Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Dynamické formuláře ve webových aplikacích

Bakalářská práce

Illya Balabanov

Školitel: PhDr. Miloš Prokýšek, Ph.D.

České Budějovice 2017

Balabanov, I., 2017: Dynamické formuláře ve webových aplikacích. [Dynamic forms in web applications. Bc. Thesis, In Czech.] – 29p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Bakalářská práce se zaměřuje na analýzu existujících nástrojů pro dynamickou definici a generování UI formulářů. Součástí této práce je funkční řešení webové aplikace s dynamicky generovaným obsahem. Také tato práce obsahuje dokumentaci k softwaru a příklady zdrojového kódu. Software je vyvíjen na základě požadavků Biologického centra AV ČR.

Abstract

This bachelor's thesis focuses on analysis of existing tools for dynamic definition and generation of UI forms. Working solution of web application with dynamic content generation is the component of this thesis. Also, this thesis includes software documentation and source code examples. Software is developed upon request of Biology Center CAS.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 11. 12. 2017 ……..

Illya Balabanov

Poděkování

Chtěl bych poděkovat panu PhDr. Miloši Prokýškovi PhD. za odborné vedení práce, cenné rady a vstřícnost při konzultacích.

Obsah:

1 Úvod

V dnešní době je technický rozvoj mnohem rychlejší, něž kdykoliv předtím, a to se týká i webu. Webové stránky už nejsou pouze statické a umožňují sdílení obsahu a interaktivitu mezi návštěvníky webu. To znamená, že web není postaven jen na obsahu, který vytvoří jeho majitel, ale i na obsahu, který vytváří komunita uživatelů. Vývojář webových aplikacích tedy musí počítat s interaktivitou a navrhovat uživatelské rozhraní tak, aby uživatel mohl intuitivně a bez překážek vytvářet vlastní datové prvky.

V úvodu je vysvětlen pojem "dynamický formulář" a k čemu je používán. Bude demonstrováno, že základní struktura formuláře není popsaná ve zdrojovém kódu stránky, ale je ovlivnitelná z klientské strany. Uživatel má například možnost přidat více než jednou adresu v kontaktním formuláři. Dynamické formuláře nabízí velmi jednoduchý způsob, jak snadno a rychle vytvořit sofistikovaný interaktivní online formulář, který lze následně použít pro definici vlastních datových objektů. Hlavní výhoda spočívá v tom, že tento úkon zvládne i netechnicky vzdělaný uživatel.

Tato bakalářská práce je zaměřená zejména prakticky. Jelikož se jedná o „proof-of-concept", autorovým hlavním úkolem bude provést analýzu již existujících nástrojů a vyzkoušet metodu dynamického generování formulářů v praxi. Dalším důležitým cílem je na základě analýzy navrhnout a implementovat funkční řešení uživatelského rozhraní pro práci s daty z Bioarcheologické databáze ČR. Výstupem této práce je softwarové řešení, které následně budou používat pracovníci Biologického centra AV ČR.

2 Předběžná studie

2.1 Popis problematiky

Jak už bylo zmíněno v úvodu, tato bakalářská práce se zabývá problematikou dynamických formulářů. Tento způsob tvorby webových formulářů není obvyklý, protože struktura zpracovávaných dat je tvůrcům aplikace předem známá. Klasickým příkladem jsou přihlašovací nebo kontaktní formuláře. Tyto nevyžadují žádnou interaktivitu ze strany uživatelů. Statická data mají pevnou strukturu a posílají se na server. Ale existují případy, kdy data nemají pevnou strukturu a vývojář předem neví, jak má formulář vypadat.

Něž se začne fáze vývoje, je potřeba zvážit výhody a nevýhody dynamického přístupu. K nesporné výhodě patří možnost dynamicky definovat webový formulář a následně používat pro ukládaní dat pro různé typy záznamu. Hlavní nevýhodou je složitá implementace (bez použití už existujících nástrojů) a obtížné zpracování dat, které nemají pevnou strukturu.

V následující kapitole bude provedena analýza existujících nástrojů a volba nejlepšího řešení.

2.2 Rešerše existujících nástrojů

Ještě před analýzou samotných nástrojů na vytváření dynamických formulářů, je obecně nutné zvolit framework, v rámci kterého bude vyvíjen samotný projekt. Jednou z možností je AngularJS – komplexní nástroj, který obsahuje všechno, co potřebuje programátor pro vývoj webových aplikacích. AngularJS má i své nevýhody – programátor v tomto případě využije jen malou část ze všeho, co tento framework nabízí. Proto byl zvolen modernější nástroj - React.js. Jedná se o poměrně nový nástroj, který má na starosti práci s DOMem[1] a oddělení modelu od jeho vizuální reprezentace. Podrobněji je o ReactJS pojednáno v kapitole 4 (Steigerwald, 2014).

[1] DOM – je rozhraní, které umožňuje pomoci jazyka JavaScript přistupovat k prvkům webového dokumentu.

V tabulce 1 jsou uvedeny výsledky analýzy. Předem nutno říct, že vzhledem k velkému počtu knihoven, byly vybrány pouze ty, které jsou napsány v jazyce JavaScript a jsou open-source. Následující seznam obsahuje kritéria, která jsou při volbě knihovny důležitá:

1. Otevřenost zdrojového kódu a vhodná licence.
2. Kompatibilita se stávajícím frameworkem (pokud je využíván).
3. Rozsáhlá dokumentace a use-case příklady.
4. Funkčnost.
5. Podpora ze strany vývojářů a komunity.

Na základě rešerše bylo zjištěno, že zvolit knihovnu, která by splňovala všechny výše uvedené požadavky, není jednoduchá záležitost. Výsledky analýzy jsou zachyceny v tabulce 1. Během analýzy byl vyzkoušen vetší počet knihoven a nástrojů, něž je uveden v tabulce, ale vzhledem k velkému počtu a nerelevantnosti byly vybrány jen některé z nich.

Po této analýze autor rozhodl, že nejlepším řešením pro zamýšlenou aplikaci bude knihovna, která se jmenuje “formBuilder“. Tento balíček obsahuje velmi rozsáhlou dokumentaci, sadu návodu na použití a byl snadno implementován do projektu.

Tabulka 1: Rešerše existujících nástrojů

Název	Technologie	Dokumentace	Funkcionalita	Komunita a funkčnost	Odkaz
Formbuilder	JavaScript, CoffeScript	Základní příklady použití, demo ukázka, návod na instalaci pro vývoj.	Poskytuje implementaci grafického rozhraní na vytváření formulářů a následné exportování ve formátu JSON	Poslední commit 26.09.2017 – knihovna se dál nevyvíjí. 94 otevřených issues na GitHubu.	https://github.com/dobtco/formbuilder
formBuilder	JavaScript, jQuery	Velmi bohatá a rozsáhlá. Obsahuje hodně demo příkladů a každý případ použití je detailně popsán. Veškera dokumentace je dostupná na http://formbuilder.readthedocs.io/en/latest/	Plně funkční drag-and-drop nástroj. Poskytuje jak možnost vytváření, tak i renderování na základě již existujících formulářů.	Knihovna se neustále vyvíjí. Komunita je velká. Existuje chat na Gitteru, kde uživatel vždy dostane odpověď.	https://formbuilder.online https://github.com/kevinchappell/formBuilder http://formbuilder.readthedocs.io/en/latest/
React-form-builder	JavaScript, React.js	Dokumentace je dostačující.	Obsahuje FormBuilder a FormRender. Snadno přizpůsobitelný.	Komunita není velká. Knihovna není funkční, protože chybí tam závislosti. Evidentně knihovna se	https://github.com/blackjk3/react-form-builder
Angular-form-builder	JavaScript, AngularJS, CoffeeScript	Dokumentace není textová, ale obsahuje pouze příklady kódu.	Pouze vytváření formulářů. Podporuje možnost drag-and-dropu.	Poslední commit byl v roce 2014.	https://github.com/kelp404/angular-form-builder

Zdroj: Vlastní zpracování

3 Analýza a specifikace

3.1 Funkční požadavky

1. Aplikace bude umožňovat dynamicky vytvářet schémata (definice) webových formulářů.
2. Aplikace bude ukládat schémata do výzkumné databáze.
3. Aplikace bude schopna vykreslit formulář podle uživatelských schémat.
4. Aplikace bude umožňovat vytvářet, ukládat, mazat, filtrovat a odstraňovat záznamy.
5. Aplikace bude zobrazovat seznam všech nálezů.
6. Vzhledem k velkému počtu dat aplikace bude umožňovat stránkovat mezi záznamy.

3.2 Nefunkční požadavky

1. Jazyk – Klientská část aplikace bude napsána v jazyce JavaScript.
2. Dostupnost – Aplikace bude přístupná pomocí internetového prohlížeče.
3. Výkonost – Zpracování velkých objemů dat.

3.3 Logický rámec

Tabulka 2: Logický rámec projektu

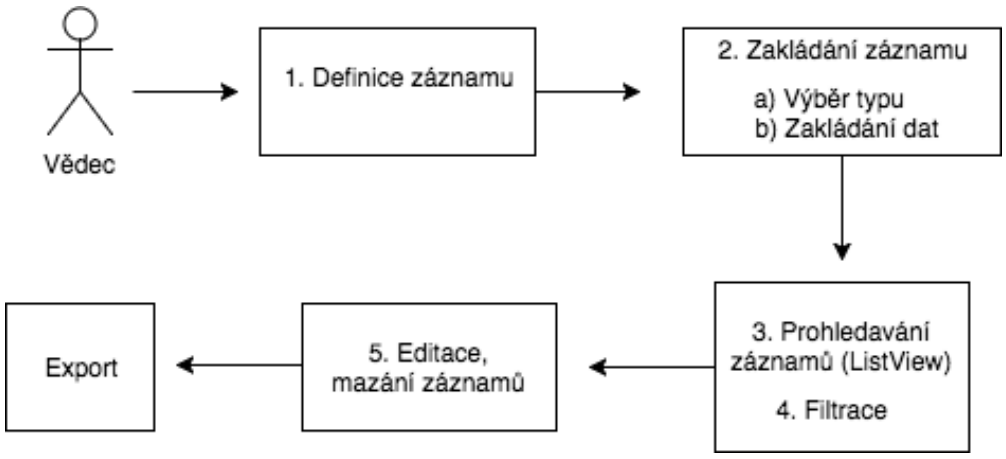
Popis projektu	Objektivně ověřitelné ukazatele	Prostředky ověření	Předpoklady
Záměr projektu: Usnadnit a zrychlit zpracování bioarcheologických nálezů.	Aplikace je kladně oceněná pracovníky a je trvalé využívaná.	Provozní statistiky systému. Slovní vyjádření a odezva od pracovníků Biologického centra AV ČR.	
Cíl projektu: Vytvořit intuitivní uživatelské rozhraní pro práce s daty. Umožnit snadné vyhledávaní a filtraci podle vybraných kritérií.	Aplikace je využívaná vědeckými pracovníky.	Dokumentace kódu. Výsledky testovacího procesu.	Stručně a jasně stanovené požadavky na systém.
Výstupy: Webové rozhraní, které umožňuje manipulovat s daty. Nástroj na vytváření dynamických formulářů. Databázová vrstva.	Provozuschopný a funkční systém na podzim 2017	Dokumentace projektu. Výsledky manuálních testů.	Zájem o systém ze strany zákazníků.
Klíčové činnosti: Zprovoznění databáze. Návrh a implementace UI. Sepsání dokumentace. Testování kódu.	Jasně definovaný plán vývoje. Dostatečná kvalifikace programátorů. Projektová dokumentace.	Leden 2017 - analýza požadavků a výběr technologií. Duben–červen 2017 - vývoj aplikací a psáni dokumentaci. Červenec 2017 - testování.	Výběr vhodných technologií. Spolupráce se zákazníky. Dobře navržená architektura.

Zdroj: Vlastní zpracování

3.4 Scénáře použití aplikace

Na základě funkčních požadavků je možné znázornit scénář použití, který je uveden na obrázku 1.

Obrázek 1: Scénář použití aplikace



Zdroj: Vlastní zpracování

Jednotlivé kroky pro využití systému jsou popsány níže.

3.4.1 Definice záznamu

1. Uživatel stiskne tlačítko “Build” v horní části stránky.
2. Objeví se modální okno, které obsahuje plugin na definici formulářů.
3. Metodou “Drag and Drop” uživatel si vytvoří formulář.
4. Schéma se uloží do databáze stisknutím tlačítka “Save”.

3.4.2 Zakládání záznamu

1. Uživatel stiskne tlačítko “Create” v horní části stránky.
2. Objeví se modální okno, které obsahuje formulář statické části a selectbox pro výběr dynamické části.
3. Uživatel si nejprve vyplní statickou část a pak si vybere příslušný dynamický formulář, který následně systém vykreslí.
4. Data se uloží do databáze stisknutím tlačítka “Save”.

3.4.3 Prohledávaní, editace a mazaní

1. Uživatel bude moct prohledávat záznamy v seznamu.
2. Kliknutím na příslušnou ikonku se dá smazat záznam.
3. Kliknutím na příslušnou ikonku se dá editovat záznam.

4 Návrh systému

4.1 Přehled vybraných technologií

Pro vývoj moderní webové stránky už rozhodně nestačí jen HTML, CSS a Vanilla JavaScript[2]. Front-end vývoj se posunul neskutečně dopředu. Projekty se automatizují, verzují a obvykle se využívají frameworky, které poskytují už předefinované rozhraní a řeší za programátora většinu rutinních problémů. Možností je velmi mnoho, ale ne všechny jsou vhodné pro daný účel. Proto je výběr správné technologie klíčovou věcí ve fázi návrhu. V následujících kapitolách budou zvoleny, ohodnoceny a doporučeny vhodné technologie.

4.1.1 JavaScript React.js

JavaScript je interpretovaný programovací jazyk, který se používá při tvorbě webových aplikací. Umožňuje vývojářům vytvářet mnohem interaktivnější webové stránky a interagovat s uživateli. JavaScript je de-facto standard v oblasti vývoje webových aplikací, proto je evidentní, proč je používán v projektu. Další důležitá věc je výběr JavaScript frameworku. Framework usnadňuje programování a slouží jako podpora pří vývoji softwaru. Obvykle je zaměřen na vyřešení rutinních problémů a šetří čas programátorům.

React.js je framework umožňující uživatelům tvorbu interaktivních uživatelských rozhraní. Webová uživatelská rozhraní jsou spouštěna přímo u návštěvníka v jeho internetovém prohlížeči, s výpočetní logikou na straně serveru (backend) komunikují prostřednictvím REST (Representational State Transfer) služeb. React.js pracuje s programovacím jazykem JavaScript. Výhody React.js:

1. Jednodušší kód. Nejzásadnějším důvodem je to, že React zjednodušuje kód a vede k celkově racionálnějšímu kódu a jednoduššímu debuggování.
2. Rychlost. JavaScript aplikace obsahují několik stovek DOM elementů. Modifikace domu po každé změně může být velmi pomalá, proto má React svou vlastní, zjednodušenou verzi pod názvem VirtualDOM[3] a používá algoritmus, který aplikuje na skutečný DOM pouze nezbytné změny.

[2] Vanilla JavaScript – JavaScript bez použití knihoven a frameworků.

[3] Virtual DOM – je virtuální strom, reflektující strukturu skutečného DOMu.

3. Podpora a komunita. K dnešnímu dni (červen 2017) má 8346 commitů, 67 500 stargazerů a 935 contributorů a je tak jedním z nejaktivnějších repozitářů na GitHubu.[4]

4. JSX. Syntaktický cukr[5], který umožňuje zápis HTML tagů přímo do zdrojového kódu JavaScript.

4.1.2 Node.js a Express.js

Node.js je cross-platformní běhové prostředí JavaScriptu, které běží na stráně serveru. Je třeba podotknout, že Node.js samo o sobě není server, ale umožňuje tvorbu vlastního HTTP serveru nebo jiných síťových služeb. Node.js také umožňuje přístup k souborovému systému a dalším nástrojům, pomocí kterých může programátor provádět spoustu dalších operací v systému. Proč se dá použit Node.js a jaké výhody přináší?

Výhody Node.js:

- Asynchronní volání – provádění operací je neblokující. To znamená, že nemusíme čekat na výsledek, aby se zahájila další operace.
- Rychlost - Node.js je založen na V8 JavaScript Engine od Google, což umožňuje velmi rychlé spouštění a zpracování kódu.
- NPM (Node Package Manager) - nástroj, který podporuje instalaci a aktualizaci opakovaně použitelných modulů z online repozitáře. Zabývá se také vyřešením závislostí mezi moduly.

Jak už bylo zmíněno, Node.js není server, a proto k vytvoření jednoduchého HTTP serveru se používá Express.js. Express je minimalistický a flexibilní webový aplikační framework pro Node.js, který poskytuje sadu funkcí pro vývoj webových a mobilních aplikací. Jedná se o open source technologii. Dané řešení je potřebné pro návrh API pro CRUD[6] aplikace.

[4] Odkaz na GitHub – https://github.com/facebook/v

[5] Syntaktický cukr – konstrukcí programovacího jazyka, které mají za účel zpřehlednit kód.

[6] CRUD – zkratka pro Create, Read, Update a Delete.

4.1.3 NoSQL. MongoDB

NoSQL je soubor konceptů, které umožňují rychlé a efektivní zpracování dat se zaměřením na výkon, spolehlivost a agilnost. Z toho důvodu zkratku NoSQL vysvětlujeme spíše Not-only-SQL, než no-SQL. NoSQL není názvem konkrétního produktu, ale označuje nový databázový koncept, nový trend, kterým se mnoho databázových systémů vydalo. Samotné NoSQL databázové systémy nevylučují využívání jazyka SQL. Přínos NoSQL databází spočívá v tom, že přinášejí nové myšlenkové řešení do databázových systémů. Základní vlastnosti, které jsou totožné pro většinu NoSQL databází:

- nevyužívají relační model (neexistuje přesná struktura v záznamech)
- jsou dostupné open-source
- jsou horizontálně škálovatelné (přidávání dalších prvků systému)
- podpora pro replikaci

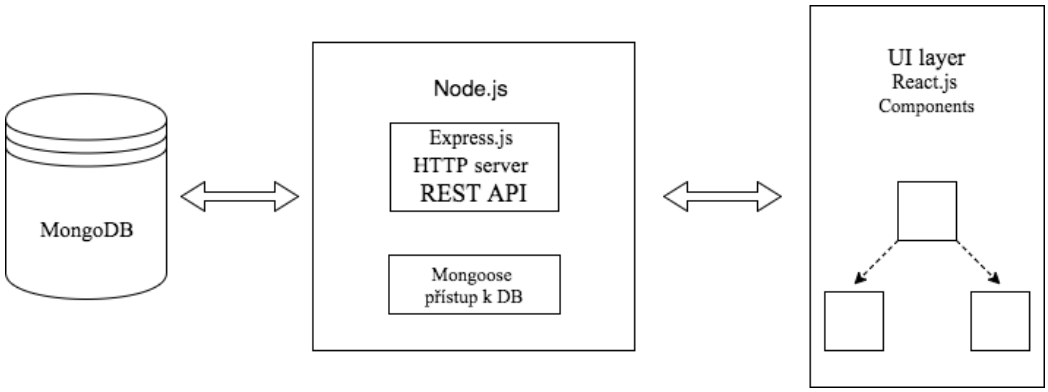
Důvody použití na konkrétním projektu budou podrobněji popsány v kapitole 4.3.

V současném světě existují desítky různých NoSQL databázových systémů. Mezi nejpoužívanější patří například MongoDB, Redis a Memcached, ElasticSearch nebo Apache Cassandra. Naším reprezentantem NoSQL databází bude právě MongoDB, které poslouží jako stavební pilíř pro navržené řešení v této práci.

MongoDB je dokumentově orientovaná databáze se zaměřením na flexibilní ukládání dat, horizontální škálovatelnost a snadnost použití. „S MongoDB se komunikuje přes JavaScript a data si databáze ukládá ve formátu BSON (Binary JSON). V MongoDB máme databáze, které však obsahují kolekce, které dále obsahují libovolný počet dokumentů“. Samotné jádro je implementováno v jazyce C, ale existují k ní ovladače v mnoha běžně používaných jazycích (Mrozek, 2012).

4.2 Architektura systému

Obrázek 2: Architektura systému



Zdroj: Vlastní zpracování

Při vývoji aplikace bude použit tzv. MERN stack.

MERN (MongoDB, Express, React, Node.js) je sada technologií, která se využívá především pro snadný vývoj dynamických webových aplikací. Je to velmi populární nástroj a je propagován hlavně společností Facebook. Hlavní výhodou je že klientská a serverová část je napsána v jazyce JavaScript. Na obrázku 2 je vidět použití výše uvedených technologií a jak komunikují mezi sebou.

Pro realizaci tohoto projektu autor zvolil MERN, a to je z důvodu, že bych chtěl získat nové zkušenosti s frameworkem React.js. Dalším důvodem je široká podpora, velké množství návodů online a výborně fungující ekosystém jazyka JavaScript.

Aplikace je rozdělená na dvě části: serverová a klientská. Nabízí se, aby pro vývoj byla použita velmi populární architektura MVC – ta se však v této situaci přímo aplikovat nedá. React.js je v podstatě písmenko “V” v kombinaci MVC. To znamená, že React.js je jen knihovna, která se používá pro tvorbu uživatelského rozhraní.

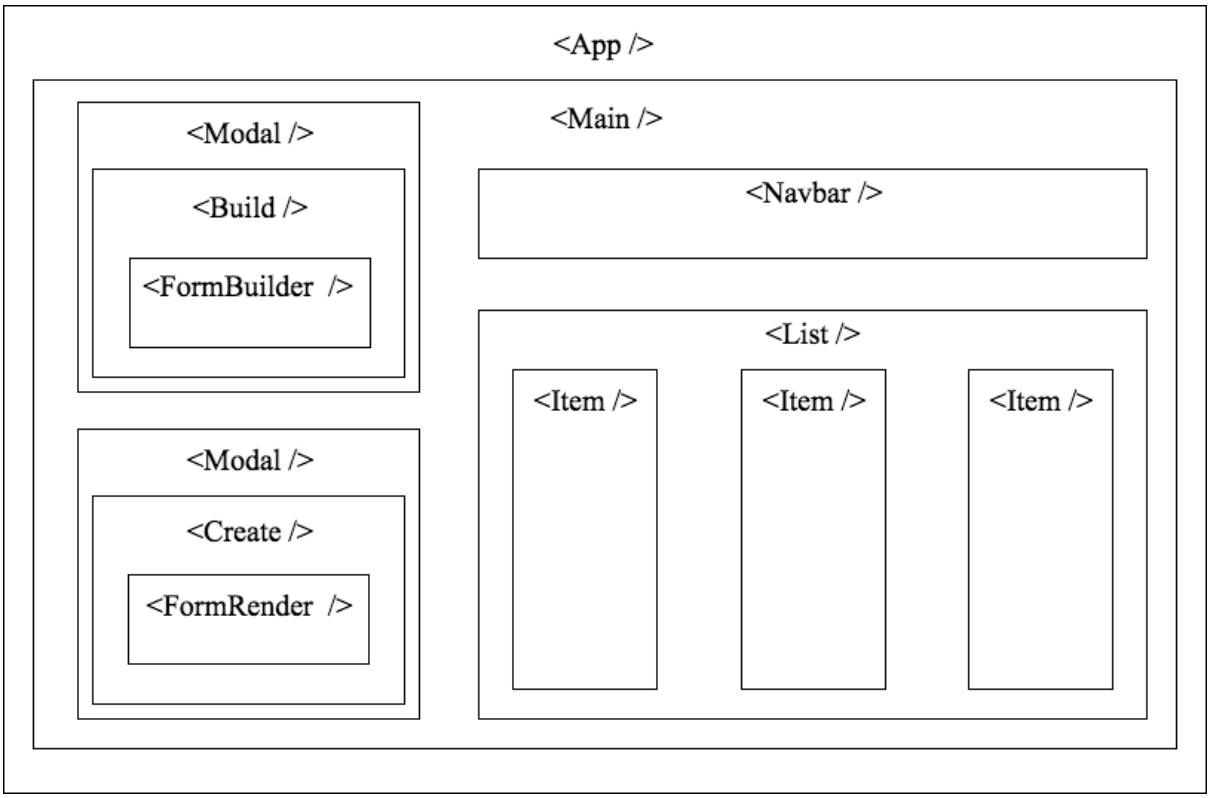
4.3 Serverová část

Na backendu běží Node.js server s využitím Express.js frameworku, který slouží pro vytvoření jednoduchého HTTP serveru. Server je samotné běžící instance a v ideálním případě by neměl o klientovi vůbec nic vědět. Klient bude komunikovat se serverem prostřednictvím REST API.

4.4 Klientská část

Klientská část je napsána v jazyce JavaScript s využitím React.js a dalších pomocných knihoven. Jedná se o tzv. “Single page application”. Aplikace je rozdělená na jednotlivé komponenty, které odpovídají grafickému návrhu. Komponenty v React.js je možné vnořovat, základní kompozice je popsána na obrázku 3.

Obrázek 3:Schéma rozložení komponentů



Zdroj: Vlastní zpracování

4.4.1 Komponenty

App

Komponenta App je bodem vstupu aplikace. Obsahuje definici API cest.

Main

Komponenta Main je hlavní obrazovka aplikace. Obsahuje všechny další pomocné komponenty. Také je zodpovědná za otevíraní a zavírání modálních oken.

List

Komponenta List vykreslí seznam všech záznamu a také implementuje stránkovaní.

Item

Komponenta Item představuje jednotlivý záznam, který je rozdělen na statickou a dynamickou část.

Build

Komponenta Build obsahuje jQuery plugin na interaktivní vytváření formulářů. Uživatel bude mít možnost definovat vlastní formulář a uložit ho do databáze.

Create

Komponenta Create umožní vytvořit nový záznam na základe schématu z databáze.

4.5 Databázový model

Jak už bylo řečeno, pro realizaci databáze bude použita MongoDB. Bioarcheologická data mají specifický tvar, kde je nutné ukládat jeden objekt (nález) jako celek ve formátu JSON. V daném případě NoSQL je ideální řešení. V podstatě je potřeba vytvořit v databázi jen dvě kolekce: jedna bude sloužit pro ukládaní schémat, a druhá bude obsahovat samotná data. Na obrázku 4 je vidět, jak může vypadat vzorový dokument z těchto kolekcí.

Obrázek 4: Definice schématu

```
{
    "_id" : ObjectId("59a28b4797dbd2ec85ecb573"),
    "name" : "archeoBot",
    "definition" : [
        {
            "subtype" : "text",
            "name" : "vzorek",
            "className" : "form-control",
            "label" : "Vzorek",
            "type" : "text"
        },
        {
            "subtype" : "text",
            "name" : "botdruh",
            "className" : "form-control",
            "label" : "Botanický druh",
            "type" : "text"
        },
        {
            "subtype" : "text",
            "name" : "botnik",
            "className" : "form-control",
            "label" : "Botanik",
            "type" : "text"
        }
    ],
    "__v" : 0
}
```

Zdroj: Vlastní zpracování

Na obrázku 5 je patrné, že data jsou rozdělená na statickou a dynamickou část. Obsah klíče “dynam” tvoří uživatel a může být libovolný.

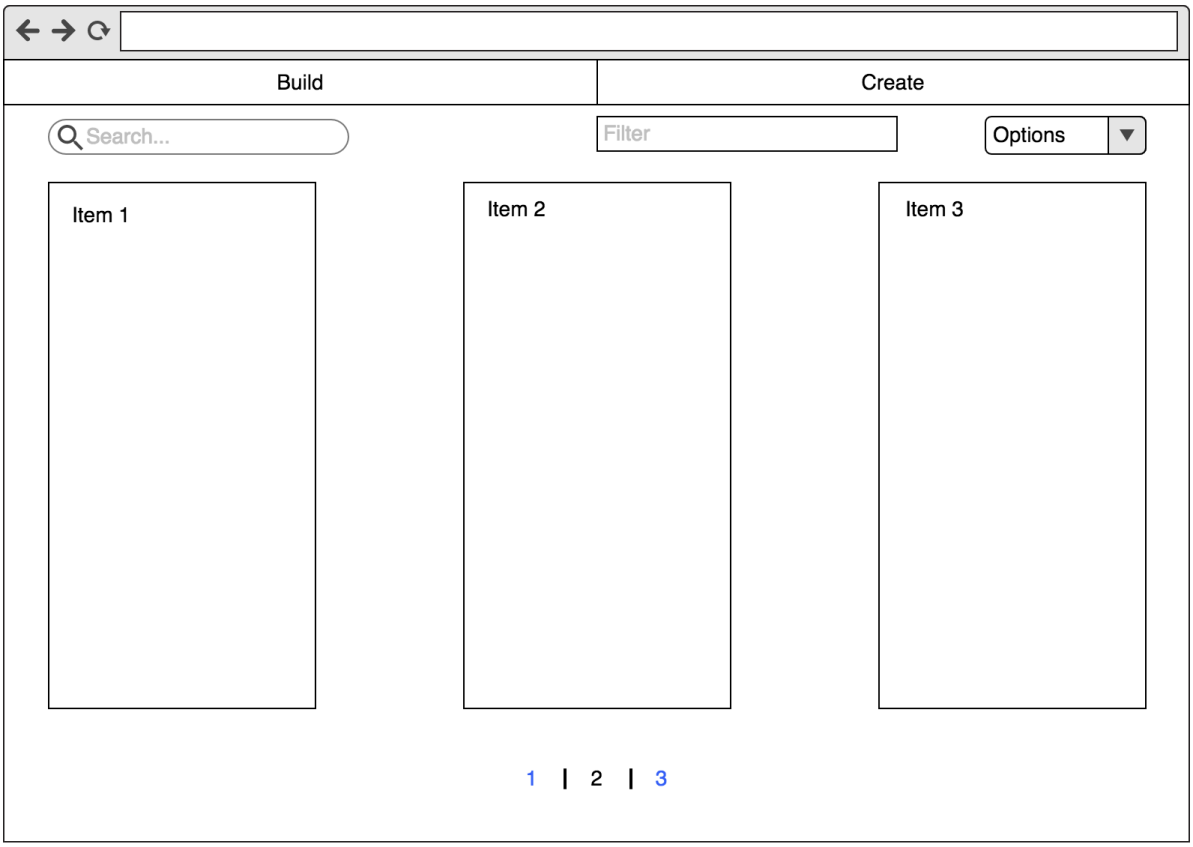
Obrázek 5: Definice záznamu

Zdroj: Vlastní zpracování

4.6 Uživatelské rozhraní

Dobře navržené uživatelské rozhraní zajišťuje jednoduchou, přirozenou a efektivní komunikaci mezi uživatelem a systémem. Pochopení uživatele je základním předpokladem pro vytvoření efektivního a použitelného rozhraní. Zkoumání potřeb a požadavků uživatelů pomáhá dosáhnout snížení složitosti UI. Předem je třeba říct, že konečný vzhled se může lišit od návrhu.

Obrázek 6: Návrh uživatelského rozhraní



Zdroj: Vlastní zpracování

4.6.1 ListView

ListView je hlavní obrazovka aplikace. Obsahuje seznám bioarcheologických záznamů, filtry, stránkování, tlačítka “Build” a “Create”. Vzhledem k velkému počtu sloupců se nedá použít klasická datová tabulka. Proto bylo rozhodnuto o použití tzv. “sloupcového layoutu”. Každý záznam je reprezentován horizontálním panelem, který obsahuje klíče a hodnoty. V rámci jednoho panelu je možné přepínat mezi statickou a dynamickou částí. Po kliknutí na tlačítka “Build” nebo “Create” se zobrazí modální okno pro definici formuláře nebo zakládaní záznamu. V dolní části stránky je umístěn stránkovací panel.

5 Implementace

V této kapitole je popsána implementace konkrétní aplikace s využitím dynamických formulářů v jazyce JavaScript. Je třeba říct, že obsahem této kapitoly nebude kompletní kód aplikace, ale budou popsány pouze nejdůležitější aspekty vývoje.

5.1 Databázový server

Instalace MongoDB serveru je v zásadě jednoduchá. Stačí stáhnout binární baliček z oficiálního webu MongoDB nebo nainstalovat pomocí správce balíčků pro cílový operační systém (v tomto případě MacOS). Server spustíme pomocí příkazu

```
sudo mongod
```

MongoDB obvykle běží na portu 27017. Shell spustíme příkazem

```
mongo
```

Tento shell představuje vyhodnocovač javascriptových příkazů v jádře. Jakmile je spuštěn, připojí se k výchozí databázi *test*, která bude vytvořena s první položkou. Chceme-li zobrazit všechny dostupné databáze, použijeme příkaz

```
show dbs
```

Aktuální databázi vidíme pomocí

```
db
```

Chceme-li se připojit k jiné databázi, stačí použít příkaz

```
use researches
```

Jak už bylo uvedeno, MongoDB obsahuje databázi, která obsahuje kolekci dokumentů. V předchozím kroky jsme se připojili k databázi *researches*. Nyní vložíme nový dokument do kolekce *findings*. Kolekce je objekt s více metodami, ze kterých používáme metodu insert() pro vkládání

```
db.findings.insert(objekt)
```

Obsah databáze můžeme vypsat voláním metody find() v kolekci

```
db.findings.find({})
```

5.2 Validační vrstva mezi DB a serverem

Je nutné mít na paměti, že do MongoDB lze vložit data jakéhokoliv typu. Žádná validace se neprovádí. Databáze je schopná zpracovat libovolný JSON dokument. Pro validaci a snadnější zpracování dat byla využita mezivrstva – Mongoose.

Mongoose je nástroj pro objektové modelování, který ve své podstatě funguje jako ORM v jiných jazycích. Mongoose nám umožňuje jednoduchý a snadný přístup ke CRUD příkazům.

Mongoose se přidává do projektu následujícím jednoduchým příkazem

```
npm install mongoose --save
```

Pak je balíček dostupný a používá se takto

```
var mongoose = require('mongoose')
```

Zbývá ještě jeden zásadní bod, a sice připojení k databází

```
mongoose.connect('mongodb://localhost/researches_test')
```

Před tím, něž můžeme používat CRUD operace, je potřeba definovat Mongoose model. Model představuje dokumenty, které lze uložit a načíst do databáze. Mongoose schema se používá pro definování atributů pro dokumenty. Nejprve se vytváří schéma, a pak na základě něj se definuje model. Striktní režim je nastaven na hodnotu “false”, protože v kolekci existuje dynamická část s předem neznámou strukturou.

5.3 Express aplikace a REST API

Jak už bylo zmíněno, pro vývoj backendu byl zvolen Express.js framework, který je postaven na platformě Node.js. Soubor index.js je počáteční bod, ve kterém probíhá spuštění serveru.

```
const express = require('express');

// set up Express app
const app = express();

const port = process.env.PORT || 8099;

// listen for request
app.listen(port, function(){
    console.log("Server is running...")
});
```

Během vývoje v Express.js je nutné chápat, co je middleware, a k čemu se používá. Express aplikace je v podstatě řada volání middleware funkcí. Middleware funkce mají přístup k request objektu, response objektu a k následující funkci v cyklu aplikace. Tyto funkce mohou provádět změny na response a request objektech, ukončit request-response cyklus nebo zavolat další funkci v pořadí. Jeden z typu middleware je Router-level. Pro tento typ si uvedeme příklad. Nejprve je třeba připojit middleware k express.Router() instanci.

```
const router = express.Router();
```

Takto následně bude vypadat definice REST API

```
router.get('/findings/:id', function(req, res){
    Finding.findOne({_id: req.params.id}).then(function(finding){
        res.send(finding);
    })
});

// create a new finding to the db
router.post('/findings', function(req, res, next){
    Finding.create(req.body).then(function(finding){
        res.send(finding)
    }).catch(next)

});

// update a finding in the db
router.put('/findings/:id', function(req, res){
    Finding.findByIdAndUpdate({_id: req.params.id},
req.body).then(function(){
        Finding.findOne({_id:
req.params.id}).then(function(finding){
                res.send(finding);
            }
        );
    })
});

// delete a finding from the db
router.delete('/findings/:id', function(req, res){
    Finding.findByIdAndRemove({_id:
req.params.id}).then(function(finding){
        res.send(finding);
    })
});
```

Tímto způsobem dojde k zachycení všech požadavků, které se budou posílat na API. Když například přijde GET request, zavolá se příslušná metoda a vrátí se požadovaný záznam. Pro komunikaci s databází se používá Mongoose model, který je popsán v kapitole 5.1.

5.4 React aplikace

Pro lepší zpracování dynamických formulářů byl zvolen nástroj “formBuilder”. Vzhledem k tomu, že není k dispozici funkční a použitelný balíček pro React.js, je nutné si využít jQuery. Takhle by vypadalo použití:

```
export default class FormBuilder extends React.Component {
    componentDidMount() {
        const {formbuilder} = this.refs;

        let options = {
            showActionButtons: false,
            dataType: 'json'
        };

        global.fb = $(formbuilder).formBuilder(options);
    }

    sendSchema() {
        let schemaObj = fb.actions.getData();
        this.props.sendFindingSchema(schemaObj);
    }
    render() {
        return (
            <div>
                <div ref="formbuilder"></div>
            </div>
        )
    }
}
```

Jak je vidět, komponent FormBuilder je pouze kontejner, ale uvnitř je jQuery plugin. Pak pro použití v šabloně stačí napsat

```
<FormBuilder />
```

Úplně stejným způsobem je definována komponenta “FormRender”, který má za úkol přijmout datovou definici formuláře a následně ho vykreslit do stránky. Už není potřeba vytvářet formuláře ručně, vše se generuje dynamicky. To je hlavní přínos těchto nástrojů.

Pro tvorbu vzhledu jsem použil balíček “react-bootsrap”. Je to souhrn UI komponent, kteří jsou napsané v React.js, ale se styly populárního grafického frameworku Bootstrap. Například modální okno se dá definovat takto

```
<Modal show={this.state.showModal} onHide={this.closeCreateModal}>
  <Modal.Header closeButton>
    <Modal.Title>Create a new finding</Modal.Title>
  </Modal.Header>
  <Modal.Body>

  </Modal.Body>
  <Modal.Footer>
    <Button onClick={this.closeCreateModal}>Close</Button>
  </Modal.Footer>
</Modal>
```

6 Testování

Testování aplikace probíhalo především manuálně, a to i během samotného vývoje. Hlavním úkolem bylo provést primárně testování uživatelského rozhraní a REST API. Cílem bylo ověřit, zda jsou splněné všechny požadavky na funkčnost a jestli rozhraní je dostatečně přijatelné pro uživatele. Grafické rozhraní bylo testováno v různých rozlišeních.

Kromě vizuální části bylo také testováno REST API za využití aplikace Postman, která usnadňuje a urychluje vývoj API. Postman dává možnost vyzkoušet všechny API požadavky ještě předtím, něž se začne programovat.

7 Možnosti dalšího vývoje

Jak už bylo zmíněno v úvodu, v této bakalářské práce se jednalo o „proof-of-concept“, a proto existuje prostor pro další rozvoj a vylepšení. V této kapitole jsou shrnuty možnosti dalšího vývoje a návrhy pro další rozvoj aplikace. V současném stavu je možné provádět pouze operace typu CRUD a definovat vlastní dynamické formuláře. Tato funkcionalita rozhodně není uspokojivá pro většinu uživatelů systému. Další vývoj aplikace se bude do budoucnosti zaměřovat na následující oblasti:

- Obecné vylepšení uživatelského rozhrání (Material Design)
- Možnost přepínání mezi tabulkou a sloupcovým layoutem v režimu náhledu.
- Zobrazení bioarcheologických nálezu na mapě.
- Management schémat.
- Automatické dokončování ve formulářích.
- Taxonomie. Pří zakládaní záznamu uživatel si může vybrat taxonomie v podobě stromové struktury.
- Zadávání GPS souřadnic.
- Pokročila filtrace: vyhledávání v dynamických datech, seřazení a tak dále.
- Stránkování a filtrace na backendu.
- Rozšíření serverové části. Import/Export dat.
- Možnost vytvoření uživatelských účtů a roli.
- Česká lokalizace.

8 Závěr

Výsledkem této bakalářské práce je software, který umožní pracovníkům Biologického centra AV ČR ukládat, prohledávat a manipulovat s daty, které jsou uložené v Bioarcheologické databázi ČR. Byla provedena rešerše existujících nástrojů pro dynamickou tvorbu webových formulářů, a na základě toho bylo vytvořeno funkční uživatelské rozhraní pro práci s daty. Také byla provedena analýza vybraných technologií, které následně byly použité během vývoje.

Funkcionalitou této aplikace je definice datového formuláře, zakládaní jednotlivých záznamů a jejich filtrace, prohledávání editace a mazání. Aplikace byla vyvíjená v souladu s požadavky zákazníka a je připravená pro další rozvoj a použíti. Je však potřeba poznamenat, že zhotovené řešení není finální a na vývoji se dále pokračuje.

Vzhledem k tomu, že aplikace je napsána s využitím nejnovějších a moderních technologií, je snadno rozšiřitelná do budoucna.

9 Literatura

Express.js. *Using middleware.* Dostupné z: http://expressjs.com/en/guide/using-middleware.html

Mongoose. Elegant mongodb object modeling for node.js. Dostupné z: http://mongoosejs.com/

Mrozek, J. (2012). *JavaScript na serveru: MongoDB, Mongoose a AngularJS.* Dostupné z: https://www.zdrojak.cz/clanky/javascript-na-serveru-mongodb-mongoose-angularjs/

Node.js Foundation. Dostupné z: https://nodejs.org/en/

npm Inc. Dostupné z: https://www.npmjs.com/

React-Bootstrap. The most popular front-end framework, rebuilt for React. Dostupné z: https://react-bootstrap.github.io/

React. A JavaScript library for building user interfaces. Dostupné z: https://facebook.github.io/react/

Steigerwald, D. (2014). *Proč Facebook React zabil jQuery*. Dostupné z: https://www.zdrojak.cz/clanky/proc-facebook-react-zabil-jquery/

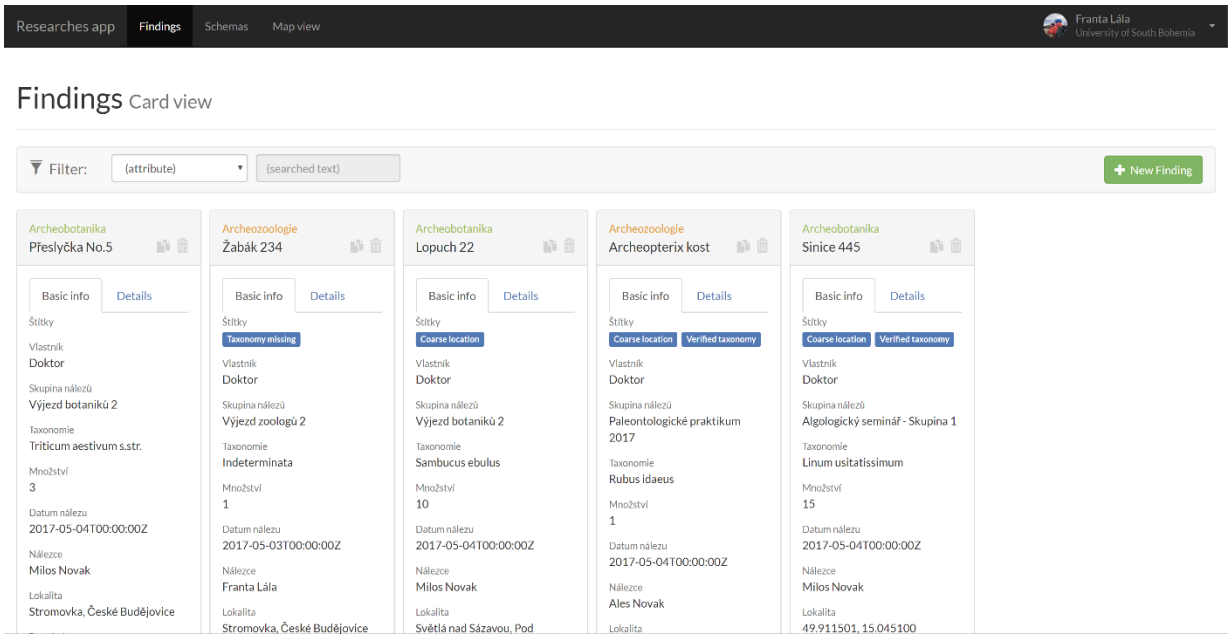
10 Seznam obrázků a tabulek

Obrázky:

Tabulky:

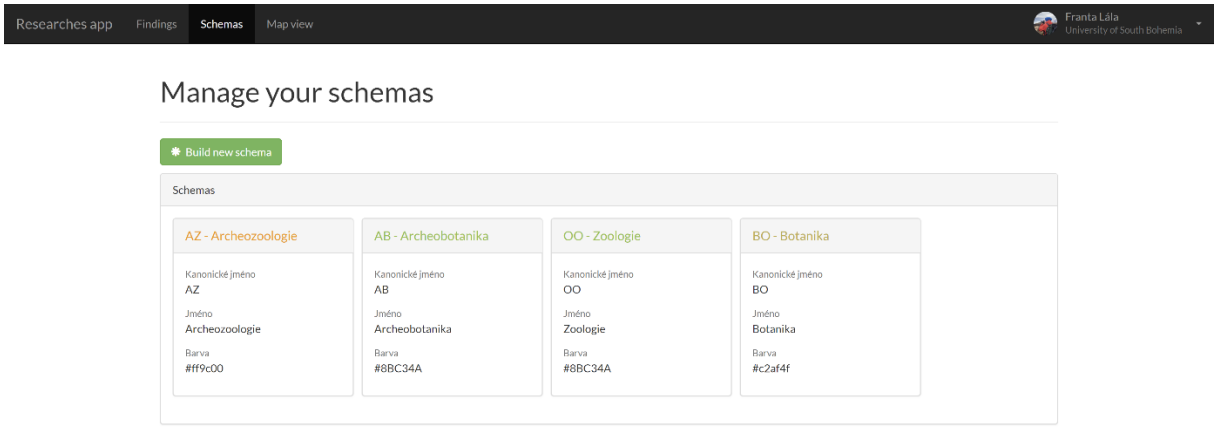
Příloha A Grafický vzhled aplikace

Obrázek 7: ListView



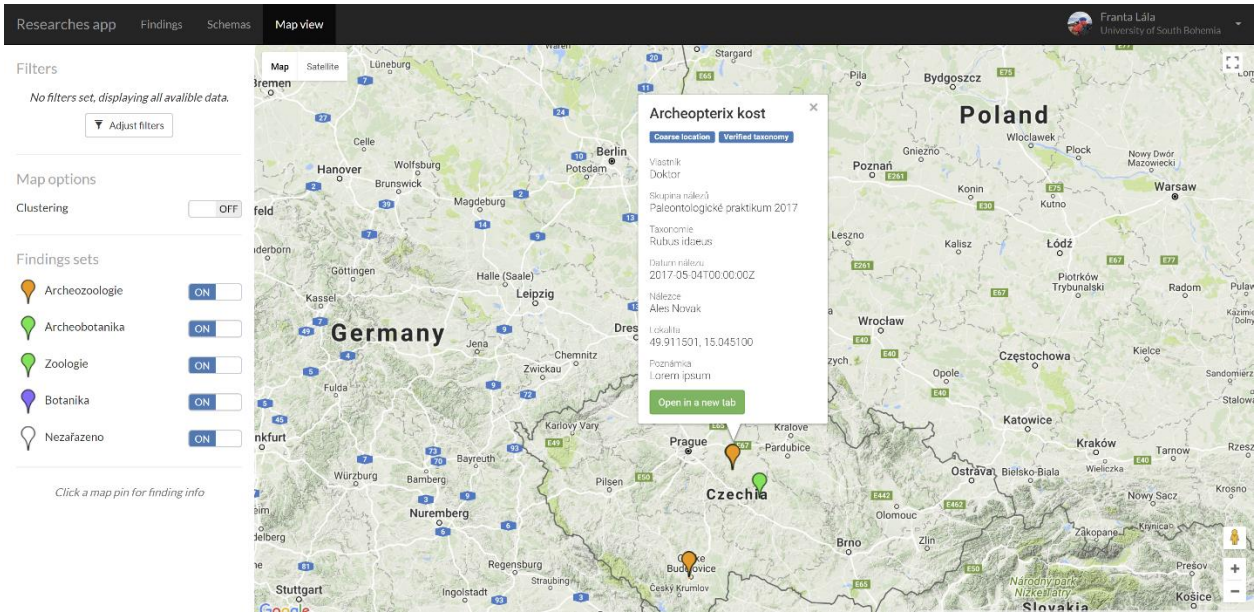
Zdroj: Vlastní zpracování

Obrázek 8: Management schémat



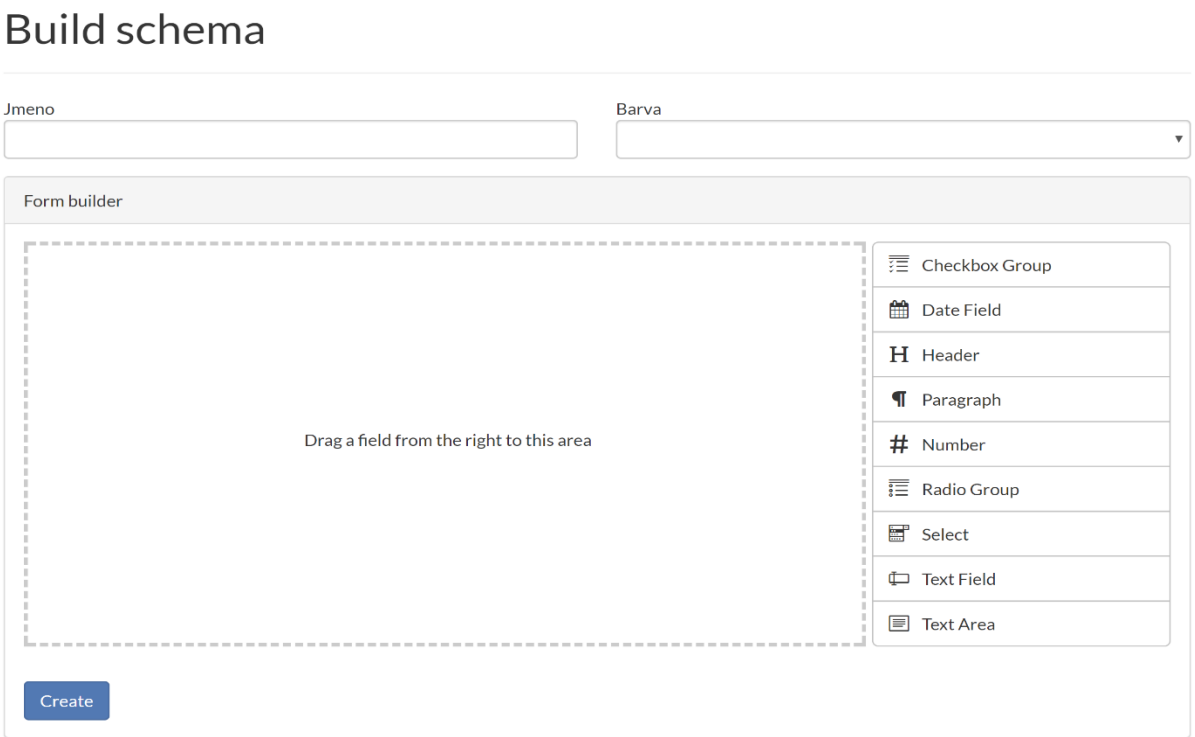
Zdroj: Vlastní zpracování

Obrázek 9: MapView



Zdroj: Vlastní zpracování

Obrázek 10: Dynamická definice formulářů



Zdroj: Vlastní zpracování