se o víceřádkový textový vstup.
SPARQL field	Tato komponenta obsahuje syntaktický analyzátor (parser) pro SPARQL[6] dotazy. Funguje uvnitř komponenty Textarea.
TURTLE field	Tato komponenta obsahuje parser pro TURTLE[7] notaci. Funguje uvnitř komponenty Textarea.
Typeahead	Komponenta připomínající dropdown. Po kliknutí je rozbaleno menu s možnostmi. Uživatel může vybrat 1 možnost.
Datetime picker	Tato komponenta umožňuje uživateli výběr času, data nebo obou dohromady z kalendáře.
Checkbox	Checkbox představuje pouze 2 hodnoty (true, false).
Masked input	Jedná se o komponentu, která je podobná komponentě Input. Tato komponenta přidá komponentě Input logiku v podobě formátu odpovědi. Uživatel vidí v jakém formátu má být odpověď.
Media content	Komponenta vykresluje HTML iframe[8] pro URL specifikovanou ve formuláři.
Section	Komponenta obalující skupinu otázek. Může být sbalena nebo rozbalena.
Wizard step	Záložka ve formuláři typu Wizard.

Tabulka 2.1: Formulářové komponenty knihovny SForms[2]

Formulářové komponenty dále disponují sekundárními vlastnostmi, které ovlivňují jejich chování. Mezi tyto vlastnosti lze zařadit například následující.

- Hidden
  - Tato vlastnost má za následek, zda je komponenta ve výchozím stavu viditelná.
- Collapsed
  - Tato vlastnost se týká hlavně komponenty Section, u které řídí výchozí stav rozbalení/sbalení.
- Disabled
  - Tato vlastnost udává otázce zda je možné na otázku odpovědět či nikoliv.

Reprezentace formuláře

SForms pracuje s formuláři reprezentovanými pomocí JSON-LD formátu. Knihovna po načtení formuláře provede zploštění (flattening) formuláře bez specifikovaného contextu. Což má za následek transformaci vlastností uzlů na absolutní URI. Dále se celý graf změní na pole grafových uzlů.

Po procesu zploštění transformují SForms formulář do vlastní stromové reprezentace. Uzlům jsou ponechány vlastnosti v podobě absolutních URI. Každý uzel může být jedním z několika typů, kterými jsou například Question, Answer, Option a Condition.

Každý formulář, který je vykreslen knihovnou SForms, musí obsahovat uzel Question, který označuje kořen. Toho je docíleno tak, že má daný uzel vlastnost **has-layout-class** o hodnotě „form“. Dále je nutné aby formulář neobsahoval cykly, což vyplývá z faktu, že SForms každý formulář transformují do stromové struktury.

Vlastnosti uzlů

V této sekci si probereme jakých vlastností můžou nabývat základní typy uzlů Question a Answer. Každý uzel musí obsahovat minimálně vlastnosti

@id a **@type**. Vlastnost **@type** určuje o jaký typ uzlu se jedná a vlastnost **@id** je unikátní identifikátor uzlu pomocí absolutní URI.

Question

Uzel typu Question může obsahovat širokou škálu vlastností, ale 4 vlastnosti jsou kritické a musejí být pokaždé přítomny. Těmito vlastnostmi jsou **@id**, **@type**, **label** a **has-layout-class**. Vlastnost **label** určuje textový popis otázky a vlastnost **has-layout-class** určuje formulářovou komponentu, která má být použita pro odpověď na danou otázku.

Další vlastnosti, kterých může uzel typu Question nabývat jsou popsány v tabulce 2.2. Nejedná se o všechny možné vlastnosti.

Vlastnost	**Popis**
comment	Komentář otázky
has_related_question	Pole podotázek, které je reprezentováno textovými řetězci @id otázek.
has_answer	@id uzlu, který představuje odpověď na danou otázku.
has-preceding-question	@id uzlu, který má být ve formuláři vykreslen před danou otázkou.
has-question-origin	Identifikace uzlu v propojených datech. Slouží k ukotvení významu otázky, kde význam otázky je popsán v ontologii.
description	Textový řetězec určený pro nápovědu otázky.

Tabulka 2.2: Doplňkové vlastnosti uzlu Question

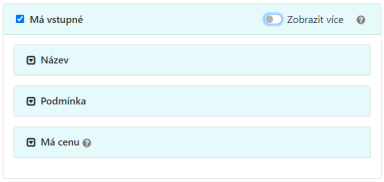
Answer

Tento typ uzlu obsahuje mimo vlastností **@id** a **@type** také vlastnosti **has_data_value**, **has_object_value** a **has-answer-origin**. Vlastnost **has_object_value** obsahuje odpověď na otázku v podobě RDF zdroje. Vlastnost **has_data_value** obsahuje odpověď v podobě RDF literálu.

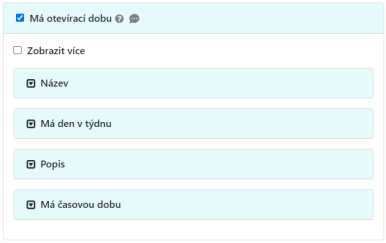
2.1.4 SForms Components

SForms Components[9] je knihovna rozšiřující základní knihovnu SForms o nové komponenty a funkcionality. Knihovna rozšiřuje základní komponenty SForms pomocí dědičnosti a kompozice, což umožňuje upravit pouze potřebné metody pro změnu vzhledu a chování komponenty, aniž by bylo nutné kopírovat základní funkcionality z SForms. Mezi změny, které přinášejí SForms Components, patří například přesunutí přepínače zobrazení rozšiřujících otázek do komponenty Section. Vzhled a chování upravené komponenty Section oproti původnímu vzhledu v SForms je k vidění na obrázku 2.1.

Pro úplnou integraci knihovny SForms Components do SForms bylo nezbytné vytvořit v původní knihovně mapovací funkci, která pro danou otázku vrátí odpovídající komponentu k vykreslení. Tato funkce má v rámci knihovny SForms své vlastní chování, což je důležité pro udržení výchozí funkcionality. Současně je však volána i další mapovací funkce, kterou SForms přijímá jako jeden ze svých vstupních parametrů. Pokud SForms nedostane žádnou takovou funkci, zachovává svou původní funkcionalitu. Pokud však takovou funkci obdrží, zkontroluje, zda pro danou otázku existuje odpovídající mapovací pravidlo. Pokud existuje, je volána komponenta specifikovaná v tomto pravidle. Pokud pravidlo neexistuje, je volána výchozí komponenta z SForms.



(a) : SForms komponenta Section



(b) : SForms Components komponenta Section

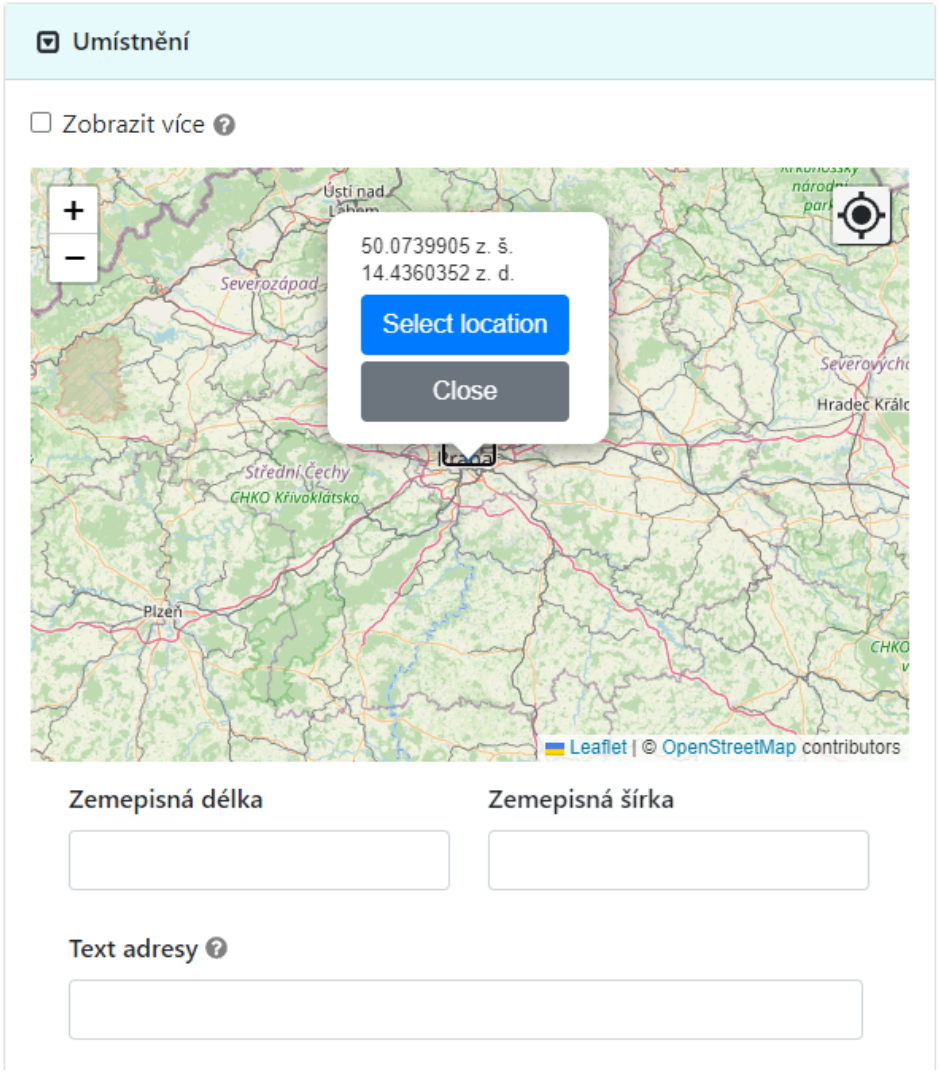
Obrázek 2.1: Ukázka rozdílu mezi komponentou Section

2.1.5 SForms Geocomponent

SForms Geocomponent[10] je další knihovnou obohacující základní funkcionality SForms. Tato knihovna byla vytvořena za účelem obohacení SForms o mapovou komponentu. Knihovna zároveň pracuje s Geolocation API[11],

[9] https://github.com/kbss-cvut/s-forms-components
[10] https://github.com/VojtechLunak/s-forms-geo-components
[11] https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

čímž uživatelům umožňuje přístup k polohovým službám zařízení. Jak jsem již zmínil, knihovna nabízí vykreslení mapy, se kterou má uživatel možnost interagovat. Interakce mapy dovoluje zvolení bodu na mapě nebo nalezení aktuální polohy zařízení za pomoci služby GPS. Uživatelem vybraný bod na mapě je možné použít jakožto odpověď na patřičné otázky, které souvisí se zeměpisnou šířkou a délkou. Jak tato komponenta vypadá je k nalezení na obrázku 2.2.



Obrázek 2.2: SForms Geocomponent

2.2 Moderní webová API

Moderní webová API jsou klíčová pro vývoj offline aplikací, které umožňují efektivní sběr dat v terénu a zjednodušují práci s aplikací i při nestabilním nebo nedostupném internetovém připojení.

V současné době se web stále více stává pokročilým a interaktivním prostředím, které nabízí bohaté zážitky pro uživatele. Moderní webová API hrají klíčovou roli v tom, jak se vývojáři mohou přizpůsobit těmto novým požadavkům a vytvářet aplikace, které fungují na různých platformách a zařízeních. Tato kapitola se zaměřuje na některá z těchto moderních API, která umožňují přístup k integrovaným databázím moderních prohlížečů, jako je IndexedDB, a k přístupu k nativním službám zařízení, jako je GPS. Také se podíváme na vytváření offline aplikací, které mohou poskytovat užitečné funkce i bez připojení k internetu.

Jedním z klíčových aspektů moderního webu je schopnost ukládat a spravovat data na straně klienta, což umožňuje rychlé a plynulé zážitky pro uživatele. IndexedDB je výkonná databáze, která běží přímo v prohlížeči a umožňuje ukládat strukturovaná data ve velkém měřítku. S využitím IndexedDB mohou vývojáři vytvářet sofistikované webové aplikace, které pracují s daty i v offline režimu.

Kromě toho moderní webová API umožňují přístup k nativním službám zařízení, jako je GPS, což rozšiřuje možnosti webových aplikací a poskytuje uživatelům lokalizační služby, které dříve byly vyhrazeny pouze pro nativní aplikace. Tímto způsobem se web stává stále více konkurenceschopným prostředím pro vývoj mobilních aplikací.

Offline aplikace jsou další důležitou součástí moderního webu. Díky API, jako je Service worker, mohou vývojáři vytvářet aplikace, které fungují bez připojení k internetu a poskytují uživatelům konzistentní zážitek i v případě, že nemají přístup k síti. Tento přístup je zvláště důležitý pro uživatele v oblastech se slabým nebo nepravidelným pokrytím internetu.

2.2.1 Web Workers API

Webové pracovníky (Web workers) lze použít k provádění kódu na pozadí, a to v novém vlákně, odděleně od hlavního vlákna webové aplikace, které se stará o uživatelské interakce. Tímto způsobem je možné dosáhnout nepřerušovaného běhu hlavního vlákna a zajištění lepší responsivity webové aplikace. Web workers jsou primárně určeni pro výpočetně náročné operace, které by v běžných podmínkách zatěžovaly hlavní vlákno a mohly by tak způsobovat neresponsivitu aplikace. Jinými slovy, Web workers představují způsob, jak zlepšit uživatelskou zkušenost s aplikací.

Web workers dokáží spustit pomalu jakýkoliv JavaScriptový kód. Výjimku však tvoří manipulace s DOM a přístup k metodám a vlastnostem objektu window. Web workers tedy nemají přímou možnost ovládání webové stránky. Mohou ovšem s DOM (hlavním vláknem) komunikovat pomocí zpráv. Každý Web worker může naslouchat události *onmessage*. Tato událost obsahuje element *message*, který má vlastnost *data*. Pomocí těchto zpráv jsou Web workers schopni komunikovat s hlavním vláknem a naopak.[5]

Web workers existuje několik typů. Pro naše účely je však důležitý hlavně Service worker.

Service worker

Service worker je virtuální proxy mezi prohlížečem a sítí. Jeho hlavním použitím je cachování požadavků a souborů. Jelikož zachytává síťové požadavky, a vrací již uložené (cachované) odpovědi. Zároveň může zachycené požadavky a odpovědi upravovat a připravit k dalšímu použití v aplikaci. Jelikož jsou HTTP spojení náchylná na útoky typu Man-in-the-Middle, je nutné využívat Service worker pouze v zabezpečeném kontextu (HTTPS). Důvodem je, že by útočník mohl jednoduše nainstalovat vlastní Service worker a tím pádem ovládat celou komunikaci mezi napadeným prohlížečem a zbytkem světa.[6]

Životní cyklus Service workeru se skládá ze tří částí, kterými jsou následující.

1. Download
    - Service worker se stáhne, když uživatel navštíví poprvé stránku, která obsahuje Service worker nebo je k dispozici nový Service

worker.

2. Install
   - Instalační část se stará o instalaci Service workeru. V této části mohou nastat dva případy. Prvním je, že neexistuje žádný aktivovaný Service worker. V takovém případě je Service worker ihned aktivován. V druhém případě je Service worker pozdržen a dán do stavu *waiting*, kdy čeká na aktivaci.
3. Activate
   - Jak již z názvu napovídá, aktivační část má na starost aktivaci Service workeru. Když je Service worker aktivován, může začít vykonávat práci.[7]

2.2.2 Web Share API

Web Share API[12] je moderní a výkonný nástroj, který poskytuje vývojářům webových aplikací možnost integrovat do svých webových aplikací nativní možnosti sdílení. V této části se budeme zabývat požadavky na implementaci rozhraní Web Share API a také jeho rozsáhlými možnostmi a potenciálním dopadem na uživatelskou zkušenost a zapojení.

Požadavky

Mezi tři klíčové požadavky patří podpora prohlížeče, připojení pomocí HTTPS a aktivace gestem uživatele. Web Share API je podporováno většinou moderních webových prohlížečů včetně Chrome, Edge a Safari. Bohužel toto API aktuálně není podporování v prohlížeči Firefox.

Web Share API vyžaduje, aby připojení webové stránky bylo zabezpečeno pomocí připojení HTTPS, jelikož pracuje s citlivými údaji uživatele. Tím zajišťuje, že data jsou šifrována a tím pádem chráněna před případnými útoky.

Posledním požadavkem je aktivace gestem. Tento požadavek značí souhlas uživatele sdílet data a předchází nežádoucímu sdílení dat bez uživatelova vědomí.[8]

[12] `https://developer.mozilla.org/en-US/docs/Web/API/Web_Share_API`

Přínosy

Mezi hlavními přínosy tohoto API pro uživatele lze zařadit nativní sdílení, které může vyústit ve větší zapojení uživatelů. Zároveň i pro vývojáře existuje výhoda využití tohoto API a tou je flexibilita.

Nativní sdílení umožňuje aplikacím využívat stejné nativní možnosti sdílení jako nativní aplikace, čímž zjednodušuje proces sdílení a poskytuje konzistentnější uživatelskou zkušenost napříč platformami. Tím, že rozhraní API pro sdílení webu nabízí uživatelům známý a pohodlný způsob sdílení obsahu, může vést ke zvýšení zapojení uživatelů, jelikož uživatelé budou sdílet obsah jim již známou cestou.

2.2.3 Geolocation API

Geolocation API[13] je další moderní API, které s sebou přináší možnost využití polohových služeb zařízení. Polohové služby jako zdroje informací využívají globální polohové systémy, jako je například GPS, síťové signály (IP adresa, RFID, WiFi a Bluetooth MAC adresa) a ID buněk GSM/CDMA.

Požadavky

Pro využití Geolocation API je potřeba splnit dva hlavní požadavky. Těmito požadavky jsou zabezpečený kontext (HTTPS) a souhlas uživatele s využitím polohových služeb.

Offline použití

Bez připojení k internetu lze Geolocation API použít, ale jeho schopnost určit polohu může být omezena nebo vysoce nepřesná. Také záleží na zařízení a jeho schopnostech určování polohy. Mobilní zařízení mohou primárně využívat systém GPS, ačkoli se může stát, že tento systém v dané lokaci není přístupný (např. uvnitř budov) nebo zařízení samotný GPS modul neobsahuje.

[13] `https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API`

V takových případech se může Geolocation API pokusit o nalezení polohy za pomoci WiFi sítě, což může vyžadovat přístup k internetové síti. [9]

2.2.4 Notifications API

Notifications API[14] je další z rodiny moderních API webových prohlížečů. Jedná se o zásadní komponentu moderních webových aplikací, která umožňuje vývojářům zobrazovat uživatelům nativní systémová oznámení (notifikace) jako součást uživatelské zkušenosti. Dle dokumentace je toto API možné používat z Web workerů, mezi které se řadí i Service worker. Tím pádem je možné vytvářet notifikace na pozadí bez nutnosti mít spuštěnou aplikaci. Ovšem stejně jako u předchozích moderních API, i zde je potřeba využití zabezpečeného kontextu, což znamená využití HTTPS.

Před možností využití tohoto API je třeba, aby uživatel udělil souhlas s jeho využitím. Vývojář tedy musí vytvořit možnost udělení souhlasu od uživatele. Proces udělení souhlasu by měl obsahovat zobrazení výzvy, která vysvětluje účel udělení souhlasu a žádost o souhlas.[10]

Notifications API umožňuje vývojářům vytvářet notifikace s vlastním obsahem, čímž zajistí, že notifikace budou relevantní a poutavé. Dále mají vývojáři možnost správy uživatelské interakce s notifikacemi, díky čemuž mají možnost nastavit vlastní akce, které mají nastat například při kliknutí na notifikaci.[11]

2.2.5 Push API

Push API[15] je další moderní API webových prohlížečů, které dovoluje přijímání zpráv push ze serveru i když je webová aplikace v pozadí nebo dokonce i není zapnuta ve webovém prohlížeči. Tato funkcionalita umožňuje doručení zpráv (novinek) v reálném čase.

[14] `https://developer.mozilla.org/en-US/docs/Web/API/Notification`
[15] `https://developer.mozilla.org/en-US/docs/Web/API/Push_API`

Požadavky

Podobně jako u Notifications API je i zde nutné vyžádat oprávnění od uživatele. Dalším požadavkem je aktivní Service worker, jelikož bez jeho využití by nešlo registrovat zařízení do služby zajišťující push notifikace. Dále by aplikace neměla jak odchytávat push notifikace na pozadí. Dle dokumentace je dále extrémně důležité v aplikaci vytvořit ochranu před CSRF/XSRF útoky.[12]

Služba push notifikací je taková služba, která se stará o doručování zpráv push do zařízení. Jedná se o prostředníka mezi aplikačním serverem a klientskou (webovou) aplikací. Pokud aplikační server chce poslat push notifikaci do určitého zařízení, je třeba poslat požadavek na službu push notifikací, která se následně o požadavek postará a pošle zprávu push do daného zařízení.

Životní cyklus

Životní cyklus Push API zahrnuje následující fáze:

1. Registrace zařízení
   - Webová aplikace vytvoří požadavek na službu push notifikací o registraci zařízení pod unikátním ID. ID lze brát jako unikátní URL koncového bodu.
2. Řízení odběru
   - Webová aplikace se stará o odběr zpráv push ze služby push notifikací. Odběr zahrnuje ID zařízení a šifrovací klíče pro bezpečnou komunikaci mezi webovou aplikací a službou push notifikací.
3. Posílání zpráv
   - Když má server doručit nový obsah, odešle službě push notifikací zprávu s jedinečnou adresou URL koncového bodu a obsahem push notifikace. Obsah obvykle zahrnuje název oznámení, text těla notifikace, ikonu a další údaje relevantní pro obsah.
4. Doručení zprávy
   - Služba push notifikací poté doručí zprávu do zařízení uživatele, i když webová aplikace není právě otevřena.

5. Zpracování zprávy
    - Po přijetí zprávy ji webová aplikace zpracuje a zobrazí uživateli oznámení. Uživatel pak může s oznámením interagovat, což může vyvolat další akce v rámci webové aplikace.[13]

Kapitola 3

Existující řešení

V této kapitole se zaměříme na existující řešení sběru dat a jejich následné analýzy. Mobilních nebo webových aplikací umožňujících sběr formulářových dat je k dispozici poměrně velké množství. Každá takováto aplikace umožňuje vytvoření vlastního formuláře. Všechny níže zmíněné aplikace fungují jako webové i mobilní aplikace.

3.1 MONDIS

MONDIS (zkratka anglického MONument Damage Information System) byl výzkumný projekt zabývající se dokumentací a analýzou poruch kulturního dědictví. Tento projekt je spravován Katedrou kybernetiky Fakulty elektrotechnické ČVUT v Praze a Ústavem teoretické a aplikované mechaniky Akademie věd České republiky. Cílem projektu bylo vytvořit systém, který umožní lépe porozumět vztahům mezi poruchami stavebních památek, jejich materiály a technologiemi, vnějším prostředím a zatížením.[14]

Aplikace MONDIS mobile umožňovala uživatelům vyplnit údaje o stavbě. Tyto údaje se skládali ze 3 částí. Těmito částmi byly:

1. identifikace,
2. lokalizace,
3. posouzení stavu.

3.1.1 Identifikace

Část identifikace umožnila uživateli identifikovat budovu na základě následujících vlastností:

- architektonický styl,
- konstrukční typ,
- funkční typ,
- strukturní typ.

3.1.2 Lokalizace

Uživatel měl na možnost lokalizovat budovu pomocí:

- vyplnění z ÚSKP,
- adresy,
- souřadnic GPS.

3.1.3 Posouzení stavu

Tato sekce umožnila uživateli vyplnění stavu stavby. Toto posouzení vychází z ontologického modelu, který je možné vidět na obrázku A.1.

3.1.4 Funkcionality

Aplikace nabízela několik funkcionalit. Mezi nimi byla možnost duplikovat otázky uvnitř formuláře, což mohl uživatel kdykoli smazat. Tato možnost smazání duplikované otázky byla zavedena k nápravě uživatelských chyb. Uživatelé mohli také formulář doplnit o multimediální soubory.

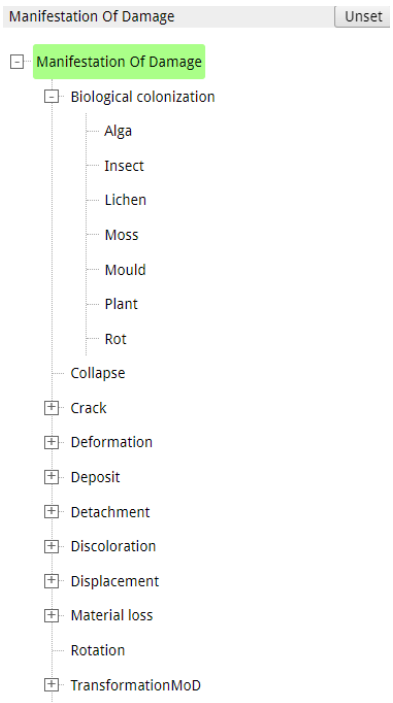
Další klíčovou funkcí bylo lokální ukládání formulářů pro offline použití. Aplikace využívala jak WebSQL databázi, tak IndexedDB. WebSQL je v dnešní době považována za zastaralou a není již používána.

Aplikace umožňovala uživateli vybírat možnosti z tzv. „Intelligent tree select“, což je stromová struktura. Tato funkce je zobrazena na obrázku 3.1. Každá položka formuláře mohla mít přidán textový popis a vlastnosti, které se liší na základě typu položky a její vybrané hodnoty.

Například, pokud uživatel vybral, že prvek fasáda je z materiálu *Fired clay* (pálená cihla), mohl u materiálu nastavit vlastnosti jako *Density* (hustota), *Rebound hardness* (tvrdost odrazu) a *Water absorption* (absorpce vody).

Aplikace také umožňovala vytvoření záznamu formuláře ze šablony. Mezi uloženými (staženými) záznamy mohli uživatelé řadit na základě jména, času stažení a kompletní historie záznamu.

V poslední řadě aplikace byla schopna pracovat s formuláři obsahujícími vnořené sekce.



Obrázek 3.1: Prvek Intelligent tree select

Aplikace jako taková ovšem byla určena hlavně pro sběr dat a analýzu poruch kulturního dědictví. Nebyla určena k obecnému využití pro sběr dat. Tato aplikace, resp. její použití, představuje požadavek této diplomové práce, který bych měl naplnit. To znamená implementovat potřebné funkcionality pro vyplňování formulářů založených na MONDIS. Formuláře založené na MONDIS nemusí obsahovat vlastnosti jednotlivých formulářových položek.

Na základě analýzy byly zjištěny případy užití aplikace MONDIS, které jsou k vidění na diagramu případů užití A.3.

3.2 SurveyMonkey

SurveyMonkey[1] je software pro online průzkumy, který uživatelům umožňuje vytvářet, odesílat a analyzovat průzkumy. SurveyMonkey nabízí různé typy otázek, včetně otázek s výběrem odpovědí, hodnotících škál a otevřených otázek. Tvořené průzkumy mohou nabývat tří podob. Těmito podobami jsou

[1] https://www.surveymonkey.com/

klasická formulářová podoba, „One after another“ a konverzace. Podoba „One after another“ je taková, že se uživateli ukazuje jedna otázka po druhé ve formulářové formě. Konverzační podoba vypadá a funguje jako chat na sociálních službách.[15]

Tvůrce průzkumu má na výběr několik metod vytvoření. První metodou je vytvoření prázdného průzkumu. V další řadě má tvůrce možnost vytvořit průzkum ze šablony nebo importovat starší průzkum.

Tvůrce průzkumu může přizpůsobit širokou škálu nastavení jako je ochrana soukromí a délka průzkumu. Dále SurveyMonkey poskytuje různé možnosti distribuce průzkumů, mezi které patří například sdílení odkazu, poslání e-mailové pozvánky, sdílení na sociální síti a vložení průzkumu na webovou stránku (embedding).

SurveyMonkey shromažďuje a ukládá odpovědi na průzkumy. Nad těmito odpověďmi dále poskytuje možnost analýzy za využití různých nástrojů, jako jsou grafy, tabulky a statistické analýzy. Dále může uživatel data exportovat do různých formátů nebo je případně integrovat do dalších aplikací.[16]

SurveyMonkey umožňuje tvůrcům spolupracovat na průzkumech přidáváním členů týmu, přidělováním úkolů a sdílením přístupu k průzkumům. Tvůrci mohou zkvalitňovat své pracovní postupy nastavením automatických připomenutí, e-mailových oznámení a integrací s dalšími aplikacemi.

SurveyMonkey disponuje mobilní aplikací pro operační systémy iOS a Android. Čímž svým uživatelům umožňuje vyplňování a tvorbu průzkumů z takřka jakéhokoliv zařízení a místa.

SurveyMonkey je volně dostupný software. Bohužel základní (bezplatná) verze obsahuje pouze zlomek funkcionalit a komponent.

SurveyMonkey má svou vlastní mobilní aplikace SurveyMonkey Anywhere, která svým uživatelům dovoluje sběr dat i bez internetového připojení.

3.2.1 Funkcionality

Tvůrce průzkumu může během tvorby nastavit další užitečné vlastnosti, jako je náhodné pořadí odpovědí na otázku. Dalšími možnostmi jsou nastavení

vlastní validace odpovědi na otázku a zda je odpověď na otázku požadována.

SurveyMonkey umožňuje tvůrcům přidat logiku průzkumu. Pod logikou si můžeme představit operace jako je přenesení uživatele na specifickou stránku nebo otázku v průzkumu na základě odpovědi na otázku.

SurveyMonkey také umožňuje anonymní vyplňování průzkumů. Toto nastavení je ve výchozím stavu vypnuté a tvůrce průzkumu tedy musí tuto možnost povolit. Pokud tvůrce povolí anonymní vyplňování pro určitý průzkum, tak SurveyMonkey přestane sbírat data o uživatelích pro tvůrcem zvolený průzkum.

Uživatelé, kteří odpovídají na průzkum, vidí ukazatel průběhu vyplňování. Takže mají představu o tom, jak daleko se ve vyplňování nachází. Toto je velice užitečné při vyplňování průzkumů, dotazníku, které mají více stran.

Jak jsem již zmínil, SurveyMonkey umožňuje exportování dat do souborů. Podporovanými soubory jsou PDF, XLS, CSV, PPTX a SPSS.

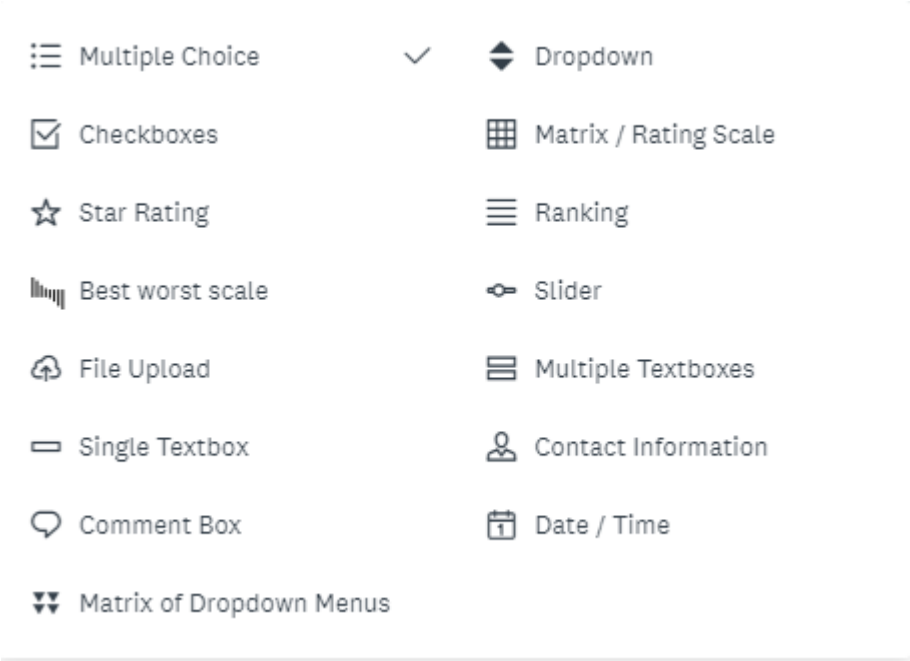
V poslední řadě bych zmínil možnost koupě cílených odpovědí. Tvůrce vybere specifickou skupinu, která má průzkum vyplnit a o distribuci se následně stará SurveyMonkey.

3.2.2 Komponenty

Komponenty dostupné tvůrci jsou k vidění na obrázku 3.2. Mezi volně dostupné komponenty se řadí následující:

- Multiple Choice,
- Checkboxes,
- Dropdown,
- Single Textbox,
- Date / Time.

Zbytek komponent je dostupný pouze v placené verzi. Také si můžeme povšimnout, že SurveyMonkey neumožňuje vytvoření sekcí v průzkumech, tím pádem nelze seskupovat otázky.



Obrázek 3.2: SurveyMonkey - komponenty

3.3 Google Forms

Google Forms je bezplatná online aplikace pro tvorbu formulářů, která je součástí sady Google Drive. Uživatelům umožňuje vytvářet čisté/prázdné formuláře. Formuláře mohou obsahovat obecné typy otázek jako jsou otázky s výběrem odpovědi, hodnotící škály a otevřené otázky. Vytvořené formuláře mohou být v podobě normálního formuláře nebo kvízu. Kvíz s sebou nese další možnosti nastavení. U kvízu lze nastavit například bodové ohodnocení odpovědí na otázky.[17]

Stejně jako SurveyMonkey i Google Forms umožňují distribuci formuláře v podobě odkazu, sdílení na sociálních sítích nebo vložení formuláře na webovou stránku.

Google Forms shromažďují a ukládají všechny odpovědi odeslané uživateli. Vlastníci formuláře mohou odpovězená data prohlížet a za pomoci různých

nástrojů jako jsou grafy a tabulky mohou data analyzovat. Data lze také exportovat jako soubor nebo propojit s dalšími nástroji sady Google Drive.

Spolupráce na formuláři je zde také povolena stejně jako u SurveyMonkey. Tvůrce má možnost přidání spolupracovníků, kteří posléze mohou na formuláři pracovat a zkoumat zodpovězená data. Pro ulehčení práce je zde možnost nastavení e-mailových oznámení.

Google Forms mají svou vlastní mobilní aplikaci dostupnou pro operační systémy Android a iOS. Tudíž uživatelé nejsou vázáni pouze na počítač, ale mohou tvořit a odpovídat pomalu odkudkoliv a z jakéhokoliv zařízení.

3.3.1 Funkcionality

Mezi funkcionality Google Forms dále patří například výchozí nastavení formuláře. Ve výchozím nastavení lze nastavit zda se mají shromažďovat e-mailové adresy uživatelů, kteří odpověděli. Dalším výchozím nastavením je zda jsou všechny otázky povinné či nikoliv.

Google Forms tvůrci nabízí možnost náhodného uspořádání odpovědí na otázku. Dále umožňují vytvoření sekcí, se kterými je úzce spjato vytváření logiky ve formuláři. Tvůrce má možnost vytvořit přechody mezi sekcemi na základě odpovědi. Tvůrce nemůže vytvářet vnořené sekce.

Dále Google Forms umožňují vytvoření vlastní validace pro formulářové odpovědi. V případě textové odpovědi má tvůrce možnost omezit odpověď na minimální/maximální počet znaků nebo dle regulárního výrazu.

Tvůrci mají možnost analýzy shromážděných dat jako celku. V případě potřeby je možné analyzovat jednotlivé odpovědi zvlášť.

V poslední řadě mají uživatelé možnost vidět ukazatel průběhu během vyplňování formuláře a také možnost přeposlání formuláře se svými odpověďmi na svůj e-mail.

3.3.2 Komponenty

Google Forms mají všechny formulářové komponenty volně dostupné. Komponentami jsou

- stručná odpověď,
- odstavec,
- výběr z možností,
- zaškrtávací políčka,
- rozbalovací nabídka,
- nahrání souboru,
- lineární stupnice,
- mřížka výběru z možností,
- mřížka zaškrtávacích políček,
- datum,
- čas.

Stručná odpověď a odstavec jsou textová pole, která se liší v maximální délce povolené odpovědi a vlastnosti odstavce vytvořit více řádků. Nahrání souboru nahrává soubory na disk Google tvůrce formuláře.

3.4 Jotform

Jotform[2] je webová aplikace, která umožňuje uživatelům vytvářet vlastní formuláře. Uživatelé mají k dispozici více než 10 000 šablon nebo si mohou vytvořit čistý formulář, případně importovat již existující formulář.[18]

Tvůrce formuláře má na výběr ze dvou možných stylů formuláře. Jotform tyto styly nazývá klasickým formulářem a karetním formulářem. Karetní

[2] `https://www.jotform.com/`

formulář se v podstatě neliší od SurveyMonkey průzkumu typu One after another.

Jotform umožňují svým uživatelům distribuci formuláře pomocí odkazu, sdílení na sociální síti nebo vložení na webovou stránku. Dále má tvůrce možnost dostávat e-mailové upozornění pokud někdo odpoví na formulář. Bezplatná verze umožňuje pouze jedno oznámení za formulář.

I zde mají tvůrci možnost kolaborace nad formulářem. Stačí pouze, když tvůrce pozve ostatní uživatele ke spolupráci nad formulářem. Tvůrci mají také možnost analýzy zodpovězených dat za pomoci grafů a tabulek.

Jotform ukládá všechny odeslané odpovědi a umožňuje uživatelům spravovat svá data v reálném čase, exportovat je do různých formátů a analyzovat je pomocí různých nástrojů, jako jsou grafy, tabulky a statistická analýza. Uživatelé mohou také spolupracovat na formulářích a automatizovat své pracovní postupy pomocí automatických oznámení a integrací s dalšími aplikacemi.

Stejně jako u předchozích aplikací, i Jotform disponují vlastní mobilní aplikací, která umožňuje uživatelům vytvářet a odpovídat na formuláře. Uživatel má možnost využití mobilní aplikace offline. To znamená, že může vytvářet a vyplňovat formuláře bez internetového připojení. Jakmile se uživatel připojí k internetovému připojení, jsou všechny jeho změny automaticky aktualizovány.

3.4.1 Funkcionality

Jednou z klíčových funkcionalit Jotform je schopnost integrace s řadou populárních služeb. To znamená, že uživatelé mohou snadno přenášet data do Google Drive, zálohovat je v Dropboxu, odesílat upozornění do Slacku nebo zpracovávat platby pomocí PayPalu. Tyto integrace umožňují uživatelům maximalizovat efektivitu a optimalizovat pracovní procesy.

Dále disponuje aplikace Jotform možností nahrávání souborů. Tato funkcionalita je zajištěna pomocí služby Uploadcare[3].

Mobilní notifikace jsou další užitečnou funkcí, kterou Jotform nabízí. Díky

[3] `https://uploadcare.com/`

tomu obdrží uživatelé upozornění na svém mobilním zařízení vždy, když je jejich formulář vyplněn. Tato funkce umožňuje rychlé reagování na nové informace nebo dotazy a zajišťuje, že žádné důležité údaje nezůstanou bez povšimnutí.

Jotform umožňuje vytvoření sekcí ve formuláři. Bohužel tyto sekce jsou pouze ploché a tím pádem není možné vytvořit vnořené sekce.

Posledním významným prvkem, který Jotform nabízí, je možnost vytvoření PDF s formulářem pro manuální vyplnění. Tato funkce umožňuje uživatelům přistupovat k informacím offline a nabízí možnost ručního vyplňování pro ty, kdo upřednostňují tuto formu.

Komponenty

- Short Text — text field
- Long Text — textarea
- Dropdown
- Single Choice — radio button
- Multiple Choice — checkbox
- Number
- Image
- File Upload
- Time
- Spinner — typ komponenty number

3.5 Shrnutí

V kapitole jsme se zabývali srovnáním několika existujících řešení, která se zaměřují na sběr dat a jejich analýzu. Zjistili jsme, že až na MONDIS žádné z těchto řešení nepracuje se sémantickými formuláři. Avšak MONDIS, ačkoli

využívá sémantických formulářů, je bohužel zastaralé řešení, které nenabízí možnost aplikace pro obecné formuláře.

Dále jsme zjistili, že ne všechny aplikace poskytují podporu pro offline režim a mobilní aplikace, což je pro sběr dat v terénu poměrně velkou překážkou. Dále jsme identifikovali funkcionality jednotlivých aplikací a možné způsoby odpovědí na otázky v podobě formulářových komponent. V poslední řadě jsme zjistili, že ne každá ze zmíněných aplikací je zdarma.

Kapitola 4

Výběr technologie

Výběr technologie, použité k vývoji mobilní aplikace, podléhal následujícím požadavkům:

- offline používání,
- jednoduchá instalace,
- využívání rozhraní pro nahrání mediálních záznamů,
- stahování šablon formulářů,
- podpora více druhů formulářů,
- podpora více jazyků.

Na základě těchto požadavků jsem při hledání objevil možnost použít 3 hlavní technologie pro vytvoření multiplatformní mobilní aplikace, které vyhovují stanoveným požadavkům. Dalším kritériem pro výběr těchto technologií byla stále trvající podpora a údržba dané technologie.

1. Progresivní webová aplikace
2. Cordova
3. Flutter

Dále se nabízela možnost vytvořit čistě nativní mobilní aplikaci. Nicméně tuto možnost jsem rovnou zavrhl, jelikož nativní aplikaci lze vytvořit pomocí Flutteru. Flutter se řadí mezi nativní aplikace a zároveň je multiplatformní. Tudíž je pro mne zbytečné vytvářet čistě nativní aplikaci pomocí AndroidSDK a iOS SDK.

4.1 Progresivní webová aplikace

Progresivní webová aplikace je taková webová aplikace, která je obohacena o moderní API, aby svým uživatelům doručila vylepšené možnosti, spolehlivost a instalovatelnost. Cílem PWA je dosažení kohokoliv, kdekoliv za použití jediné codebase. Tím je myšleno vytvoření aplikace pro co nejširší spektrum zařízení za použití jediného kódu.

4.1.1 Pilíře PWA

PWA se opírá o tři pilíře. Těmito pilíři jsou schopnost, spolehlivost a instalovatelnost.

Schopnost

Tento pilíř se opírá o využití moderních API jakožto například WebRTC, push notifications a WebGL.

Využitím těchto API webová aplikace nabírá nový rozměr a má možnost ovládat např. souborový systém či ovládání médií. Nicméně API zařízení se mohou často lišit a samotná zařízení mohou nabízet rozdílné senzory a hardwarové funkce, tím pádem nemusí být zaručeno, že budou k dispozici všechny možné funkcionality.

Spolehlivost

Spolehlivá progresivní webová aplikace musí být rychlá a použitelná nehledě na internetové připojení. Aplikace by měla své uživatele upozornit na neschopnost navázání internetového připojení nebo neschopnosti dokončit serverový požadavek, tím je myšleno, že by aplikace neměla tiše selhávat. Zároveň je důležité, aby aplikace byla instantně responsivní na uživatelské vstupy. Uživatel by neměl čekat například na scrollování v aplikaci nebo na zmáčknutí tlačítka.

Instalovatelnost

Instalované progresivní aplikace běží ve svém vlastním okně a nikoliv uvnitř internetového prohlížeče. Uživatel má možnost tuto aplikaci spustit přímo z domovské obrazovky svého zařízení. Instalace progresivní webové aplikace umožní například využívat aplikaci jakožto výchozí aplikaci pro otevírání určitých druhů souborů nebo přijímat obsah z jiných aplikací instalovaných na zařízení.[19]

Instalace progresivních webových aplikací je třeba udělat pouze jednou. Následné aktualizace se provádějí automaticky samostatně a není třeba uživatelské interakce.

4.1.2 Výhody

- využití známých technologií CSS, HTML a JS
- nižší náklady na vývoj
- obrovská komunita
- skvělá dokumentace
- aktualizace nevyžaduje interakci uživatele
- možnost přidání aplikace na domovskou obrazovku zařízení
- možnost funkce offline za pomoci service workers

4.1.3 Nevýhody

- nemusí mít přístup ke všem možnostem zařízení (senzory a hardwarové funkce)
- nemusí fungovat na starších zařízeních
- jeden design pro více platforem

4.2 Cordova

Cordova je open-source framework pro vývoj mobilních aplikací. Pro vývoj jsou použity technologie HTML5, CSS3 a JavaScript. Cordova obaluje kód do tzv. WebView, čímž umožňuje přístup k nativnímu API zařízení. Tento fakt také umožňuje vývoj multiplatformní aplikace pomocí jediné codebase. Avšak toto pozitivum je značně znehodnoceno rychlostí takovýchto aplikací. Jelikož WebView značně zpomaluje chod aplikace oproti nativním aplikacím.

Další výhodou Cordova jsou pluginy. Cordova obsahuje tzv. Core plugins, které umožňují přistupovat k dalším schopnostem mobilního zařízení. Mezi tyto schopnosti se řadí např. souborový systém, geolokace, kamera, status baterie, síťové připojení.[20]

Schéma Cordova aplikace je možné k vidění na obrázku A.2.

4.2.1 Výhody

- využití známých technologií CSS, HTML a JS
- přístup k nativním funkcím zařízení (senzory a hardwarové funkce)
- možnost distribuce přes obchody s aplikacemi

4.2.2 Nevýhody

- rychlost oproti nativním aplikacím
- interpretace kódu pomocí webového prohlížeče

4.3 Flutter

Flutter je open-source framework vytvořený společností Google. Jedná se o multiplatformní framework, který ke svému chodu potřebuje SDK platforem, pro které se bude kód kompilovat. Flutter je UI framework, který vývojáři dovolí rychlé a jednoduché napsání layoutu aplikace. Velkou předností Flutteru je Hot reload. Jedná se o vlastnost, která vývojáři dovolí vidět změny v kódu bez nutnosti další kompilace aplikace. UI komponenty Flutteru se nazývají widgety. Z widgetů se tvoří stromová struktura, dále widget tree, která je zpracována a vykreslena. Pro psaní ve Flutter frameworku je použit programovací jazyk Dart.[21]

4.3.1 Výhody

- Hot Reload
- vyvíjen společností jenž vyvíjí samotný operační systém Android
- skvělá dokumentace a návody
- s Flutter frameworkem již delší dobu pracuji
- možnost volání nativního kódu z Dartu a vice versa

4.3.2 Nevýhody

- poměrně nový
- menší vývojářská komunita
- jeden design pro více platforem

4.4 Srovnání

Každá z výše zmíněných technologií patří do vlastní kategorie vývoje aplikací. Těmito kategoriemi jsou:

- Nativní
  - Flutter
  - Flutter je během kompilace překládán do strojového kódu dané platformy.
  - Aplikace je třeba instalovat na zařízení.
- Hybridní
  - Cordova
  - V případě Cordova je využito jazyků HTML, CSS a JS k vývoji aplikace.
  - Výsledek uživatelského rozhraní je vykreslen pomocí WebView.
  - Stejně jako u Flutteru je potřeba aplikaci kompilovat a instalovat na mobilní zařízení.
- Webová aplikace
  - PWA
  - Aplikace založena na bázi webové stránky.
  - Offline režim za pomoci Service workers.
  - Aplikace není třeba instalovat na mobilní zařízení.

Důkladnější srovnání výše zmíněných technologií jsem provedl na základě 3 dalších kategorií.

1. Vzhled
2. Funkcionality
3. Nasazení

Tabulka 4.1 slouží k vysvětlení jakým způsobem je vhodné technologii použít. Tato specifikace je využita dále v dokumentu pro porovnání technologií v kategoriích Vzhled a Funkcionality.

Název	Popis
Ideální	Ideální pro použití, plně odpovídá potřebám a očekáváním
Dobrý	Dobré pro použití, splňuje většinu požadavků, mohou existovat malé problémy nebo omezení
Dostačující	Dostačující pro použití, splňuje základní minimum
Špatné	Nevyhovující

Tabulka 4.1: Vysvětlení ideálnosti použití

4.4.1 Vzhled

Tato kategorie se zaměřuje na estetický zážitek z užívání aplikace. Porovnání technologií pro tuto kategorii je vidět v tabulce 4.2.

Úvodní obrazovka

Úvodní obrazovka, také známá jakožto splash screen, je taková obrazovka, která se ukáže uživateli během načítání aplikace. Obvykle zobrazuje název aplikace, její logo a verzi. Zároveň uživatele ujišťuje, že aplikace funguje v pořádku.

Ovládací prvky

Ovládací prvky aplikace jsou důležitou součástí z hlediska použitelnosti aplikace. Ovládací prvky by měli být jednoduché na porozumění, konzistentní, přístupné a responsivní. Responsivitou je myšlena okamžitá zpětná vazba na uživatelovu akci. Mobilní a webové aplikace mají určité rozdíly ve svém používání. Mobilní ovládací prvky většinou využívají dotyková gesta.

Animace

Animace jsou mocným nástrojem pro zlepšení uživatelské zkušenosti. Správné použití animací napomáhá ke zlepšení použitelnosti aplikace. Animace také

pomáhají k přilákání uživatelovi pozornosti a jeho angažovanosti v používání aplikace.

Kritérium	Flutter	Cordova	PWA
Úvodní obrazovka	Ano	Ano	Ano
Ovládací prvky	Ideální	Ideální	Ideální
Animace	Ideální	Dobrý	Ideální

Tabulka 4.2: Porovnání vzhledových elementů

Cordova obdržela v kategorii *Animace* hodnocení *Dobrý* kvůli své funkci vykreslování skrze WebView. Toto může negativně ovlivnit rychlost aplikace a zpomalit animace, což by mohlo vést k méně plynulému pohybu.

4.4.2 Funkcionality

Tabulka 4.3 vyobrazuje porovnání technologií pro tuto kategorii.

Offline použití

Offline režim je jedním ze stěžejních faktorů pro aplikaci. Ne vždy je třeba aby aplikace potřebovala internetové připojení. V mém případě se jedná např. o vyplňování formuláře. Jakmile jej máme jednou stažený, není třeba internetového připojení. Tudíž je dobré implementovat i offline režim, který umožní uživateli nezávislost na internetovém připojení.

Výkon

Výkon aplikace je dalším faktorem, který má vliv na uživatele. Pokud je aplikace pomalá a neresponsivní, častokrát máme tendenci aplikaci ukončit a přestat používat. Výkon úzce souvisí s Responsivitou ze sekce 4.4.1Vzhled. Výkon aplikace totiž do jisté míry ovlivňuje UX.

Senzory a hardwarové funkce

Pro naše účely je třeba, aby aplikace měla přístup k mikrofonu a fotoaparátu zařízení. Důvodem je možnost doplnění formuláře o multimediální data.

Kritérium	Flutter	Cordova	PWA
Offline použití	Ideální	Ideální	Ideální
Výkon	Ideální	Dobrý	Dobrý
Senzory a hardwarové funkce	Ideální	Dobrý	Dostačující

Tabulka 4.3: Porovnání funkcionalit

PWA aplikace nedisponují tak rozsáhlým přístupem k nativním senzorům zařízení jako Flutter či Cordova. Avšak poskytují přístup k hardwarovým funkcím a senzorům, které pro naše účely jsou dostačující. Proto PWA aplikace dostaly hodnocení *Dostačující*.

Cordova aplikace obdržely hodnocení *Dobrý*, jelikož jejich jádro je nativní a tím pádem mají přístup k nativním prvkům zařízení. Nicméně pro využití těchto prvků musí existovat plugin, který je schopen s nimi pracovat a předávat data dále do WebView.

4.4.3 Vývoj a distribuce

Tato sekce se zabývá vývojem a distribucí daných technologií a následným porovnáním, které je znázorněno v tabulce 4.4.

Náročnost při vývoji

Náročnost vývoje se dá dělit na několik faktorů. Mezi tyto faktory bych zařadil přepoužitelnost kódu, multiplatformní vývoj a dostupné knihovny.

Přepoužitelnost kódu a multiplatformní vývoj odkazují na schopnost použít stejný kód. V případě přepoužitelnost se může jednat například o jednotlivé komponenty, například různé grafické elementy jako jsou tlačítka, formulářové prvky, nebo celé obrazovky. V případě multiplatformního vývoje se jedná

o skutečnost, kdy za pomoci jediného kódu vytvoříme aplikaci pro více platforem, například pro mobilní zařízení a pro web.

Posledním faktorem jsou dostupné knihovny. Volně dostupné knihovny dokáží častokrát výrazně snížit délku vývoje aplikace a tím pádem značně ulehčit její vývoj.

Publikace

Publikování aplikace je nedílnou součástí vývoje aplikace. Musíme vědět jakým způsobem se námi vyvinutá aplikace bude publikovat, pokud se jedná o veřejně dostupnou aplikaci. V případě webových aplikací se jedná o nasazení na webový server. Pro mobilní aplikace naopak existují obchody s mobilními aplikacemi jako je Google Play nebo App Store.

Publikování se zdá být poměrně jednoduchým aktem, avšak i tato aktivita s sebou nese několik potíží na které si dát pozor, pokud chceme, aby byla aplikace dobře viditelná. V případě webové aplikace se jedná o tzv. Search engine optimization, kdy musíme naši stránku optimalizovat takovým způsobem, aby vyhledávače ukazovaly naši webovou aplikaci na přední stránce. Obchody s mobilními aplikacemi jako je Google Play nebo App Store taktéž obsahují svůj vlastní hodnotící systém pro zobrazování aplikací. I v tomto případě je tedy dobré optimalizovat. Tento druh optimalizace se nazývá App store optimalization.

Podpora a údržba

Aplikace je třeba udržovat. S časem přichází nové a lepší technologie, které zlepšují výkon a zabezpečení. Je tedy nutné aplikace udržovat aktuální, aby uživatelé měli co nejlepší uživatelskou zkušenost a nemuseli se obávat aplikaci používat.

Kritérium	**Flutter**	**Cordova**	**PWA**
Náročnost vývoje	Dobrý	Dostačující	Ideální
Publikace aplikace	Dobrý	Dobrý	Ideální
Podpora a údržba	Dobrý	Dostačující	Ideální

Tabulka 4.4: Porovnání vývoje a distribuce

Cordova si v kategorii *Náročnost vývoje* zaslouží hodnocení *Dostačující*. Přestože využívá moderní webové technologie, může nastat situace, kdy budeme muset vytvořit vlastní logiku v jádru nativní aplikace. To může vést ke komplikacím ve vývoji a zvýšení komplexity aplikace. Flutter si naopak ve stejné kategorii zaslouží hodnocení *Dobrý*. Ačkoli nejde o tak vyzrálou technologii jako webové technologie, obsahuje velké množství předpřipravených funkcí a UI prvků, které značně usnadňují vývoj aplikace.

V kategorii *Publikace* obdržely aplikace Flutter a Cordova hodnocení *Dobrý*, protože je nutné je kompilovat a nasadit do aplikačních obchodů. Na druhé straně, PWA může být nasazena na jakýkoli webový server, což ji okamžitě zpřístupňuje uživatelům. Díky URL adresám mohou uživatelé tyto aplikace snadno najít a používat.

Z hlediska kategorie *Podpora a údržba* je Flutter dobrý, protože je podporován společností Google a má aktivní komunitu vývojářů. Cordova je dostačující, protože i když je to zralý projekt, může vyžadovat více údržby kvůli rychlému vývoji webových technologií. Naproti tomu, PWA je ideální, protože je navržena tak, aby byla snadno aktualizovatelná a udržovatelná, což je velká výhoda v rychle se měnícím digitálním prostředí.

4.5 Závěr

Nejlepší technologií z kategorie Vzhled vychází remízou Flutter a PWA. Obě technologie disponují více než ideálními nástroji pro tvorbu uživatelského rozhraní.

Další kategorií je Funkcionalita. Zde značně vyhrává Flutter díky své povaze nativní aplikace, díky které má nejlepší výkon a přístup k hardwarovým funkcím. Nicméně pro mé účely stačí přístup pouze k fotoaparátu a mikrofonu zařízení, což dovoluje i PWA s dostačujícím přístupem k hardwarovým funkcím.

Flutter a Cordova aplikace je nutné publikovat do obchodů s aplikacemi. Kdežto PWA aplikace fungují stejně jako webové aplikace. Tedy, běží na webovém serveru. Tím pádem se nemusí nic publikovat do obchodů třetích stran. Také nám PWA umožňují aktualizovat aplikace skoro ihned. Vzhledem k tomu, že se jedná o webové aplikace, stačí pokud nahrajeme novou verzi a uživatel tím pádem při novém spuštění aplikace uvidí změny ihned. Kdežto u

aplikací, které je třeba publikovat do obchodů, musíme čekat na schválení, poté si uživatel musí aktualizaci nainstalovat a až teprve poté uvidí změny.

Další výhodou PWA je možnost instalace aplikace do mobilních zařízení. Mobilní zařízení vytvoří ikonu na domovské obrazovce a po stisknutí této ikony se otevře samostatné aplikační okno s WebView, které spouští naši aplikaci. Tímto způsobem vypadá PWA jakožto nativní aplikace. Díky tomuto mi tedy v kategorii Nasazení PWA vychází jako jasný vítěz.

Na základě tohoto porovnání mi vychází PWA jakožto nejlepší možnost pro realizaci aplikace. Použiji tedy tuto technologii k implementaci.

Kapitola 5

Aplikační požadavky

Tato kapitola popisuje funkční a nefunkční požadavky na aplikaci. Tyto požadavky vycházejí z již zmíněné aplikace MONDIS. Struktura požadavků vychází z konceptu MoSCoW prioritizace požadavků. Formulář a záznam jsou dva základní pojmy, které souvisejí s procesem vyplňování a ukládání dat. Je důležité správně rozlišovat mezi těmito dvěma pojmy.

Formulář je strukturovaná sada polí, která slouží k zadávání informací. Je vytvořen službou generující formuláře ze šablony a ještě neobsahuje žádné vyplněné údaje. V podstatě je to prázdná šablona, která umožňuje uživatelům vyplňovat informace. Formulář tak slouží jako nástroj pro sběr dat od uživatelů. Pro úplnost bych dodal, že název formuláře je označován jako šablona.

Záznam je na druhou stranu již vyplněný formulář, který byl uložen na serveru. Oproti prázdnému formuláři obsahuje záznam konkrétní informace zadané uživatelem. Díky uložení na serveru je záznam jasně identifikovatelný, což umožňuje jeho další zpracování, analýzu a v případě potřeby i editaci či sdílení s dalšími uživateli.

Tedy zatímco formulář je prázdná šablona pro sběr informací, záznam je konkrétní soubor dat, který vznikl vyplněním formuláře.

5.1 MoSCow

Tato metoda požadavků vytváří prioritizaci požadavků na základě 4 atributů. Těmito atributy jsou:

1. Must have (M)
   - Tato kategorie popisuje takové požadavky, bez kterých není možné aplikaci používat a jsou tedy nutné k implementaci.
2. Should have (S)
   - Požadavky, které nejsou vitální pro aplikaci, ale přinášejí signifikantní hodnotu. Bez jejich implementace bude aplikace funkční.
3. Could have (C)
   - Tyto požadavky lze nazvat jako „nice to have“. Jedná se o požadavky, které mají malý dopad na aplikaci a slouží spíše k usnadnění práce s aplikací.
4. Will not have (W)
   - Tato kategorie popisuje takové požadavky, které nejsou prioritní na vývoj a mohou být zanedbány. Většinou jsou tyto požadavky implementovány v pozdější verzi aplikace. Nejedná se o požadavky, které by vyloženě nemohly být součástí aplikace.

5.2 Funkční požadavky

Mezi funkční požadavky patří takové požadavky, které popisují a definují funkce systému. Jakožto funkční požadavky jsem identifikoval následující.

5.2.1 FP1 - Přihlášení a odhlášení

FP1.1 C Systém umožní uživateli přihlášení do aplikace.

FP1.2 C Systém umožní uživateli odhlášení z aplikace.

- Požadavek je propojen s FP1.1. Bez implementace FP1.1 nelze tento požadavek implementovat.

5.2.2 FP2 - Správa formulářů

FP2.1 M Aplikace umožní uživateli zobrazit stažitelné formuláře.

- Aplikace stáhne výčet dostupných šablon ze serveru. Z nich má následně uživatel možnost výběru.

FP2.2 M Aplikace umožní uživateli zobrazit stažitelné záznamy.

FP2.3 M Aplikace umožní uživateli stažení formuláře do zařízení.

FP2.4 M Aplikace umožní uživateli stažení záznamu do zařízení.

FP2.5 M Aplikace umožní uživateli zobrazit všechny stažené formuláře a záznamy.

FP2.6 M Aplikace umožní uživateli smazání formuláře i záznamu ze zařízení.

FP2.7 C Aplikace umožní uživateli stáhnout více záznamů naráz.

FP2.8 M Aplikace umožní uživateli upravení formuláře i záznamu.

- Úpravou je myšleno otevření a úprava odpovězených hodnot v již staženém formuláři/záznamu s následným lokálním uložením změn.

FP2.9 M Aplikace umožní uživateli odeslání formuláře i záznamu na server k uložení a zaznamenání odpovědí.

FP2.10 W Aplikace umožní uživateli vytvoření vlastního formuláře.

- Tento požadavek aktuálně není nutné implementovat, jelikož existuje webová aplikace[1], která je k tomuto určena.

FP2.11 S Aplikace umožní vytvoření duplikátu formuláře.

FP2.12 S Aplikace umožní import formuláře ze souboru.

- V případě, že nebude mít uživatel k dispozici internetové připojení, ale bude mít k dispozici soubor s reprezentací formuláře, systém mu umožní jej importovat.

FP2.13 C Aplikace umožní importovat více formulářů najednou.

FP2.14 C Aplikace umožní sdílení formuláře.

FP2.15 S Aplikace umožní automatické uložení záznamů na server.

- Tímto požadavkem je myšleno automatické odeslání záznamů (odpovězených formulářů) na server k uložení. Důvodem je předcházení nechtěné ztráty dat zapříčiněné ztrátou či rozbitím zařízení.

[1] `https://semantic-form-editor.vercel.app/`

FP2.16 C Aplikace umožní uživateli zobrazit historii odpovězených a odeslaných záznamů.

FP2.17 C Aplikace umožní automatickou aktualizaci definice formuláře pokud dojde ke změně na serveru.

5.2.3 FP3 - Práce s formulářem

FP3.1 M Aplikace umožní otevřít formulář i záznam.

FP3.2 M Aplikace umožní vyplnění formuláře i záznamu.

FP3.3 M Aplikace umožní odpovědět na otázku v podobě multimediálního souboru.

- Aktuálně knihovna SForms neobsahuje možnost takovéto odpovědi. Mým úkolem tedy bude tuto možnost implementovat přímo do SForms.

FP3.4 M Aplikace umožní změnu multimediálních souborů u formuláře či záznamu.

FP3.5 M Aplikace umožní odebrání multimediálních souborů z formuláře či záznamu.

FP3.6 M Aplikace umožní přidání GPS souřadnic k formuláři.

- Pro tento požadavek lze využít mapovou komponentu SForms-GeoComponents[2], která umožňuje uživateli vybrat souřadnice přímo na mapě a využívá Geolocation API.

FP3.7 C Aplikace umožní anonymní vyplnění formuláře.

FP3.8 S Aplikace umožní lokální smazání odpovědí ze záznamu.

FP3.9 M Aplikace umožní duplikaci vybrané otázky ve formuláři.

- SForms neumožňují duplikaci uzlu Question (popsán v sekci 2.1.3). Je tedy mým úkolem SForms o tuto funkcionalitu doplnit. Vybrané uzly budou patřičně označeny v JSON-LD.

FP3.10 S Aplikace umožní smazání duplikované otázky.

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

5.2.4 FP4 - Usnadnění

FP4.1 S Aplikace umožní filtrování mezi formuláři.

- Filtrování bude probíhat na základě názvu, tagu nebo priority.

FP4.2 W Aplikace umožní filtrovat záznamy s určitou odpovědí.

FP4.3 C Filtrování bude možné kombinovat.

FP4.4 C Aplikace umožní tagování formulářů.

- Tagováním se rozumí přidání vlastního příznaku formuláři.

FP4.5 C Aplikace umožní přiřazení priority formuláři.

- Priorita určuje přednost vyplňování formuláře.

FP4.6 C Aplikace umožní řazení formulářů na základě priority, tagu nebo názvu.

FP4.7 W Aplikace umožní nastavit připomenutí nebo oznámení pro termíny sběru dat nebo jiné události.

- Tento požadavek nebude zanesen do aplikace, jelikož aktuálně neexistuje možnost nastavení času zobrazení lokálních notifikací zařízení. K implementaci plánovaných notifikací by bylo vhodné využít Notification Triggers[3]. Avšak vývoj této funkcionality byl zastaven a není možné ji použít.
- Návrh řešení tohoto požadavku ovšem bude popsán.

FP4.8 C Aplikace umožní uživateli lokálně přejmenovat stažený formulář/záznam.

- V tomto případě lze o formuláři a záznamu přemýšlet jako o stejné entitě.

5.3 Nefunkční požadavky

V této sekci jsou obsaženy požadavky na aplikaci, jenž uživatel žádným způsobem nevidí, ale na práci s aplikací mají také vliv. Jedná se o následující požadavky.

NP1 M Aplikace pro vykreslení formuláře využije knihovnu SForms.

[3] `https://developer.chrome.com/docs/web-platform/notification-triggers/`

NP2 M Formulář bude reprezentován ve formátu JSON-LD.

NP3 M Aplikace musí cachovat stažené formuláře a záznamy.

NP4 M Aplikace musí cachovat autocomplete (typeahead) hodnoty formulářů.

NP5 C Vývojář bude mít možnost vlastní stylizace formulářových prvků.

- Aplikaci bude možné používat na mobilní zařízeních a tabletech. Design bude přizpůsoben zařízení.

NP6 S Responsivní design

- Aplikaci bude možné používat na mobilní zařízeních a tabletech. Design bude přizpůsoben zařízení.

NP7 M Aplikaci bude možné používat v offline režimu.

NP8 M Aplikaci bude dostupná pro platformy Android a iOS.

NP9 C Aplikaci bude bezpečná a chránit uživatelská data před neoprávněným přístupem a zneužitím.

- Tímto je myšleno hlavně šifrování uložených dat v zařízení.

NP10 S Aplikace bude kontrolovat zda používaný prohlížeč je kompatibilní.

NP11 S Uživatel bude mít možnost aplikaci nainstalovat do svého zařízení.

- V případě, že prohlížeč nebude kompatibilní, bude uživatel náležitě upozorněn.

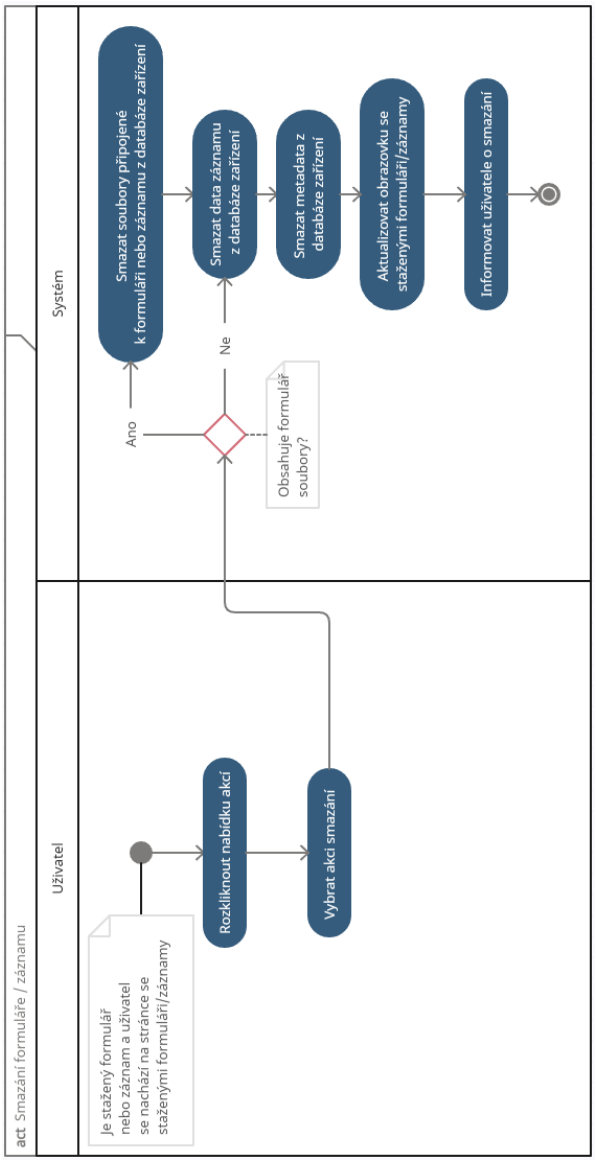
5.4 Případy užití

Případy užití vycházejí z aplikace MONDIS. Diagram A.4 znázorňuje identifikované případy užití. Oranžová barva znázorňuje případy užití, které základní knihovna SForms neumožňuje a je potřeba o ně knihovnu doplnit. Zbytek případů užití je založen na případech užití MONDIS s jednoduchou generalizací. Například místo vyplnění jednotlivých sekcí lokalizace, identifikace a posouzení stavu je v případech užití jen *Vyplnění formuláře*. Důvodem je obecné využití vytvářené aplikace.

5.4.1 Diagramy aktivit

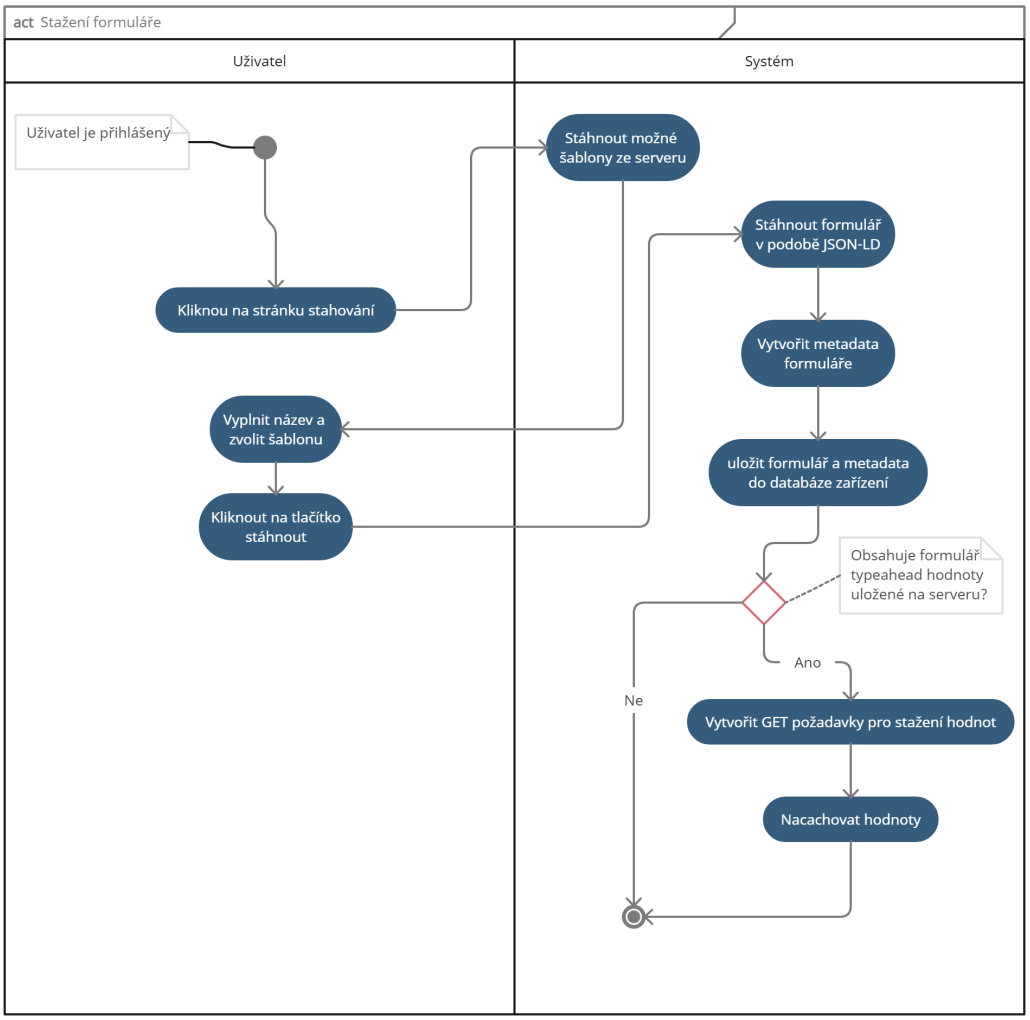
Tato sekce popisuje vytvořené diagramy aktivit, které naplňují některé funkční a nefunkční požadavky aplikace. Diagramy aktivit se zaměřují hlavně na *Must have* požadavky.

Diagram 5.1 znázorňuje proces smazání formuláře / záznamu ze zařízení. Tento diagram naplňuje funkční požadavek **FP2.5**.



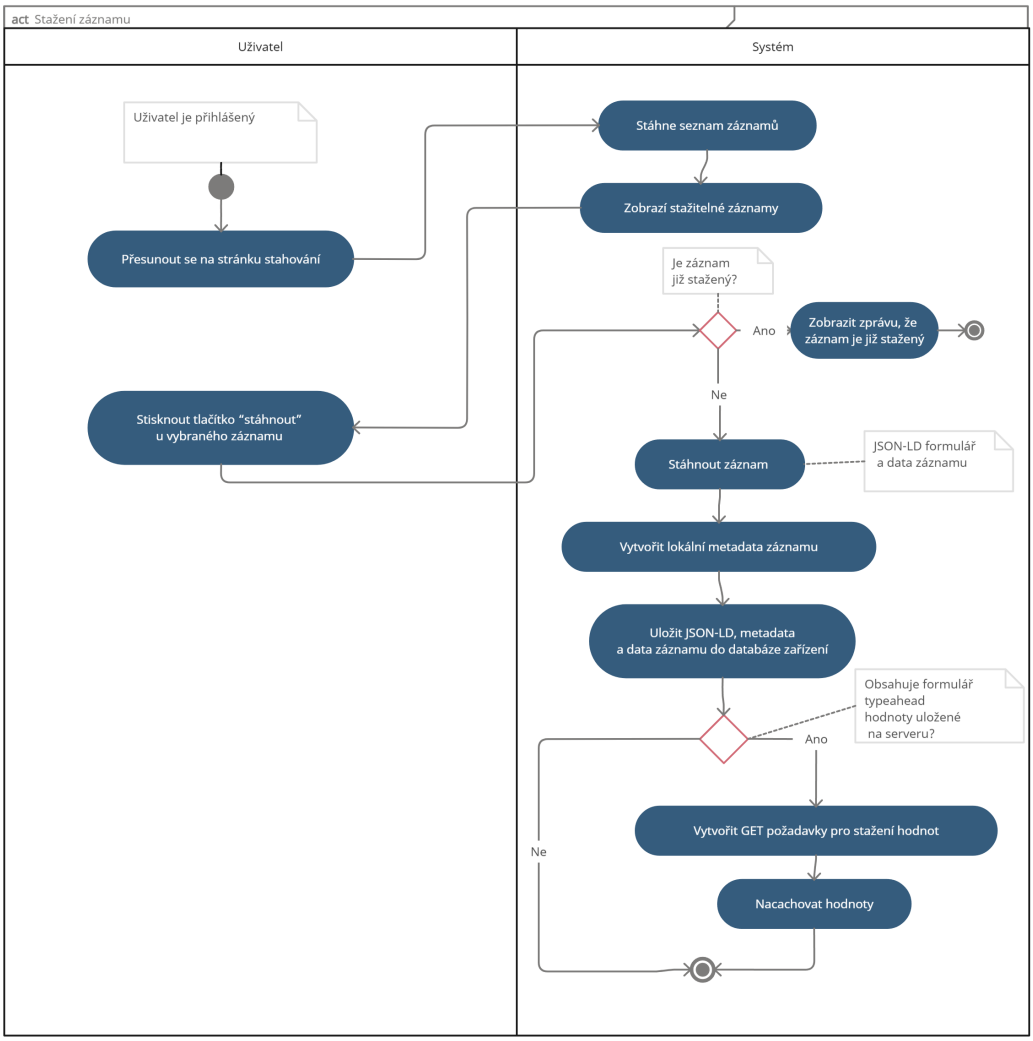
Obrázek 5.1: Diagram aktivity smazání formuláře/záznamu

Diagram 5.2 naplňuje funkční požadavky **FP2.1** a **FP2.3**, dále naplňuje nefunkční požadavek **NP3** a **NP4** a zobrazuje proces stažení formuláře a následné uložení do zařízení. Uživatelem vyplněný název slouží k pojmenování staženého formuláře v zařízení.



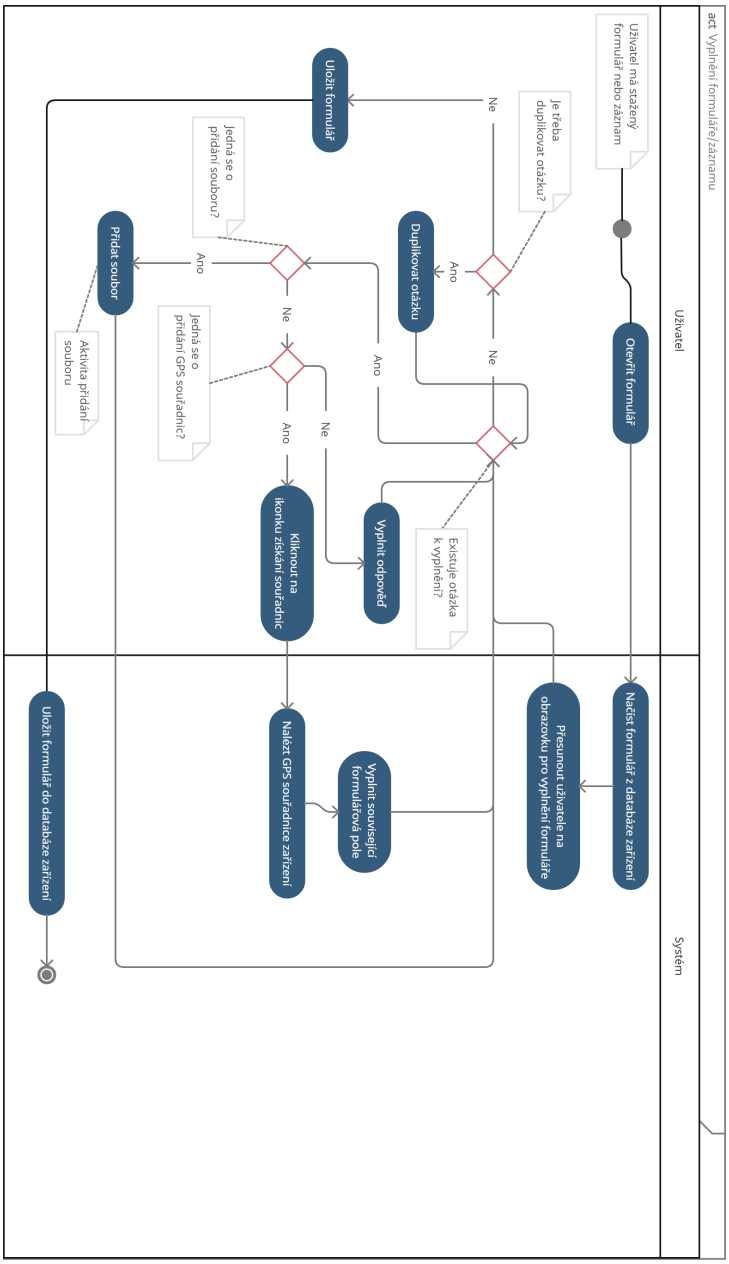
Obrázek 5.2: Diagram aktivity stažení formuláře

Dále bylo nutné navrhnout posloupnost kroků pro stažení záznamu ze serveru. Jakým způsobem je řešení navrženo je k vidění na diagramu 5.3. Tento diagram naplňuje požadavky **FP2.2** a **FP2.4**.



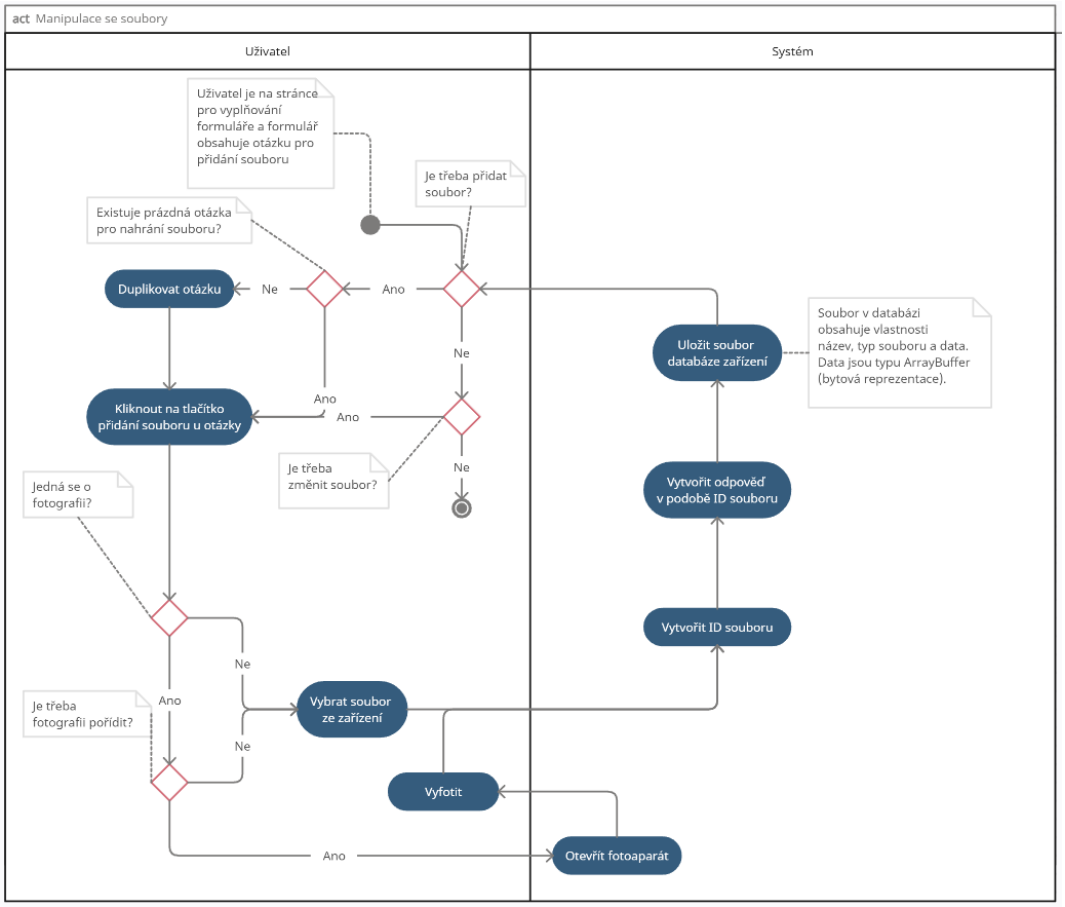
Obrázek 5.3: Diagram aktivity stažení záznamu

Diagram 5.4 znázorňuje postup vyplňování formuláře (editace záznamu). Diagram naplňuje požadavky **FP2.5**, **FP2.8**, **FP3.1**, **FP3.2**,**FP3.6** a **FP3.9**. Jinými slovy diagram ukazuje průchod vyplňování formuláře od samotného otevření, přes připojování souborů, duplikaci otázek až po samotné uložení.



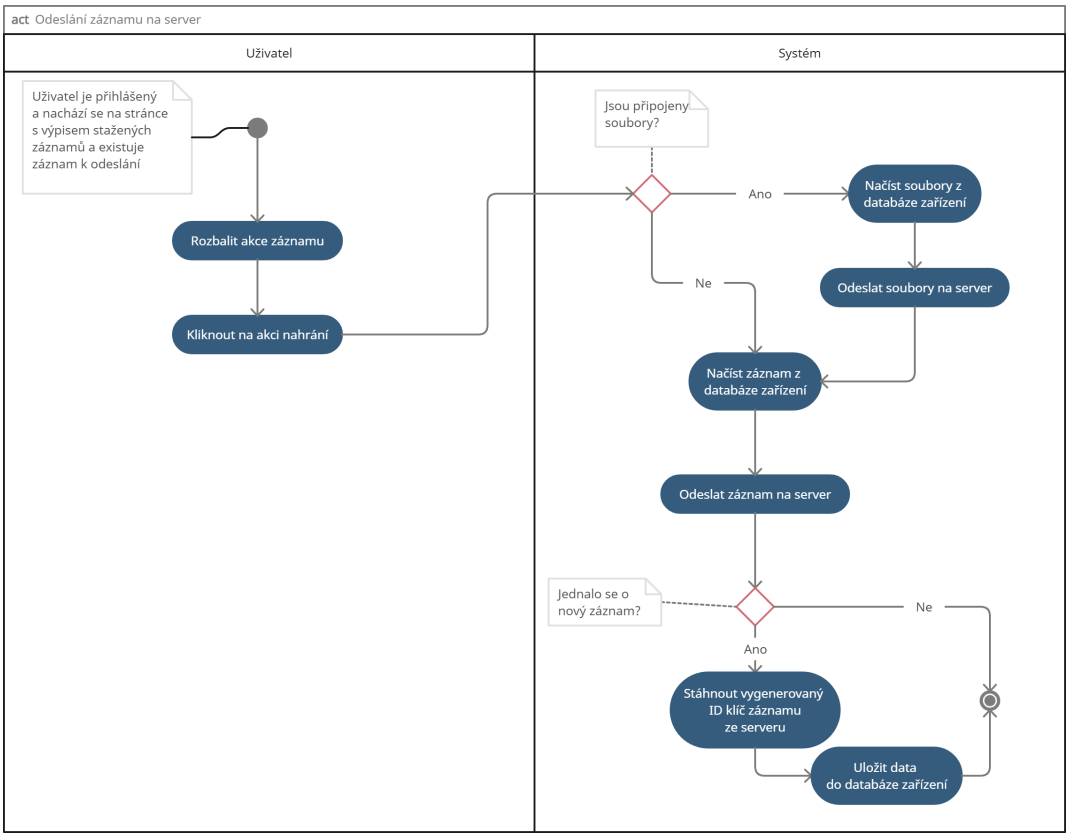
Obrázek 5.4: Diagram aktivity vyplnění formuláře

Jakým způsobem bude probíhat přidání souboru k formuláři je znázorněno na diagramu 5.5. Diagram naplňuje požadavky **FP3.3** a **FP3.4**. Ačkoliv diagram neobsahuje část smazání souboru, pokusím se ji alespoň textově nastínit. Jakmile uživatel nahraje soubor do formuláře, objeví se u otázky tlačítko pro smazání souboru (odstranění odpovědi). Zmáčknutí tohoto tlačítka bude mít za následek odstranění souboru z formuláře a následně smazání souboru z lokální databáze.



Obrázek 5.5: Diagram aktivity přidání souboru

V poslední řadě diagram 5.6 ukazuje proces odeslání záznamu na server k uložení. Tento diagram splňuje požadavek **FP2.9**. V případě, že se na server posílá nový záznam k uložení, je serverem k takovému záznamu vytvořen identifikační klíč. Tento klíč je nutné stáhnout a lokálně přidat do záznamu. Nový záznam vznikne lokálně tím způsobem, že uživatel stáhne formulář, který následně vyplní.



Obrázek 5.6: Diagram aktivity odeslání záznamu na server

Kapitola 6

Návrh řešení

V této kapitole popíši návrh aplikace. Přiblížím jaké technologie budou použity a jak bude dosaženo některých funkčních a nefunkčních požadavků.

6.1 Technologie

Jakožto nejlepší možná technologie pro tvorbu aplikace mi vyšla PWA. Jelikož je třeba využít knihovny SForms pro vykreslení sémantických formulářů, je vhodné využít knihovnu React[1]. Důvodem je, že samotná knihovna SForms je napsána právě za pomoci knihovny React.

Pro zjednodušení vývoje bude využito nástroje Create-react-app[2]. Create-react-app obaluje nástroje jako je Webpack a Babel, které slouží k sestavování React aplikací. Tento nástroj slouží k usnadnění a optimalizaci sestavení aplikace.

[1] `https://reactjs.org/`
[2] `https://create-react-app.dev/`

6.2 Vzhled

Vývoj moderních online aplikací je do značné míry závislý na designu uživatelského rozhraní (UI), který určuje celkovou užitečnost, přístupnost a estetiku projektu. Material-UI[3] a React-Bootstrap[4] jsou dvě oblíbené technologie uživatelského rozhraní, které usnadňují vytváření rozhraní, jež jsou vysoce funkční a zároveň estetická.

6.2.1 Material-UI

Material-UI je populární framework UI komponent implementující pokyny Material Designu společnosti Google. Nabízí bohatou sadu komponent, možnosti tematizace a systém responzivního rozvržení, který zjednodušuje proces vývoje estetických a vysoce funkčních rozhraní.[22]

Výhody

- Poskytuje rozsáhlou sbírku předpřipravených komponent, které dodržují zásady Material Designu, a zajišťují tak konzistentní a moderní uživatelské rozhraní.
- Bohatá dokumentace, která vývojářům usnadňuje učení a osvojení frameworku.
- Možnost vytvářet vysoce přizpůsobená rozhraní, bez nutnosti vzdát se výhod systému Material Designu.
- Rozsáhlá a aktivní komunita, která vývojářům nabízí přístup k bohatým zdrojům a podpoře.

Nevýhody

- Strmější křivka učení pro vývojáře, kteří s tímto frameworkem začínají.

[3] `https://mui.com/`
[4] `https://react-bootstrap.github.io/`

- Větší velikost balíčků, kvůli rozsáhlosti knihovny, což může mít potenciální vliv na výkon aplikace.

6.2.2 React-Bootstrap

React-Bootstrap je knihovna UI komponent postavená nad toolkitem Bootstrap[5] a speciálně navržená pro práci s knihovnou React. Poskytuje sadu komponent, které využívají sílu komponentové architektury React.[23]

Výhody

- Využití široce známého frameworku Bootstrap, což vývojářům usnadňuje jeho osvojení a použití v jejich projektech.
- Navržen tak, aby nativně spolupracoval s Reactem, což zajišťuje snadnou integraci komponent Bootstrap do projektu založeného na knihovně React.
- Struktura založená na komponentách podporuje vývoj modulárního, opakovaně použitelného kódu, což zlepšuje udržovatelnost.
- Zachovává responzivní design Bootstrapu a zajišťuje konzistentní výkon na různých zařízeních a při různých velikostech obrazovky.

Nevýhody

- React-Bootstrap nabízí ve srovnání s Material-UI méně komponent, což může potenciálně omezit možnosti návrhu.
- Absence systému návrhu, jak je tomu u Material-UI (Material design), může od vývojářů vyžadovat další úsilí k zajištění konzistence uživatelského rozhraní.

[5] `https://getbootstrap.com/`

6.2.3 Shrnutí

Z hlediska rozmanitosti komponent, návrhového systému a jednoduchosti při vývoji mají Material-UI a React-Bootstrap své výhody i nevýhody. Nicméně právě kompatibilita React-Bootstrap s již používanou knihovnou SForms vedla k rozhodnutí použít React-Bootstrap v této diplomové práci. Použití Material-UI by popřelo všechny potenciální výhody tím, že by do projektu přidalo zbytečnou složitost a velikost aplikace.

React-Bootstrap umožňuje projektu využít výhod responzivního a dobře známého toolkitu pro uživatelské rozhraní Bootstrap, který dobře spolupracuje se stávající knihovnou pro vykreslování formulářů, což podporuje efektivitu a udržovatelnost při vývoji uživatelského rozhraní projektu.

6.3 Požadavky

6.3.1 Případ užití MONDIS

Řešením stejné funkcionality, jako měla aplikace MONDIS, je vytvoření šablony formuláře. Tato šablona bude obsahovat stejné prvky jako měl vyplňovaný formulář v MONDIS. Šablona bude obsahovat 3 stránky, kde každá bude odpovídat jednotlivé kategorii identifikace, lokalizace a posouzení stavu. Největším problémem pro tento případ užití bude kategorie lokalizace. Tato kategorie by měla obsahovat mapu a možnost vyplnění z ÚSKP (Ústřední seznam kulturních památek České republiky). Čehož bez internetového připojení nelze dosáhnout. Uživatelé tedy budou nuceni vyplnit tuto část během připojení na internet, nebo využít možnosti lokalizace pomocí adresy nebo zeměpisných souřadnic. Jako řešení kategorie lokalizace navrhuji využít rozšíření SForms GeoComponents.

6.3.2 Offline použití

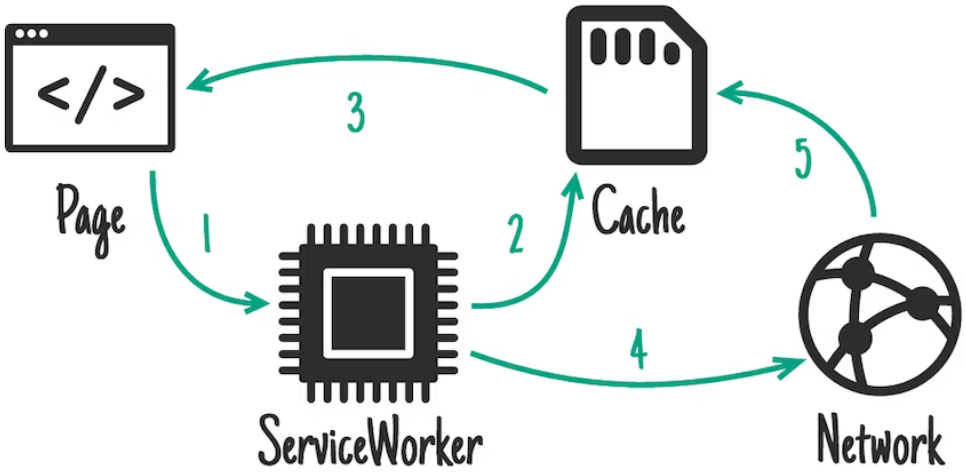
Jak jsem již nastínil v sekci 2.2.1, k dosažení offline režimu aplikace bude využit Service worker. Bohužel Service worker má omezené pole působnosti a nemůže přímo komunikovat s DOM strukturou aplikace a je možné jej

použít pouze v asynchronním kontextu. Jinými slovy nelze jej použít k XHR[6] požadavkům a využití Web Storage. Do Web Storage spadá LocalStorage a SessionStorage.

Aplikace bude vyvíjena přístupem offline-first. Což pro nás znamená, že aplikaci budeme vytvářet takovým způsobem, aby v první řadě fungovala správně v offline režimu.

V našem případě se jedná hlavně o cachování požadavků na server. Jedná se hlavně o GET požadavky. Mezi GET požadavky spadá například i stahování formulářů a možností pro SForms komponentu Typeahed. Pro naše účely bude nejlepší použít metodu podávání obsahu „Stale-while-revalidate“[7]. Tato metoda nám zajistí, že všechny cachovatelné odpovědi na požadavky se nacachují a bude možné je při dalších požadavcích podávat přímo z cache. Hlavní myšlenkou této metody je doptání se pomocí Service worker do cache, zda je požadavek nacachováný. Pokud je nacachovaný, je podán aplikaci z cache. Dále je požadavek poslán na server. Pokud není, je vytvořen požadavek na server. V tomto případě je zapotřebí internetové připojení. V takovéto situaci bude v aplikaci muset existovat služba, která uživatele upozorní v případě, že je vyžadováno internetové připojení a on připojený není.

Diagram znázorňující metodu „Stale-while-revalidate“ je k vidění na obrázku 6.1.



Obrázek 6.1: Metoda cachování - Stale-while-revalidate, zdroj: `https://web.dev/offline-cookbook/#stale-while-revalidate`

[6] `https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest`
[7] `https://web.dev/offline-cookbook/#stale-while-revalidate`

6.3.3 Cachování

Cachování je základní stavebním kamenem pro tvorbu offline aplikace. Pro naše účely je potřeba hlavně ukládat/cachovat stažené formuláře a typeahead hodnoty pro ně určené.

Pro tento účel je možné použít hned několik webových úložišť. Mezi webové úložiště se řadí například LocalStorage, SessionStorage, CacheStorage a IndexedDB. Pro výběr správné technologie je třeba si uvědomit několik faktorů, které mají na implementaci a používání vliv. Prvním a nejvýznamnějším faktorem je maximální kapacita daného úložného prostoru. Druhým faktorem je typ dat, který jsme schopni pomocí dané technologie ukládat. Jako třetí faktor bych určil, zda má Service worker přístup do daného úložiště.

Všechna výše zmíněná webová úložiště trpí stejnou vadou. Tou je, že kontrolu nad nimi má prohlížeč a data v těchto úložištích nejsou ve výchozím stavu perzistentní a prohlížeč může smazat data z těchto úložišť, když potřebuje. Abychom docílili perzistence těchto úložišť, je třeba, aby nám uživatel udělil oprávnění. Po udělení oprávnění si prohlížeč zapamatuje, že by neměl mazat data z úložišť pro danou aplikaci a tím pádem docílíme perzistence.[24]

LocalStorage a SessionStorage

LocalStorage i SessionStorage mají poměrně malou kapacitu v řádech desítek MB. Navíc SessionStorage uchovává uložená data pouze po dobu, kdy je uživatel přítomen v aplikaci. Tudíž ani jedno z těchto úložišť není ideálním pro ukládání větších souborů. Pro nás se jedná hlavně o JSON-LD soubory, které představují formuláře a záznamy.[24]

LocalStorage v našem případě můžeme použít pro ukládání jednoduchých klíč-hodnota párů, které pro nás mohou být nějakým způsobem důležité, například kdy se uživatel naposledy přihlásil do aplikace nebo jaký jazyk aplikace má nastavený.

CacheStorage

CacheStorage je rozhraní pro ukládání objektů typu Cache. Objekty typu Cache, respektive Cache API, slouží k ukládání páru HTTP požadavku a odpovědi do perzistentní paměti. Toto úložiště nedisponuje automatickou aktualizací uložených objektů. Tudíž je na nás, abychom zavedli mechanismus, který bude aktualizovat uložené objekty. CacheStorage lze využít i přímo ze Service worker.[25]

Kapacita tohoto úložiště je limitována v každém prohlížeči zvlášť. Například prohlížeč Chrome má možnost zaplnit až 80 % kapacity disku. Prohlížeč Firefox disponuje možností ukládat data až do 50 % kapacity disku.[24]

Prohlížeč Safari je v tomto ohledu poměrně rozdílný. Dle článku „Storage for the web“ umožňuje Safari využít pouze 1GB místa v zařízení. Pokud je tato kvóta překročena, nebo se blíží limitu, je uživatel upozorněn a Safari umožní navyšovat tuto hranici v 200MB inkrementech. Ovšem pokud je PWA nainstalována na zařízení, je pro aplikaci vytvořen nový úložný kontejner, který při vyčerpání kvóty nenavyšuje kapacitu.[26]

Pro naše účely se dá toto úložiště použít na cachování požadavků pro typeahead hodnoty.

6.3.4 IndexedDB

IndexedDB je webová API pro ukládání a získávání strukturovaných dat v prohlížeči. Jedná se o klíčovou technologii pro vytváření offline aplikací, protože umožňuje ukládání dat přímo na straně klienta. Toto úložiště, na rozdíl od předchozích, disponuje možností ukládat jakákoliv strukturovaná data, mezi které se řadí i soubory/bloby (binárně velké objekty). IndexedDB je transakční databázový systém. Ukládání a čtení z této databáze je pouze asynchronní, aby zbytečně nezatěžovalo a neblokovalo webové aplikace, které jej používají. Jelikož je IndexedDB asynchronní, lze použít ze Service worker.[27]

Pro naše účely je vhodné toto úložiště využít pro ukládání JSON-LD souborů formulářů a případné doplňující informace k formuláři.

6.3.5 Duplikace formuláře

Dalším požadavkem je duplikace formuláře. Mnou navrhované řešení vypadá následovně.

1. Vytvořit duplikát JSON-LD souboru, který představuje formulář.
2. Uvnitř zduplikovaného souboru změnit ID grafových uzlů za pomoci UUID[8].
3. Změnit ID uzlů relací každého uzlu na nová ID. Těmito relacemi jsou:
   - has-preceding-question
   - has-possible-value
   - has_related_question
4. Uložit takto pozměněný soubor.

6.3.6 Duplikace otázky

Duplikace otázky se týká přímo knihovny SForms. Je nutné v knihovně přidat možnost duplikace otázky. V první řadě je nutné přidat nový UI prvek, který po kliknutí spustí proces duplikace otázky. Ne každá otázka ovšem může být duplikována, proto tedy navrhuji každému uzlu přidat vlastnost *is-duplicable*. Pokud uzel tuto vlastnost obsahuje a je nastavena na hodnotu *true*, je možné daný uzel duplikovat. Algoritmus pro duplikaci otázky bude obsahovat následující kroky.

1. Nalezení rodiče duplikovaného uzlu
2. Vytvoření kopie originálního uzlu
3. Změna ID a relací kopie uzlu, včetně všech jeho dětí (podotázek)
4. Smazání případné odpovědi
   - Je možné, že původní uzel obsahoval odpověď. V takovém případě je odpověď smazána z kopie.
5. Přidání kopie uzlu do pole podotázek rodičovského uzlu

[8] `https://www.ietf.org/rfc/rfc4122.txt`

6. Nastavit hodnotu vlastnosti has-preceding-question u kopie uzlu na hodnotu ID původního uzlu

7. Nalézt prvního sourozence původního uzlu

8. Nastavit hodnotu has-preceding-question prvního sourozence na ID kopie uzlu

6.3.7 Lokální ukládání formulářů/záznamů

Ukládání formulářů a záznamů bude v aplikaci řešeno pomocí ukládání souboru JSON-LD přímo do databáze IndexedDB. Pro správnou offline funkcionalitu je nutné uložit i případné typeahead hodnoty. Ukládání typeahead hodnot se dá vyřešit pomocí CacheStorage a využití strategie Stale-While-Revalidate. Tato strategie stáhne možné hodnoty, uloží je do Cache a následně je používá v případě, že uživatel není připojen k internetovému připojení. Pokud se uživatel k internetu připojí a aplikace vytvoří nové požadavky pro stažení možných hodnot, tak tato strategie zkontroluje, zda jsou hodnoty uložené v Cache stále validní. V případě, že nejsou, jsou přepsány novými validními hodnotami.

Pro ulehčení ukládání typeahead hodnot a využití strategie Stale-While-Revalidate navrhuji využít knihovnu Workbox[9]. Workbox obsahuje implementaci této strategie a jeho využití je poměrně jednoduché a přímočaré. Jakým způsobem využít knihovnu Workbox pro cachování GET požadavků je možné vidět na ukázce kódu 4. Daný kód je nutné vložit do Service workeru.

6.3.8 Lokální přejmenování formuláře/záznamu

Pro možnost lokálního přejmenování formuláře nebo záznamu je nutné vytvořit vlastní strukturu, která bude udržovat možné hodnoty připojené ke stažené entitě a bude uložena v zařízení. Navrhuji vytvoření struktury nazvané Metadata. Tato struktura bude obsahovat lokální název a případné další vlastnosti. Uložení této struktury bude opět do IndexedDB. Jak by mohla struktura vypadat je k vidění na ukázce kódu 6.

[9] `https://developer.chrome.com/docs/workbox/`

6.3.9 Přidání GPS souřadnic

Tento požadavek je možné vyřešit pomocí využití rozšíření knihovny SForms GeoComponents. Toto rozšíření pracuje s moderním API Geolocation API a má tedy možnost využití polohových služeb zařízení. Dále disponuje možností vykreslení mapy, kde má uživatel možnost vytvořit bod na mapě. GPS souřadnice vytvořeného bodu je následně možné vložit jakožto odpověď na asociované otázky.

6.3.10 Sdílení formuláře

Pod sdílením formuláře si představme poslání reprezentace JSON-LD formuláře jinému uživateli. Tento požadavek je možné vyřešit exportováním JSON-LD reprezentace formuláře a následného stažení přímo do zařízení jakožto soubor. Tento nově stažený soubor by byl v zařízení uložen na místě ke kterému má uživatel možnost přistoupit a zároveň ví, kde se formulář nachází. Tento soubor by mohl následně sdílet libovolnou cestou, ať už by se jednalo o poslání přes email nebo sociální síť. Nicméně pro ulehčení celého tohoto zdlouhavého procesu se naskytuje možnost využití Web Share API. Toto moderní API poskytuje možnost využití nativních sdílecích služeb zařízení, na které je uživatel zvyklý. V případě, že API nebude podporováno, je dobré implementovat právě i možnost exportování formuláře jakožto záložní způsob.

6.3.11 Editace formuláře/záznamu

Řešení tohoto požadavku je poměrně triviální, jelikož knihovna SForms umožňuje zpětné získání formuláře i s úpravami, které uživatel udělal, v podobně JSON-LD. Tuto novou reprezentaci formuláře následně uložit do IndexedDB a přepsat stará JSON-LD data formuláře.

6.3.12 Multimediální soubory

Pro implementaci tohoto funkčního požadavku je zapotřebí doplnit RDF strukturu formuláře o trojice, které budou označovat vstup pro multimediální soubory. Příklad takových trojic je k vidění na ukázce kódu 5.

Zároveň je potřeba vytvořit komponentu do knihovny SForms, která se bude starat o vykreslování UI prvků potřebných pro nahrání a odstranění multimediálních souborů.

Dále je třeba vytvořit rozhraní pomocí kterého by knihovna SForms reagovala na akce spojené s nahráním a mazáním souborů. Bude tedy nutné knihovnu obohatit o 3 metody, které budou volány v návaznosti na vytvořené akce. Těmito akcemi jsou otevření formuláře, změna/nahrání souboru a odstranění souboru. Pro akci otevření formuláře je nutné do SForms přidat metodu, která se vykoná pouze při otevření formuláře a prvotním vykreslení. Metoda musí být definována mimo SForms, aby měl vývojář možnost vlastní implementace. V našem případě se jedná o načtení souborů z IndexedDB. Další metoda bude reagovat na akci nahrání souboru a v poslední řadě bude existovat metoda reagující na odstranění souboru. Všechny tyto metody musejí mít možnost definování mimo SForms ze stejného důvodu jako metoda reagující na otevření formuláře.

Jak jsem již nastínil v předchozím odstavci, implementace multimediálních souborů bude řešena pomocí IndexedDB. Každý soubor bude uložen jakožto ArrayBuffer[10] a ne přímo jako blob nebo soubor. Důvodem je, že některé prohlížeče nepovolují ukládání těchto souborů do IndexedDB. Nejlepším řešením je tedy soubory převézt do ArrayBuffer typu a následně s dalšími vlastnostmi, jako je název nebo typ souboru, uložit do IndexedDB.

6.3.13 Importování formuláře

Mimo normální stahování formulářů z internetu vytvořím i možnost pro nahrání JSON-LD souboru do aplikace. Aplikace si tento soubor uloží mezi ostatní stažené formuláře a dovolí s ním pracovat.

6.3.14 Priorita a Tagování

Tagování a priorita bude řešeno ukládáním doplňujících informací do struktury Metadata. Hlavním důvodem je fakt, že se jedná pouze o uživatelem přidané hodnoty. Nebudou tedy staženy společně s formulářem ze serveru. Metadata s přidanými vlastnostmi pro tagy a prioritu lze vidět na ukázce kódu 6.

[10] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer`

6.3.15 Filtrování

Filtrování je v našem případě úzce spjato s prioritou a tagováním, jelikož pomocí informací uložených v již zmíněném JSON souboru bude možné filtrovat stažené formuláře. Pro implementaci filtrování bude vytvořeno textové vstupní pole a Select komponenta[11]. Pomocí Select komponenty si uživatel vybere na základě jaké vlastnosti chce filtrovat. Vstupní pole bude sloužit pro určení vstupní hodnoty, dle které budou stažené formuláře filtrovány. Výsledkem vyfiltrovaných formulářů budou všechny formuláře, které obsahují vstupní řetězec.

6.3.16 Notifikace

Jak jsem již zmiňoval při psaní funkčního požadavku FP4.7, aktuálně neexistuje způsob vytvoření lokální notifikace, která by se uživateli ukázala v daný čas. Tudíž řešení této funkcionality by muselo být řešeno vyloženě skrze serverovou část aplikace. Pokud by uživatel chtěl nastavit notifikační událost, byl by vytvořen požadavek na server, který by obsahoval potřebné informace pro vytvoření push notifikace. Mezi potřebné informace může patřit název formuláře, id formuláře a čas notifikace. Server by následně obsahoval smyčku, která by kontrolovala všechny vytvořené notifikační události a v případě, že by se aktuální čas rovnal času nastavenému u notifikace, byl by vytvořen požadavek na službu push notifikací, která by následovně poslala push notifikaci do zařízení uživatele.

6.3.17 Instalace aplikace

Aby byla aplikace instalovatelná, je nutné aby aplikace obsahovala následující. V první řadě je třeba, aby byl k aplikaci přidán tzv. manifest. Manifest je JSON soubor, který obsahuje údaje o aplikaci, jako je její název, ikona, popis, startovní url a další. Příklad manifestu je k vidění na ukázce 7. Dále je nutné aby byla aplikace podávána přes HTTPS spojení a existoval Service worker, který umožňuje funkci offline.

[11] `https://mui.com/material-ui/react-select/`

Kapitola 7

Implementace

V této kapitole popíši jakým způsobem byly některé funkční a nefunkční požadavky implementovány. Nejdříve popíši implementaci změn v knihovně SForms, které bylo nutné implementovat k dosažení případu užití MONDIS. Dále se budeme zabývat implementací aplikace jako takové a jak bylo docíleno jednotlivých funkčních a nefunkčních požadavků.

7.1 SForms

Tato sekce popisuje mnou vytvořené změny v knihovně SForms, které bylo třeba implementovat pro uskutečnění alespoň základní funkcionality scénáře MONDIS.

7.1.1 Duplikace otázky

- Implementování požadavku FP3.9

Jako první jsem implementoval funkcionalitu duplikace otázky. K implementaci bylo zapotřebí doplnit SForms o zobrazování ikony sloužící k duplikaci

otázky. Kliknutí na tuto ikonu následně spustí proces duplikace otázky. Proces duplikace otázky nejdříve nalezne uzel rodiče a uzel prvního sourozence (pokud existuje) duplikované otázky. Následně změní ID, question_origin a relace všech uzlů duplikované otázky.

Dále se proces stará o změnu relace *has_preceding_question* u duplikované otázky na původní uzel otázky. Toto zapříčiní, že je otázka ve formuláři vykreslena hned po původní otázce. Následně je duplikovaná otázka přidána do pole podotázek rodičovského uzlu. V případě, že existuje sourozenec původní otázky, který je vykreslen hned po ní, je třeba změnit hodnotu relace *has_preceding_question* na duplikovanou otázku. Ukázka implementovaného kódu je k vidění na ukázce kódu 8.

7.1.2 Připojení souboru

- Implementování požadavků FP3.3, FP3.4 a FP3.5

K implementaci připojení souborů bylo nutné v knihovně SForms vytvořit novou komponentu pro odpověď (FileAnswer) a zároveň nový typ otázky, který umožní vykreslení komponenty FileAnswer. Vytvoření dané otázky bylo v celku přímočaré. Do knihovny SForms byla přidána konstanta *File*, která je využita ve formuláři pod vlastností *has_layout_class*. V případě, že je načtena otázka s touto vlastností, je využita komponenta FileAnswer k vykreslení komponenty pro odpověď.

Komponenta FileAnswer je poměrně jednoduchá a vykresluje maximálně 3 UI elementy, kterými jsou tlačítko umožňující nahrání souboru, vygenerovaný název nahraného souboru a tlačítko pro odstranění souboru. V případě, že není nahrán žádný soubor, je vykresleno pouze tlačítko pro nahrání. Naopak když je nahrán soubor, jsou vykresleny všechny 3 elementy.

Dále bylo nutné této komponentě přidat 3 metody. Těmito metodami jsou *getFile*, *onFileUpload* a *onFileDelete*. Funkcionalitu každé z těchto metod lze rozdělit na lokální funkcionalitu SForms a dále na specifickou funkcionalitu vývojáře. Funkcionalita SForms je konstantní a uživatel knihovny nad ní nemá kontrolu. Ovšem jak jsem zmínil v předešlém odstavci, existují různé způsoby jak může chtít vývojář pracovat se soubory. Vytvořil jsem tedy možnost definice vlastní funkcionality volané v každé z metod. Vývojář má možnost definování vlastní funkcionality (metody), kterou předá jako parametr při využití knihovny SForms.

- getFile
  - Tato metoda umožňuje získat informace o aktuálně nahraném souboru. Vrací objekt souboru, který obsahuje metadata, jako je název, typ a samotná data souboru.
  - V našem případě se jedná o načtení souboru z IndexedDB.
- onFileUpload
  - Tato metoda je určena pro manipulaci s nahranými soubory. Je spuštěna, když uživatel nahrává soubor pomocí komponenty FileAnswer. Vývojáři mohou tuto metodu rozšířit o vlastní funkcionalitu, jako je uložení souboru na server.
  - V našem případě se jedná o uložení souboru v IndexedDB.
- onFileDete
  - Tato metoda je spuštěna, když uživatel smaže soubor z komponenty FileAnswer. Stejně jako u metody onFileUpload, mohou vývojáři přidat vlastní funkcionalitu, například pro odstranění souboru ze serveru nebo aktualizaci seznamu dostupných souborů.
  - V našem případě se jedná o smazání souboru z IndexedDB.

Díky těmto novým funkcím mohou vývojáři, kteří integrují knihovnu SForms do svých aplikací, snadno implementovat funkcionalitu pro práci se soubory a zároveň ji přizpůsobit svým potřebám.

7.2 Aplikace

Tato sekce se zabývá vývojem samotné aplikace. V první řadě popíši jaké technologie byly využity k vytvoření aplikace. Následně popíši, jak bylo docíleno vytvoření PWA aplikace, se kterou se pojí využití aplikace v offline režimu a instalovatelnost. V poslední řadě se zaměříme na implementaci jednotlivých funkčních a nefunkčních požadavků.

7.2.1 Technologie

Aplikace byla vyvinuta za využití následujících technologií.

React

React je otevřená knihovna pro vývoj uživatelských rozhraní. Umožňuje vytvářet komplexní webové aplikace, které mohou být rychle aktualizovány a renderovány bez nutnosti obnovovat celou stránku. React je známý pro svůj koncept komponent, které poskytují modularitu a usnadňují znovupoužitelnost kódu. Také zavádí koncept virtuálního DOM (Document Object Model), který zvyšuje výkonnost tím, že minimalizuje přímé manipulace s DOMem prohlížeče.

TypeScript

TypeScript je typově silný, kompilovatelný jazyk. Jde o nadstavbu nad JavaScriptem, která přidává statické typy, třídy, rozhraní a moduly, což zlepšuje čitelnost a robustnost kódu. Kód napsaný v TypeScriptu je kompilován do čistého JavaScriptu, což znamená, že může být spuštěn v jakémkoli prostředí, které podporuje JavaScript.

React-Bootstrap

React-Bootstrap je knihovna pro frontendový vývoj, která kombinuje funkce Reactu a Bootstrapu. Poskytuje sadu reaktivních komponent, které jsou postaveny na Bootstrapu. React-Bootstrap umožňuje vývojářům vytvářet responzivní webové aplikace s konzistentním designem a předdefinovanými styly. Oproti klasickému Bootstrapu je React-Bootstrap plně kompatibilní s Reactem, jelikož nahrazuje JavaScriptové komponenty Bootstrapu za React komponenty.

Create React App

Create React App[1] je nástroj, který umožňuje rychlé zahájení vývoje nové React aplikace bez nutnosti ruční konfigurace nástrojů a závislostí. Tento nástroj generuje kostru projektu s optimalizovanými konfiguracemi pro Babel[2]

[1] `https://create-react-app.dev/`
[2] `https://babeljs.io/`

a Webpack[3], a také poskytuje testovací prostředí, server pro vývoj a skripty pro sestavení produkční verze aplikace. Create React App tedy umožňuje vývojářům soustředit se přímo na vývoj aplikace, místo aby museli nejprve nastavovat a udržovat vývojové prostředí.

7.2.2 Založení projektu

Při vytváření nového projektu byl využit nástroj Create React App s šablonou cra-template-pwa-typescript. Tato kombinace nástrojů mi umožnila rychle a efektivně vytvořit nový projekt, který už od začátku obsahoval všechny nezbytné prvky pro vývoj Progresivní webové aplikace v TypeScriptu.

Provedením příkazu *npx create-react-app my-app –template cra-template-pwa-typescript* v příkazovém řádku se nástroj Create React App postaral o vytvoření nového adresáře s názvem "my-app". Tento adresář obsahoval základní kostru projektu.

Výhodou šablony cra-template-pwa-typescript je, že již obsahuje všechny potřebné konfigurační soubory a skripty pro vytvoření PWA. Například Service worker, který je klíčový pro funkčnost PWA, byl již předem nastaven a připraven k použití.

7.2.3 Offline použití a instalovatelnost

Díky nástroji Create React App bylo vytvoření instalovatelné aplikace opravdu jednoduché. Jedinou potřebnou úpravou bylo změnit řádek v souboru **index.tsx**, kde se hodnota

serviceWorkerRegistration.unregister();

měnila na

serviceWorkerRegistration.register();.

[3] `https://webpack.js.org/`

Tato změna zajistí registraci a využití Service workeru v aplikaci. Dále díky Create React App a použité šabloně obsahuje Service worker potřebný kód pro přednačítání (pre-caching) potřebných souborů pro offline použití single page aplikace.

Dále, aby byla aplikace instalovatelná, je nutné, aby obsahovala soubor manifest. I v tomto případě nám Create React App pomohl a vytvořil projekt s tímto souborem. Zde pouze stačilo pozměnit jméno aplikace, a vše bylo připraveno.

7.2.4 Přihlašování

- Implementování požadavku FP1.1

Přihlášení probíhá ve dvou fázích. Nejdříve se na server odešle POST požadavek s uživatelskými údaji. V případě, že je uživatel přihlášen, odpověď na původní POST požadavek obsahuje nastavení cookie, která slouží k autorizaci uživatele pro aktuální session. Dále je odeslán GET požadavek pro stažení doplňujících informací o uživateli.

Uživatel není nucen přihlásit se, aby mohl aplikaci využívat, nicméně je nutné, aby se přihlásil v případě, že chce stáhnout nebo odeslat data na server. V případě, že chce uživatel stáhnout nebo odeslat data a není přihlášen, je uživateli zobrazeno informační okno, které obsahuje tlačítko pro přesměrování na obrazovku přihlášení. Po úspěšném přihlášení je přesunut na obrazovku, na které se nacházel naposledy.

Pro zjednodušení práce s daty formuláře byla využita knihovna *react-hook-form*. Tato knihovna nám umožňuje jednoduché čtení dat formuláře, jejich validaci a případné zobrazení chybové hlášky.

7.2.5 FP2 - Správa formulářů

V této sekci se budeme zabývat implementací některých funkčních požadavků ze skupiny FP2.

Stahování formulářů/záznamů

- Implementování požadavků FP2.1, FP2.2, FP2.7
- První část implementace FP2.3 a FP2.4

Stahování formulářů a záznamů jsou dvě rozdílné aktivity, a každá z nich má svou vlastní obrazovku, která se o danou aktivitu stará. Pro stahování formulářů existuje jednoduchý formulář, ve kterém uživatel vyplní název a vybere šablonu. Šablonu si uživatel vybírá z dropdown menu. Možné šablony jsou nejprve staženy ze serveru a následně jsou zobrazeny jako možné hodnoty v dropdown menu. Po odeslání formuláře se vytvoří POST požadavek na server, který vygeneruje formulář z vybrané šablony. Tento formulář je poté v podobě JSON-LD poslán jako odpověď.

Pro stažení záznamů má uživatel možnost stáhnout jeden nebo více záznamů současně. Obrazovka pro stahování záznamů obsahuje seznam dostupných záznamů. Stažení jednoho záznamu je možné pomocí tlačítka připojeného přímo k zobrazeným datům daného záznamu. Uživatel má také možnost přidat záznamy do dávkového stahování. Pokud uživatel přidá záznamy do dávkového stahování, na spodní navigační liště se zobrazí tlačítko, které spustí proces dávkového stahování.

Lokální ukládání formulářů/záznamů

- Druhá část implementace FP2.3 a FP2.4

Jak jsem již několikrát zmínil, formuláře i záznamy jsou ukládány do IndexedDB. Typeahead hodnoty jsou ukládány pomocí knihovny Workbox do CacheStorage. Každý formulář, respektive záznam, je rozdělen na čtyři samostatné entity v IndexedDB, a to FormMetadata, FormData, FormRecord a FormFile. Nejprve je vytvořen objekt FormMetadata, který obsahuje lokální název formuláře, tagy, prioritu a uuid klíč, pod kterým jsou uloženy FormData, FormRecord a FormFile. Definici FormMetadata je možno vidět na ukázce kódu 9.

Je možné ukládat vše do jediného záznamu, ale dle článku „Best Practices

for Using IndexedDB“[4] není dobré praktikovat tento způsob ukládání. Důvod je následující: IndexedDB API je sice asynchronní a tím pádem teoreticky nezatěžuje hlavní vlákno. Problém však spočívá v samotném ukládání dat. Nejdříve musí totiž hlavní vlákno vytvořit strukturovanou kopii dat, která je následně předána IndexedDB. Tudíž čím větší jsou data, tím větší je potřeba udělat kopii. To zase vede k větší blokaci hlavního vlákna.[28]

Pro uložení záznamů byla vytvořena struktura FormRecord, která představuje JSON objekt záznamu formuláře. Tento objekt mimo jiné slouží k odesílání odpovědí na server. Nejedná se totiž o plnohodnotnou strukturu JSON-LD. Definice objektu FormRecord je k nalezení na ukázce kódu 10. Můžeme si povšimnout, že tento objekt obsahuje vlastnost *question*. Tato vlastnost je objektem typu Question, který můžeme vidět na ukázce kódu 11. Ve zkratce se jedná o reprezentaci otázek a jejich odpovědí. Tato reprezentace se ve finále skládá pouze z ID otázky, originu a její případné odpovědi. Není zde obsaženo spoustu dalších vlastností, které jsou obsaženy v JSON-LD reprezentaci formuláře. Tím pádem se při odesílání na server přenáší menší množství dat.

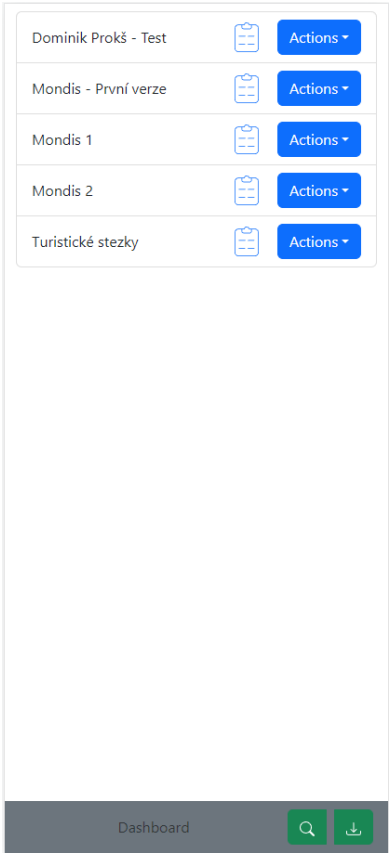
Struktura FormData obsahuje pouze klíč v podobě uuid a následně data v podobě JSON-LD souboru. Struktura FormFile se skládá z vygenerovaného uuid klíče, názvu souboru, typu souboru a dat v podobě ArrayBuffer.

Zobrazení stažených formulářů/záznamů

- Implementování požadavků FP2.5 a FP2.6

Pro zobrazení stažených formulářů a záznamů byla vytvořena jednoduchá obrazovka s názvem „Dashboard“. Lokálně uložené formuláře a záznamy jsou na této obrazovce reprezentovány jako řádek. Každý řádek obsahuje atribut lokální název formuláře a tlačítka pro otevření a další akce. Mezi další akce patří „Editace metadat“, „Odeslání na server“, „Duplikace formuláře“, „Smazání ze zařízení“ a „Export ve formátu JSON-LD“. Vizuální podoba této obrazovky je znázorněna na příslušném obrázku 7.1.

[4] `https://web.dev/indexeddb-best-practices/`



Obrázek 7.1: Dashboard obrazovka

Ukládání změn formuláře/záznamu

- Implementování požadavku FP2.8

Připomeňme si, že formulář je takový záznam, který neobsahuje odpovědi a není spojen s reálným objektem. Tudíž jakmile uživatel provede změny ve formuláři a uloží jej, stává se záznamem.

Ukládání změn formuláře a záznamu je implementováno pomocí reference, která je jedním ze vstupních parametrů (ref) SForms. Díky referenci máme možnost přistoupit k datům SForms. V referenci jsou pro nás důležité dvě funkce: getFormData a getData. Funkce getFormData vrací strukturu Question, která je zobrazena na ukázce kódu 11. Tato struktura je následně přidána do struktury FormRecord a uložena do IndexedDB.

Dále bylo nutné využít funkce getData, která vrací grafovou část struktury JSON-LD. Tato grafová část struktury obsahuje i uživatelem vyplněné odpovědi. Abychom dokázali uložit formuláře nebo záznam v původní podobě, jak byl původně, je nutné strukturu přetransformovat pomocí algoritmu „flatten“ z knihovny jsonld. K tomu zároveň potřebujeme „context“, aby algoritmus dokázal zkrátit URI uzlů a relací. Poté, co je struktura úspěšně zploštěna pomocí algoritmu, je uložena místo původní struktury v IndexedDB. Tímto jsme docílili lokálního ukládání změn při práci s datovou reprezentací formuláře. Při úspěšném uložení je hodnota „wasUpdated“ u struktury FormMetadata nastavena na true.

Kód, který se stará o ukládání změn (odpovědí) do lokální databáze zařízení je k vidění na ukázce kódu 12.

Odesílání na server

- Implementování požadavků FP2.9 a FP2.15

Odeslání záznamů na server se provádí přímo z obrazovky Dashboard. U každého uloženého záznamu má uživatel možnost rozbalit nabídku akcí pomocí dropdown tlačítka. Mezi těmito akcemi se nachází i odeslání na server pod názvem „“Upload. Tato akce má za následek odeslání záznamu na server k uložení změn.

K odesílání slouží struktura FormRecord. Pokud tato struktura neobsahuje vlastnost *key*, jedná se o nový záznam a je nutné po odeslání provést stažení vygenerovaných informací o záznamu ze serveru. Po úspěšném stažení je struktura FormRecord aktualizována a uložena do IndexedDB. V případě, že struktura již obsahuje prvek *key*, není nutné tuto akci provádět. Právě vlastnost *key* určuje, zda se jedná o nový záznam či nikoliv. Po úspěšném odeslání a stažení informací o záznamu je proces dokončen.

Dále je aplikace doplněna o automatické odesílání záznamů. Logika automatického odesílání se opírá o vlastnosti ze struktury FormMetadata, konkrétně *wasUpdated*, *lastServerUpload* a *downloadDate*. Vlastnost *wasUpdated* indikuje, zda byl záznam uživatelem upraven. Vlastnosti *lastServerUpload* a *downloadDate* jsou časové hodnoty a jak již názvy naznačují, *lastServerUpload* značí poslední nahrání na server a *downloadDate* značí čas, kdy byl záznam stažen do zařízení.

Algoritmus postupně projde všechny lokálně uložené záznamy a kontroluje, zda byly od posledního odeslání na server upraveny. V případě, že byl záznam upraven a čas posledního nahrání na server je delší než 30 dní, je záznam nahrán na server. Pokud záznam tuto hodnotu neobsahuje, je využita hodnota *downloadDate* jako fallback. Pokud byl záznam stažen před více než 30 dny a byl upraven, je odeslán k uložení na server.

7.2.6 FP3 - Práce s formulářem

Tato sekce popisuje jakým způsobem bylo implementováno vykreslování formulářů a záznamů pomocí knihovny SForms.

Vykreslení formuláře/záznamu

- Implementování požadavků FP3.1 a FP3.2

Jak jsem již několikrát zmiňoval, pro vykreslení stažených formulářů a záznamů je využita knihovna SForms. Využití této knihovny v kódu je k vidění na ukázce kódu 13.

Komponenta SForms přijímá hned několik vstupních parametrů. Pro nás jsou hlavně důležité parametry: form, ref, fetchTypeAheadValues, getFile, onFileUpload a onFileDelete.

Parametr form představuje formulář nebo záznam v podobě JSON-LD. Dalším parametrem je ref, který je typu Reference[5] a umožňuje získání dat z SForms. Díky tomuto parametru máme například přístup ke struktuře JSON-LD, která je průběžně aktualizována při vyplňování formuláře.

Existuje také parametr fetchTypeAheadValues, což je funkce, která se volá v případě, že uzel typu Question obsahuje typeahead hodnoty uložené na serveru. Tímto způsobem je implementováno stažení nebo poskytnutí typeahead hodnot.

[5] `https://react.dev/reference/react/useRef#reference`

V poslední řadě máme parametry getFile, onFileUpload a onFileDelete. Funkcionality těchto parametrů byly popsány v sekci 7.1.2 „Připojení souboru“.

Přidání GPS souřadnic

Navzdory vynaloženému úsilí a těsné spolupráci s tvůrcem rozšíření SForms GeoComponents Vojtěchem Luňákem se nám nepodařilo toto rozšíření úspěšně integrovat do aplikace. Během procesu integrace rozšíření SForms GeoComponents jsme se setkali s řadou problémů.

První potíže se týkaly vytváření instalačního balíčku rozšíření. Specificky jsme narazili na problém s bundlerem[6] Microbundle[7], který je rozšířením používán pro sestavování balíčku. Při důkladném pátrání po příčině jsme zjistili, že Microbundle ignoruje některé assety a nepřidává je do vytvořeného balíčku. Nicméně, našli jsme dočasné řešení - manuálně jsme přidali chybějící assety a v kódu upravili cestu k nim.

Druhým problémem byl error Invalid hook call[8]. Tento problém může nastat ze tří důvodů. Těmito důvody jsou:

1. rozdílné verze React a React-DOM,
2. porušování pravidel React hooks,
3. více než jedna kopie React v projektu.

Po pečlivém přezkoumání jsme zjistili, že žádný z dříve uvedených problémů se neprojevil. Nicméně, při spuštění aplikace v lokální vývojové verzi pomocí skriptu *react-scripts start* jsme narazili na nový problém. Ten spočíval ve špatném sestavování balíčku „React Leaflet“, který rozšíření SForms GeoComponents využívá pro vykreslení map. Konkrétně, problém vznikal během procesu sestavování a byl spojen s operátorem **??** (nullish coalescing operator[9]). Detailní popis tohoto problému lze nalézt v repositáři balíčku React Leaflet pod issue s číslem 877[10]. Problém je možné vyřešit změnou

[6] `https://snipcart.com/blog/javascript-module-bundler`
[7] `https://github.com/developit/microbundle`
[8] `https://legacy.reactjs.org/docs/error-decoder.html/?invariant=321`
[9] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing`
[10] `https://github.com/PaulLeCam/react-leaflet/issues/877`

konfigurace sestavovacího nástroje Webpack, který Create React App používá k sestavování aplikace. Nicméně, Create React App standardně neumožňuje uživatelům měnit tuto konfiguraci. Je tedy nutné do aplikace přidat další balíčky, které by umožnily tuto změnu. Využití těchto balíčků však narušuje integritu Create React App a může vést ke špatnému chování aplikace.

Řešením problému konfigurace je například využití balíčku „customize-cra“[11], který umožňuje změnu konfigurace Create React App.

Bohužel jsme neměli dostatek času na nalezení kompletního řešení a úspěšnou integraci mapové komponenty do aplikace. Na druhou stranu se nám podařilo zprovoznit formulář MONDIS, který jsem vytvořil, v aplikaci Vojtěcha Luňáka. Tato aplikace zobrazuje mapovou komponentu. Ukázku funkcionality lze nalézt na adrese `https://deploy-preview-11--s-forms-geo-components.netlify.app/?path=/story/sforms-geo-component--mondis`.

7.2.7 FP4 - Usnadnění

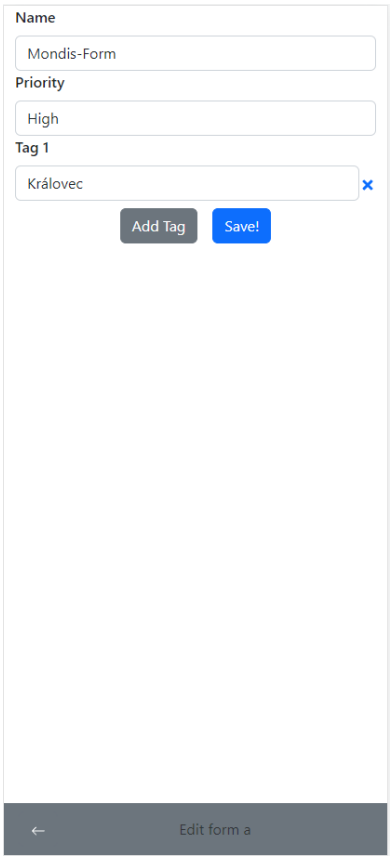
V této sekci se budeme zabývat implementací některých požadavků ze skupiny FP4.

Změna lokálních dat (Metadat)

- Implementování požadavků FP4.4, FP4.5 a FP4.8

Aby mohl uživatel manipulovat s lokálními hodnotami (metadaty) formuláře, respektive záznamu, byla vytvořena jednoduchá obrazovka, která umožňuje uživateli nastavovat právě tyto hodnoty. Vzhled obrazovky pro změnu lokálních dat je možné vidět na obrázku 7.2. Uživatel má v tomto okně možnost změnit lokální název formuláře (záznamu), prioritu a přidávat a odebírat tagy. Na základě všech těchto vlastností je možné provádět filtrování mezi formuláři a záznamy.

[11] `https://github.com/arackaf/customize-cra`

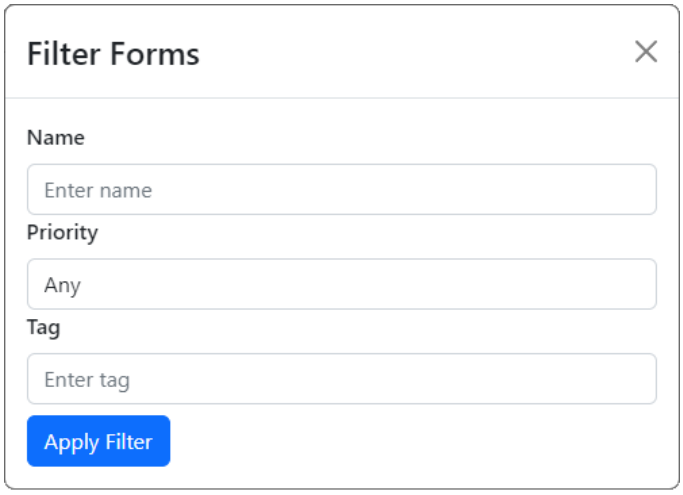


Obrázek 7.2: Obrazovka pro změnu metadat formuláře/záznamu

Filtrování

- Implementování požadavků FP4.1 a FP4.3

K filtrování mezi staženými formuláři a záznamy bylo vytvořeno modální okno, kde uživatel může nastavit hodnoty, podle kterých chce provádět filtrování. Funkce filtrování je znázorněna na ukázce kódu 14. Uživatel má možnost filtrovat na základě lokálního názvu, priority a tagů. Pokud uživatel ve filtrovacím modálním okně nevyplní některou z vlastností, je tato vlastnost při filtrování ignorována. Vzhled modálního okna pro filtrování je zobrazen na obrázku 7.3.



Obrázek 7.3: Modální okno pro filtrování mezi staženými formuláři a záznamy

Kapitola 8

Evaluace

V této kapitole se budeme zabývat evaluací vytvořené aplikace pomocí uživatelského testování. Cílem této kapitoly je zhodnotit, jak dobře funkce a design aplikace splňují potřeby uživatelů a jak jsou vhodné pro použití v reálném světě.

8.1 Uživatelské Testování

V rámci evaluace jsme provedli uživatelské testování se třemi vybranými uživateli pomocí specifických scénářů. Tyto scénáře byly navrženy tak, aby pokryly různé oblasti naší aplikace, konkrétně navigaci v aplikaci, use case MONDIS, offline využití a instalovatelnost. Cílem těchto testů bylo zhodnotit uživatelskou přívětivost a funkčnost naší aplikace v různých situacích. Využité scénáře a dotazník s odpovědmi uživatelů jsou k nalezení v příloze C. Během testování jsem byl v telefonickém spojení se dvěma ze tří testerů.

8.1.1 Scénář 1: Navigace

Testování scénáře 1 se zabývalo navigací uživatele v aplikaci. Scénář sloužil k otestování, zda je užívání aplikace intuitivní, či nikoliv. Během testování tohoto scénáře uživatelé nahlásili následující chyby.

Ch1 Prázdná obrazovka Dashboard po otevření aplikace, což vedlo ke zmatení uživatelů

Ch2 Obrazovka stahování ukazovala chybové hlášení, jelikož uživatel nebyl přihlášen, což vedlo ke zmatku

Ch3 Použití přepínačů mezi stahováním formulářů a záznamu na stránce stahování je neintuitivní

Ch4 Po úspěšném importování formuláře se neresetoval formulář pro další import a informační toast měl příliš krátkou dobu trvání

Ch5 Neintuitivní umístění jednotlivých obrazovek

Ch6 Prázdná obrazovka Dashboard pokud filtrování nenašlo žádný stažený formulář nebo záznam

Ch7 Neintuitivní názvy akcí v dropdown tlačítku

8.1.2 Scénář 2: MONDIS

Tento testovací scénář byl zaměřen i na otestování případu užití MONDIS. Uživatelé měli k dispozici formulář k importování do aplikace, který následně alespoň z části vyplnili. Formulář byl vytvořen nástrojem Semantic Form Editor[1]. Zároveň formulář obsahoval potřebné prvky pro připojení souborů k formuláři.

Jelikož se mi nepovedlo integrovat rozšíření SForms GeoComponents, testovací scénář nehodnotil využití mapové komponenty a využití GPS. Nicméně je důležité v budoucnu tuto komponentu implementovat pro správné použití MONDIS. Během testování tohoto scénáře narazili uživatelé na následující problémy.

Ch8 Vygenerované ID souboru se zobrazovalo mimo obrazovku.

Ch9 Pozice tlačítka pro odstranění souboru byla neintuitivní.

Ch10 Přechody mezi záložkami formuláře byly pomalé.

Ch11 Během uploadu záznamu na server nebyla žádná indikace o průběhu procesu.

[1] `https://semantic-form-editor.vercel.app/`

8.1.3 Scénář 3: Offline použití

Tento scénář zkoumal, zda mají uživatelé problém s instalováním nebo offline funkcionalitou aplikace. Během testování narazili uživatelé na následující chyby.

Ch12 Schází zmínka o tom, že se uživatel nachází v offline režimu

Ch13 Nenačítají se multimediální soubory, které slouží jako doplňky pro otázku

Doplňkové multimediální soubory se liší od těch, jejichž nahrávání jsme implementovali. Jejich hlavní účel ve formuláři je poskytnout uživatelům potřebné informace k otázkám. Například, představte si situaci, kdy lékař musí vyplnit formulář týkající se identifikace fraktury. Ve formuláři uvidí rentgenový snímek, který mu pomůže identifikovat frakturu. Tento rentgenový snímek je příkladem doplňkového souboru. Důležité je poznamenat, že doplňkové soubory nejsou odpověďmi na otázky v rámci formuláře. SForms knihovna zobrazuje tyto doplňkové soubory ve formuláři pomocí HTML elementu iframe[2].

8.1.4 Odstraněné chyby

Po skončení uživatelského testování a identifikování chyb, byly odstraněny následující chyby.

Ch1 Obrazovka Dashboard nyní obsahuje text, který uživatele informuje o tom, že aktuálně nemá stažený žádný formulář nebo záznam.

Ch2 Chybové hlášení bylo změněno na informativní (změněna barva)

Ch4 Resetování formuláře pro importování formulářů po úspěšném nahrání.

Ch6 Obrazovka Dashboard byla doplněna o informační text, v případě kdy filtr nenalezne formulář nebo záznam.

Ch7 Názvy akcí byly upraveny, aby lépe reflektovaly zamýšlenou funkcionalitu.

- změna „Edit“ na „Edit attributes“

[2] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe`

- změna „Upload“ na „Upload to server“

Ch8 Oprava UI komponenty FileAnswer v SForms. Text této komponenty je nyní ohraničen a nemůže přetéct mimo ohraničené pole.

Ch9 Tlačítko pro odstranění souboru bylo přesunuto.

Ch11 Při nahrávání záznamu je uživatel informován o prováděné akci za využití točícího se kolečka.

8.1.5 Shrnutí uživatelského testování

Aplikace byla otestována třemi uživateli na třech testovacích scénářích. Testování zkoumalo, zda se uživatel dokáže zorientovat v aplikaci bez prvotní pomoci. Dále bylo testováno zda je aplikaci možné využít pro případ užití MONDIS a v poslední řadě bylo testováno, zda aplikace funguje v offline režimu a je schopna být nainstalována do zařízení a fungovat jakožto mobilní aplikace. Každý scénář se zaměřoval na jinou funkcionalitu aplikace. Avšak scénáře se do jisté míry prolínaly.

Testování odhalilo celkově 13 chyb. Převážně se jednalo o chyby v UI designu, které zhoršovali uživatelskou zkušenost s aplikací. Testování ukázalo značný nedostatek v uživatelské zkušenosti z hlediska nedostatečného informování uživatele o prováděných akcích. Dále testování ukázalo značnou chybu v návrhu rozložení obrazovek aplikace. Aktuální rozložení není zcela intuitivní a je nutné jej předělat.

Z nalezených 13 chyb bylo 8 chyb odstraněno. Odstranění zbytku chyb představuje budoucí práci, jelikož na jejich řešení nezbyl čas.

8.1.6 Evaluace vůči původní aplikaci MONDIS

Uživatelé mají možnost aplikaci využít pro případ užití MONDIS, avšak mohou čelit některým výzvám. Jednou z nich je absence mapy a vyhledávací funkce z ÚSKP, což může komplikovat použití. Kromě toho mohou uživatelé s menšími mobilními zařízeními narazit na problémy při vyplňování sekce „Posouzení stavu“, a to především kvůli nevyhovujícímu mobilnímu UI designu knihovny SForms. Jako dočasné řešení je nahrazena mapa v sekci lokalizace textovými poli. Uživatelé mohou do těchto polí vkládat potřebné informace,

jako je adresa a zeměpisná šířka a délka. I přes tyto obtíže však aplikace nabízí možnost svého využití pro případ užití MONDIS.

V poslední řadě by bylo dobré knihovnu SForms doplnit přímo o komponentu, která by se specifikovala přímo na vykreslení sekce „Posouzení stavu“. Tato komponenta by zahrnovala funkcionality, které byly zahrnuty v původní aplikaci MONDIS. Mezi chybějící funkcionality můžeme zařadit například:

- smazání duplikované otázky,
- označení otázky jako zodpovězené,
- inteligentní výběr z nabídky (intelligent tree select),
- zobrazení otázek v závislosti na odpovědi na určitou otázku (podmíněné vykreslení).

8.2 Budoucí práce

V první řadě je nutné odstranit chyby, které se naskytly během vývoje a nebyly odstraněny z časových či jiných důvodů. Takovéto identifikované chyby jsou následující.

Ch3 Použití přepínačů mezi stahováním formulářů a záznamu na stránce stahování je neintuitivní

Ch5 Neintuitivní umístění jednotlivých obrazovek

Ch10 Přechody mezi záložkami formuláře byly pomalé.

Ch12 Schází zmínka o tom, že se uživatel nachází v offline režimu

Ch13 Nenačítají se multimediální soubory, které slouží jako doplňky pro otázku

Dále je třeba doplnit integraci mapové komponenty SForms GeoComponents, která umožní vyhledávání na mapě. Tato komponenta by navíc mohla být rozšířena o možnost používání mapy offline, což zatím není implementováno.

Kromě toho je také potřeba implementovat funkcionality MONDIS, které jsem zmínil v předchozí sekci (8.1.6). Možným řešením je vytvoření specifické komponenty pro sekci „Posouzení stavu“, která by zahrnovala zbylé

funkcionality MONDIS. Pokud tato komponenta nebude implementována, měly by být některé z těchto funkcionalit, které lze využít obecně, jako je mazání duplikované otázky, intelligent tree select nebo označení otázky jako zodpovězené, začleněny přímo do knihovny SForms.

Kapitola 9

Závěr

Tato diplomová práce se zaměřila na vývoj mobilní aplikace pro sběr dat v terénu s využitím sémantických formulářů. Cílem bylo poskytnout uživatelům flexibilní a přizpůsobitelné nástroje pro sběr dat, které by mohly být využity v offline prostředí a umožnily připojení multimediálních souborů. Výsledek práce představuje kombinaci pečlivého výzkumu, návrhu a implementace, která byla následně otestována a vyhodnocena.

V kapitole o technologickém pozadí byly nastíněny základní technologie a nástroje potřebné pro vývoj aplikace. To poskytlo základ pro pochopení technické struktury projektu a také kontext pro další kapitoly. Následně byla provedena detailní analýza existujících řešení. Tato analýza ukázala, že žádné z komerčních řešení neumožňuje vytváření sémantických formulářů, což potvrdilo potřebu této práce.

V následující kapitole byla provedena volba technologie pro vývoj aplikace. Mezi možnostmi hybridní, nativní a webové aplikace byla jako nejlepší možnost vybrána webová aplikace PWA. Tento výběr byl podložen porovnáním jednotlivých možností a zohledněním specifických požadavků projektu, včetně potřeby offline použití a připojení multimediálních souborů.

Analýza byla provedena pro identifikaci funkčních a nefunkčních požadavků, jakož i pro vytvoření případů užití a diagramů aktivit. Tato analýza poskytla základ pro návrh řešení, které bylo následně s lehkými výhradami implementováno. Implementace spočívala ve vytvoření mobilní aplikace s využitím vybraných technologií a knihovny SForms pro vykreslení formulářů.

V posledním kroku bylo provedeno uživatelské testování a evaluace aplikace. Tato fáze poskytla cenné zpětné vazby, které umožnily vylepšit návrh a funkčnost aplikace. Získané zkušenosti a poznatky mohou být využity pro další vývoj a zlepšení aplikace.

V závěru, tato studie představuje pokrok v oboru terénního sběru dat prostřednictvím sémantických formulářů. Navzdory některým omezením, jako je závislost na specifických technologiích a potřeba dalšího průzkumu v určitých oblastech, práce nabízí užitečný základ pro další výzkum v tomto poli a potenciálně odemyká nové možnosti pro vývoj a inovace v oblasti mobilních aplikací pro sběr dat.

9.1 Budoucí práce

Následné kroky v rozvoji tohoto projektu obnáší několik klíčových úprav. První z nich je integrace mapového rozšíření SForms GeoComponents do aplikace, což poskytne uživatelům možnost vyhledávání na mapě. Současně je nezbytné zařadit do SForms GeoComponents offline režim, což umožní flexibilnější využití aplikace.

Dále je třeba zlepšit UI design knihovny SForms tak, aby lépe vyhovoval potřebám mobilních aplikací a byl přívětivější pro uživatele.

Navíc by bylo dobré vyvinout konkrétní komponentu pro případ užití MONDIS. Tato komponenta bude zodpovědná za vykreslení sekce „Posouzení stavu“ a rozšíří tak knihovnu SForms o funkcionality MONDIS, které momentálně chybí.

Kromě toho bude zásadní implementace mechanismu pro cachování doplňkových multimediálních souborů, což zvýší výkonnost aplikace a zefektivní využití prostředků uživatele.

Tato zlepšení jsou klíčové pro další rozvoj a zdokonalení projektu.

Literatura

[1] Riccardo Cacciotti, Jaroslav Valach, Petr Kuneš, M. Čerňanský, Miroslav Blasko, and Petr Kremen. Monument damage information system (mondis): An ontological approach to cultural heritage documentation. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-5/W1:55–60, 07 2013.

[2] Ing. Tomáš Klíma. Semantic form editor. Master's thesis, České vysoké učení technické v Praze, 2021.

[3] W3C. Rdf - semantic web standards. `https://www.w3.org/RDF/`. Navštíveno: 9.10.2022.

[4] W3C. Son-ld 1.1. `https://www.w3.org/TR/json-ld11/`. Navštíveno: 9.10.2022.

[5] Web workers api - web apis | mdn. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API`. Navštíveno: 3.5.2023.

[6] Service worker api - web apis | mdn. `https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API`. Navštíveno: 3.5.2023.

[7] The service worker lifecycle. `https://web.dev/service-worker-lifecycle/`. Navštíveno: 3.5.2023.

[8] Web share api - web apis | mdn. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Share_API`. Navštíveno: 4.5.2023.

[9] Geolocation api - web apis | mdn. `https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API`. Navštíveno: 4.5.2023.

[10] Notifications api - web apis | mdn. https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API. Navštíveno: 4.5.2023.

[11] Notification behavior. https://web.dev/push-notifications-notification-behaviour/. Navštíveno: 4.5.2023.

[12] Push api - web apis | mdn. https://developer.mozilla.org/en-US/docs/Web/API/Push_API. Navštíveno: 4.5.2023.

[13] Push notifications overview. https://web.dev/push-notifications-overview/. Navštíveno: 4.5.2023.

[14] Ing. Petr Křemen, Ph.D. Systém mondis. http://www.mondis.cz/web/portal/mondis-system. Navštíveno: 8.10.2022.

[15] Surveymonkey. https://surveymonkey.com/. Navštíveno: 8.10.2022.

[16] Surveymonkey survey analysis tools - turning insights into action. https://www.surveymonkey.com/mp/analyze/. Navštíveno: 20.2.2023.

[17] Google forms. https://www.google.com/forms/about/. Navštíveno: 8.10.2022.

[18] Jotform. https://www.jotform.com/. Navštíveno: 8.10.2022.

[19] What are progressive web apps? https://web.dev/what-are-pwas/. Navštíveno: 13.11.2022.

[20] Architectural overview of cordova platform - apache cordova. https://cordova.apache.org/docs/en/latest/guide/overview/index.html. Navštíveno: 13.11.2022.

[21] Flutter - build apps for any screen. https://flutter.dev/. Navštíveno: 13.11.2022.

[22] Mui: The react component library you always wanted. https://mui.com/. Navštíveno: 20.4.2023.

[23] React-bootstrap · react-bootstrap documentation. https://react-bootstrap.github.io/. Navštíveno: 20.4.2023.

[24] Storage quotas and eviction criteria - web apis | mdn. https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Storage_quotas_and_eviction_criteria. Navštíveno: 25.4.2023.

[25] Cachestorage - web apis | mdn. https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage. Navštíveno: 25.4.2023.

[26] Pete Lepage. Storage for the web. https://web.dev/storage-for-the-web/. Navštíveno: 3.1.2023.

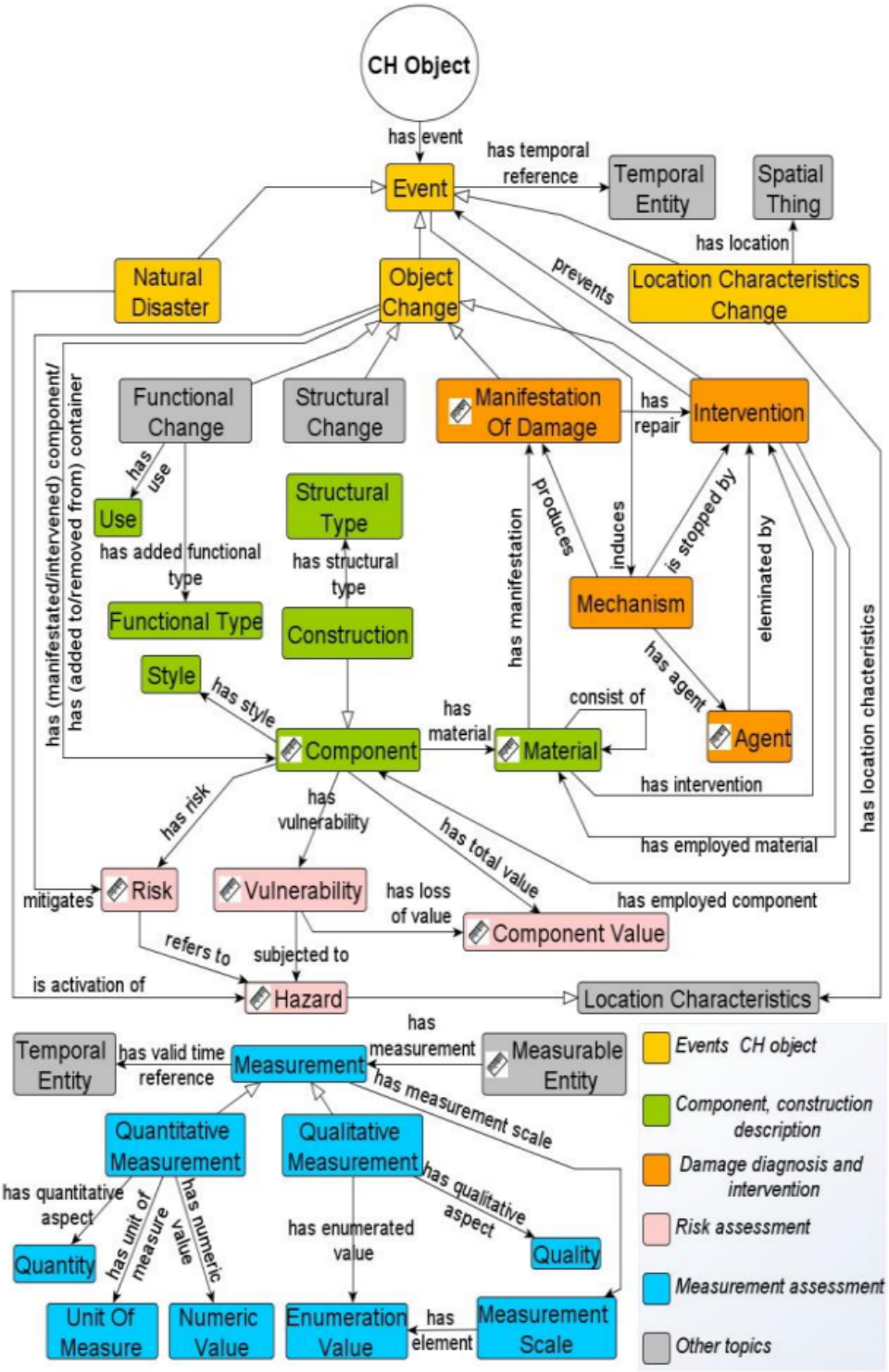
[27] Indexeddb api - web apis | mdn. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. Navštíveno: 25.4.2023.

[28] Best practices for using indexeddb. https://web.dev/indexeddb-best-practices/. Navštíveno: 25.4.2023.

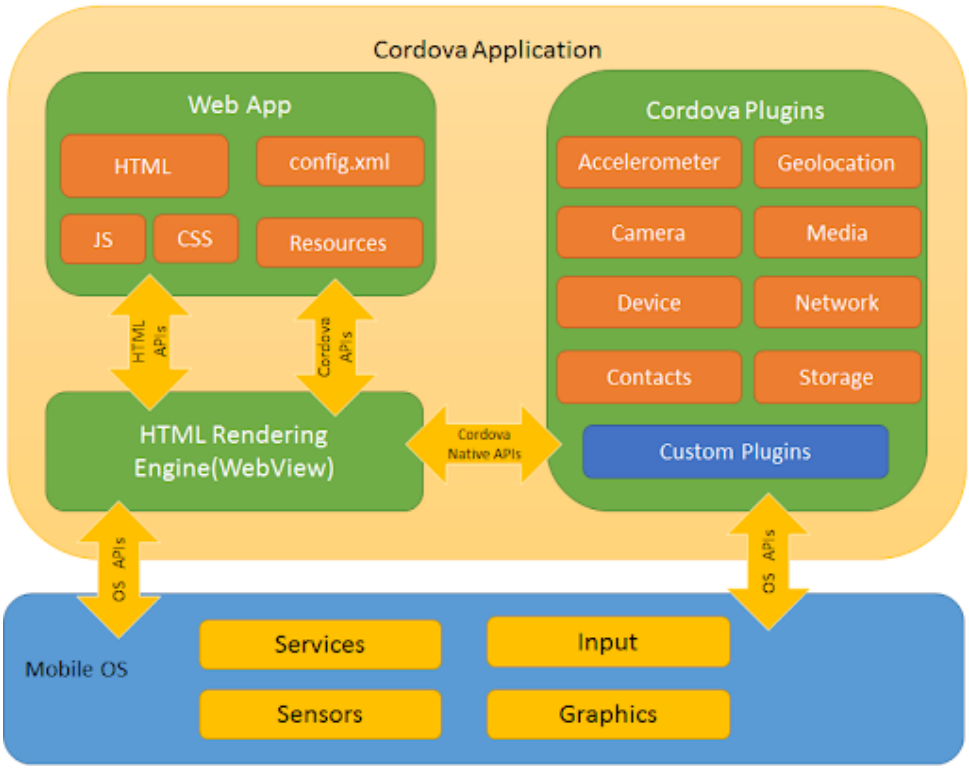
Příloha A

Obrázky a diagramy

A.1 Obrázky



Obrázek A.1: MONDIS ontologický model [1]



Obrázek A.2: Cordova struktura, převzato z `https://cordova.apache.org/docs/en/10.x/guide/overview/`

A.2 Diagramy

Diagram A.3: Případy užití MONDIS

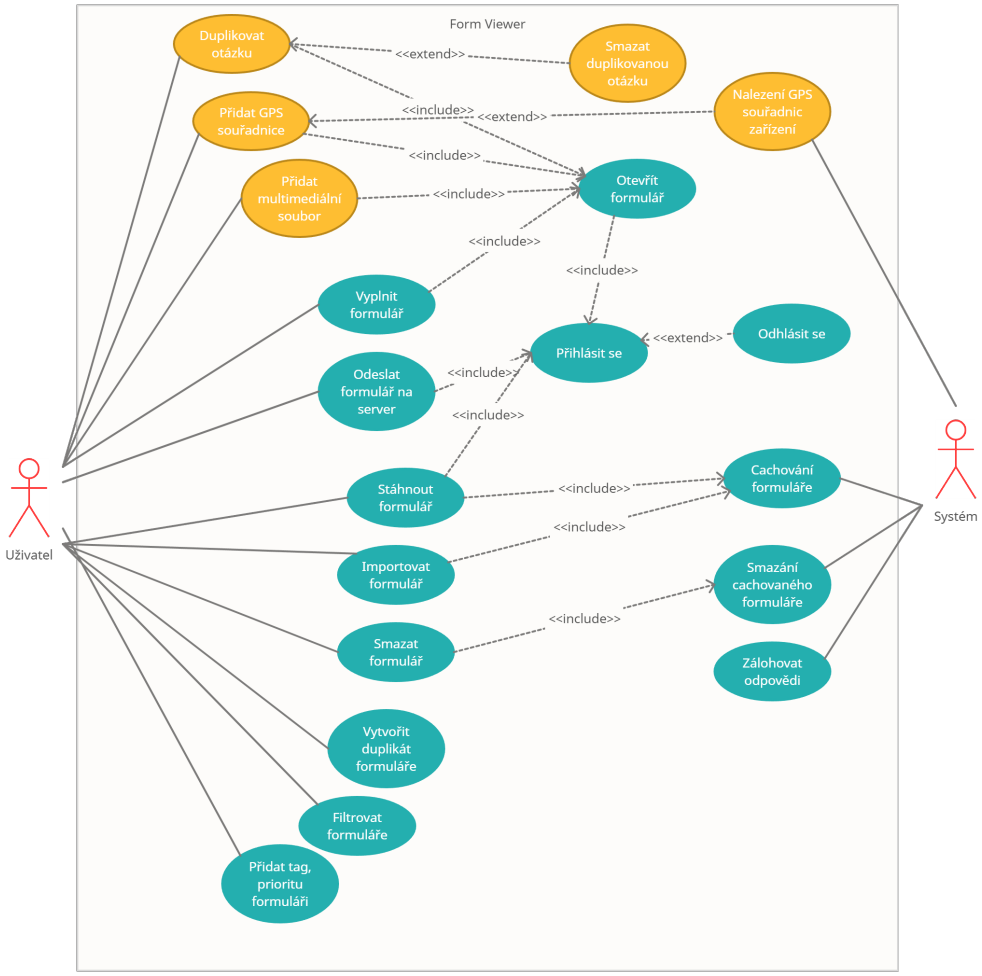


Diagram A.4: Případy užití aplikace

Příloha B

Ukázky kódu

```
[
  {
    "@id": "http://me.markus-lanthaler.com/",
    "http://xmlns.com/foaf/0.1/name": [
      { "@value": "Markus Lanthaler" }
    ],
    "http://xmlns.com/foaf/0.1/homepage": [
      { "@id": "http://www.markus-lanthaler.com/" }
    ]
  }
]
```

Kód 1: Ukázka souboru JSON-LD v expandované serializaci

```json
{
  "@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/0.1/homepage",
      "@type": "@id"
    }
  },
  "@id": "http://me.markus-lanthaler.com/",
  "name": "Markus Lanthaler",
  "homepage": "http://www.markus-lanthaler.com/"
}
```

Kód 2: Ukázka souboru JSON-LD v kompaktní serializaci

```json
[{
  "@id": "http://me.markus-lanthaler.com/",
  "http://xmlns.com/foaf/0.1/name": [
    { "@value": "Markus Lanthaler" }
  ],
  "http://xmlns.com/foaf/0.1/knows": [
    { "@id": "_:b0" }
  ]
}, {
  "@id": "_:b0",
  "http://xmlns.com/foaf/0.1/name": [
    { "@value": "Dave Longley" }
  ]
}]
```

Kód 3: Ukázka souboru JSON-LD ve zploštělé serializaci

```
registerRoute(({url}) =>
    url.pathname.includes('possibleValues'),
    new StaleWhileRevalidate({cacheName: 'forms-possible-values'})
);
```

Kód 4: Ukázka využití knihovny Workbox

```
<http://onto.fel.cz/file/ABCD> a form:multimedia.
<http://onto.fel.cz/file/ABCD> form:has-file-type "JPG".
```

Kód 5: Příklad řešení trojice pro multimediální soubory

```
[
  {
    "id": "1",
    "name": "Proper name 1",
    "tags": ["MONDIS", "Kaplice"],
    "priority": "medium"
  },
  {
    "id": "2",
    "name": "Another proper name",
    "tags": ["CTU", "Multimedia"],
    "priority": "low"
  },
  {
    "id": "3",
    "name": "Mondis",
    "tags": [],
    "priority": "high"
  }
]
```

Kód 6: Příklad struktury Metadata s prioritou a tagováním

```
{
  "name": "js13kGames Progressive Web App",
  "short_name": "js13kPWA",
  "description": "Progressive Web App that lists games submitted to the
    A-Frame category in the js13kGames 2017 competition.",
  "icons": [
    {
      "src": "icons/icon-32.png",
      "sizes": "32x32",
      "type": "image/png"
    },
    // ...
    {
      "src": "icons/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/pwa-examples/js13kpwa/index.html",
  "display": "fullscreen",
  "theme_color": "#B12A34",
  "background_color": "#B12A34"
}
```

Kód 7: Příklad manifest souboru

```
const cloneQuestion = (question) => {
    const graphCopy = {...data}
    // deep copy
    const questionCopy = JSON.parse(JSON.stringify(question));
    // object that holds the mapping of new and old node IDs
    const nameMap = {};

    // recursive change of node ID and question origin,
    // adds node's new and old ID into the nameMap object
    Utils.copyNode(questionCopy, nameMap);
    // change IDs of relations so that they contain the new IDs of nodes
    Utils.renamePrecedingQuestionRelations(questionCopy, nameMap);

    questionCopy[Constants.HAS_PRECEDING_QUESTION]
        = {'@id':question['@id']};

    const parent = Utils.findParent(graphCopy.root, question);
    const parentSubquestions
        = Utils.asArray(parent[Constants.HAS_SUBQUESTION])

    // find first subsequent sibling of original question
    // through the has\_preceding\_question relation
    const nextQuestion = parentSubquestions.find((q) => {
        const precedingQuestion = q[Constants.HAS_PRECEDING_QUESTION];
        if(!precedingQuestion || !precedingQuestion['@id']) return false;

        return precedingQuestion['@id'] === question['@id'];

    });

   // append duplicate to parent's subquestions array
    parentSubquestions.push(questionCopy);

    if(nextQuestion)
        nextQuestion[Constants.HAS_PRECEDING_QUESTION]
            = {'@id':questionCopy['@id']};

    updateData(data, parent);
}
```

Kód 8: Implementace duplikace otázky

```
export interface FormMetadata {
    dataKey: string;
    priority?: Priority;
    name:string;
    tags?:string[];
    description?:string;
    lastServerUpload?:number;
    downloadDate?:number;
    wasUpdated:boolean;
    hasRecord: boolean;
}
```

Kód 9: Reprezentace objektu FormMetadata

```
export interface FormRecord {
    uri?: string;
    key?: string;
    formTemplate?: string;
    localName: string;
    author?: Author;
    dateCreated?: number;
    lastModified?: number;
    lastModifiedBy?: Author;
    institution?: Institution;
    question?:Question;
}
```

Kód 10: Reprezentace objektu FormRecord

```
export interface Question{
    subQuestions: Question[];
    answers: Answer[];
    origin:string;
    originPathId: string;
    types: string[]
}
export interface Answer{
    textValue:string;
    codeValue:string;
    origin:string;
    types:string[];
    uri?: string;
}
```

Kód 11: Reprezentace objektů Question a Answer

```
const getFormData: React.MouseEventHandler<HTMLButtonElement> =
  async (e: React.MouseEvent<HTMLButtonElement>) => {
  const formRefValue = formRef.current;
  if (formRefValue) {
    console.log(formRef);
    const formData:Question = formRefValue.getFormData();
    const formRoot = formRefValue.context.getData();
    const recordUpdate = {...record, question: formData};

    await setInDB(FORMS_RECORDS_STORE, uuid as string, recordUpdate);

    if(formRoot.hasOwnProperty('root')){
      const graph = formRoot['root'];
      const fqData = await jsonld.flatten(graph, CONTEXT_CONSTANT);
      await setInDB(FORMS_DATA_STORE, uuid as string, fqData);
    }
  }
}
```

Kód 12: Kód ukládání změn formuláře/záznamu

```
<SForms form={form}
            ref={formRef}
            options={SFormsOptions}
            fetchTypeAheadValues={fetchTypeAheadValues}
            mappingRule={GeoComponents.mappingRule}
            loader={<Spinner animation={"border"}/>}
            enableForwardSkip={true}
            getFile={onGetFile}
            onFileUpload={onFileUpload}
            onFileDelete={onFileDelete}
     ></SForms>
```

Kód 13: Využití knihovny SForms v kódu

```
const filterForms = (name: string, priority: Priority, tag: string) => {
  console.log(name, priority, tag);
  setForms((prevForms) => {
    return prevForms.filter((entry) => {
        const entryData: FormMetadata = entry.value;
        const boolName = !name ? true
            : entryData.name?.includes(name) ?? false;
        const boolPriority = priority === Priority.DEFAULT
            ? true : entryData.priority === priority;
        const boolTag = !tag ? true
            : entryData.tags?.includes(tag) ?? false;

        return boolName && boolPriority && boolTag;
      });
  });
  setIsFiltering(true);
}
```

Kód 14: Kód filtrování mezi formuláři a záznamy

Příloha C

Uživatelské testování

C.1 Scénář 1: Navigace

Tento scénář slouží k otestování uživatelské schopnosti orientovat se v aplikaci. Cílem je zjistit jestli existují části aplikace, ke kterým se uživatel těžko dostává nebo není schopen se dostat.

C.1.1 Postup

1. Otevřít aplikaci (https://viewer.prukes.cz/)
2. Přejít na stažení formulářů/záznamů
3. Přejít na importování formulářů
4. Importovat alespoň 2 formuláře, mohou být stejné
5. Přejít na Dashboard (stránka se staženými formuláři a záznamy)
6. Přejít na editaci atributů u vybraného formuláře
7. Změnit libovolný atribut
8. Otevřít okno vyhledávání (filtrace) stažených formulářů
9. Filtrovat dle vámi změněného atributu

10. Zkontrolovat, že mezi zobrazenými formuláři je i vámi pozměněný formulář

C.1.2 Otázky

1. Měli jste problém s nalezením stránky importování formulářů?
2. Měli jste problém nalezením editace atributů formuláře?
3. Měli jste problém nalezením filtrování mezi staženými formuláři?
4. Objevil se jiný problém během průchodu scénářem? Pokud ano, jaký?
5. Změnili byste některé věci, ať už se jedná o názvy jednotlivých stránek nebo rozpoložení UI elementů? Pokud ano, jaké?
6. Napadá vás způsob jak zlepšit navigaci aplikací?
7. Přijde vám umístění jednotlivých stránek logické? Lze se zaměřit hlavně na stránku importování.

C.2 Scénář 2: MONDIS

Tento scénář slouží k evaluaci scénáře užití MONDIS. Scénář zkoumá, zda má uživatel možnost sběru dat stejně jako v aplikaci MONDIS. Dále scénář hledá zda existují nějaké nedostatky v implementaci scénáře MONDIS. Prerekvizitou pro tento scénář je stažený formulář MONDIS v zařízení. Případně mít formulář v podobě JSON-LD souboru, který je možné importovat. Další prerekvizitou je připojení k internetu.

C.2.1 Postup

1. Otevřít formulář pro scénář MONDIS
2. Vyplnit sekci Identifikace
3. Přejít na sekci Lokalizace
4. Vyplnit tuto Lokalizace

5. Přejít na sekci Posouzení stavu
6. Přidat soubor k formuláři
7. Duplikovat otázku pro připojení souboru
8. Připojit fotografii
9. Vyplnit libovolná pole
10. Uložit formulář
11. Přejít zpět na Dashboard
12. Znovu otevřít právě upravený formulář
13. Zkontrolovat, že vámi vyplněné odpovědi jsou obsaženy ve formuláři
14. Přejít zpět na Dashboard
15. Odeslat formulář k uložení na server

C.2.2 Otázky

1. Měli jste problém s vyplněním sekce Identifikace?
2. Měli jste problém s vyplněním sekce Lokalizace?
3. Měli jste problém s vyplněním sekce Posouzení stavu?
4. Měli jste problém s přidáním souboru?
5. Obsahoval formulář vámi vyplněné údaje poté, co jste jej uložili a znovu otevřeli?
6. Objevil se jiný problém během průchodu scénářem? Pokud ano, jaký?

C.3 Scénář 3: Instalovatelnost a offline použití

Tento scénář testuje funkcionalitu aplikace jakožto plnohodnotné mobilní aplikace. Scénář testuje instalovatelnost na zařízení a offline použití.

C.3.1 Postup

1. Otevřít aplikaci ve webovém prohlížeči
2. Nainstalovat aplikaci
   - Každý internetový prohlížeč má možnost instalace jiným způsobem. V případě prohlížeče Google Chrome klikněte na 3 tečky v pravém horním rohu. Kliknutí by mělo otevřít seznam dalších možností. Zde najděte a klikněte na tlačítko „Nainstalovat do zařízení“.
3. Nalezněte aplikaci ve vašem zařízení
4. Zapněte nalezenou aplikaci přímo z vašeho zařízení
5. Pokud nemáte stažený formulář, stáhněte jej nebo importujte
6. Odpojte zařízení od internetového připojení
7. Otevřete jakýkoliv stažený formulář
8. Vyplňte jej libovolným způsobem
9. Uložte formulář
10. Zkontrolujte, zda byly odpovědi zaneseny do formuláře (opětovné otevření)

C.3.2 Otázky

1. Měli jste problém s instalací?
2. Měli jste problém s offline použitím?
3. Očekávali jste nějakou funkcionalitu, která nebyla přítomna?
4. Další poznámky.

Příloha D

Návod k nasazení

Tato kapitola popisuje návod k nasazení, resp. lokálnímu spuštění, vytvořené aplikace.

D.1 Prerekvizity

- Docker
- git

D.2 Server

Pro úplné fungování aplikace byl použit server OFN Record Manager, který je k dispozici na adrese `https://github.com/blcham/record-manager`. Pro spuštění serveru je třeba využít postup, který je přístupný na adrese `https://github.com/blcham/vfn-data-management`.

D.3 Mobilní aplikace

Aplikaci je možné stáhnout na adrese `https://github.com/Prukes/RDF-form-viewer`. Nejjednodušší postup pro spuštění aplikace je využití Dockeru. Aplikace obsahuje Dockerfile společně s docker-compose.yml.

D.3.1 Postup

Pro spuštění aplikace pomocí Dockeru postupujte následovně.

1. stáhnout repositář pomocí *git clone https://github.com/Prukes/RDF-form-viewer.git*
2. přejít do složky ./form-viewer
3. upravit adresu serveru v souboru .env
   - Tento soubor obsahuje proměnnou REACT_APP_API_URL. Tato proměnná je dále v aplikaci použita jako základní (base) URL adresa serveru.
4. spustě příkaz *docker compose –env-file=.env up* v adresáři ./form-viewer
5. aplikace by měla být spuštěna na adrese localhost a portu 10000 (localhost:10000)

Příloha E

Obsah elektronické přílohy

- `source_codes` ..... adresář se zdrojovými kódy
  - `form-viewer` ..... zdrojový kód mobilní aplikace
  - `sforms` ..... zdrojový kód SForms s mými úpravami
  - `README.md` ..... instalační návod
- `Dominik_Proks_diplomova_prace.pdf` PDF dokument diplomové práce