STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Systém pro vkládání a zpracování ankety/zpětné vazby

Jan Hubený

Pardubický kraj

Pardubice, 2022

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Systém pro vkládání a zpracování ankety/zpětné vazby

Web application for processing surveys/feedback

Autor: Jan Hubený
Škola: DELTA - Střední škola informatiky a ekonomie, s.r.o.
Ke Kamenci 151, 530 03 Pardubice
Kraj: Pardubický kraj
Konzultant: RNDr. Jan Koupil, Ph.D.

Pardubice, 2022

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Pardubicích dne 25. Března 2022 .....

Jan Hubený

Poděkování

Prvně bych chtěl poděkovat mému vedoucímu práce RNDr. Janu Koupilovi, Ph.D. za odborné vedení projektu a věcné připomínky. Dále děkuji Bc. Tomáši Vanišovi za pomoc při prvotních krocích při tvorbě aplikace a za pomoc při výběru technologií pro tento projekt. V neposlední řadě také děkuji své rodině, přátelům a spolužákům za psychickou podporu, za účast na testování aplikace a za nápady pro zlepšení aplikace.

Anotace

Práce dokumentuje vývoj a používání webové aplikace pro zpracování ankety nebo zpětné vazby. Systém je založen na backendovém frameworku Laravel, frontendovém frameworku VueJS a na databázovém systému PostgreSQL. Uživatele v aplikaci mohou jednak formuláře vyplňovat, ale i spravovat. Kromě základní funkcionality jako je vykreslení formuláře a následná možnost odpovědi aplikace obsahuje např. omezení přístupu datem spuštění a ukončení, privátní přístup pozvaným uživatelům či export dat pro účely korelační analýzy.

Klíčová slova

Webová aplikace; Zpracování ankety; Laravel, VueJS, PostgreSQL

Annotation

The thesis documents the development and use of a web application for processing surveys or feedback. The system is based on the backend framework Laravel, the frontend framework VueJS and the database system PostgreSQL. Users can both fill out and manage forms in the application. In addition to basic functionality such as form rendering and subsequent response option, the application includes for example access restriction by start and end date, private access for invited users or data export for correlation analysis.

Keywords

Web application; Processing surveys; Laravel, VueJS, PostgreSQL

Obsah

1 Úvod

Anketa a sběr dat zpětné vždy vazby byly, jsou a budou potřeba pro různé druhy lidských aktivit jak v reálném světe, tak i na internetu. V dnešní době se lidé mohou zajímat např. o výsledky ankety zhodnocující průběh pobytu v zahraničí či zpětnou vazbu k nějaké veřejné akci. Tato data nám mohou pomoci např. při zlepšování služeb nebo pro vyhodnocení úspěšnosti a popularity dané akce či produktu. A právě pro tyto účely jsou tvořeny webové anketní systémy, které nám zajišťují plnou kontrolu nad zmíněnými daty, což má být i výstupem této práce.

Cílem práce je vytvořit intuitivní, jednoduchý, a přitom komplexní webový systém pro správu a zpracování ankety – tzn. možnost vytvoření formuláře, dále možnou úpravu existujícího formuláře a následné publikování formuláře pro zisk dat zpětné vazby.

2 Existující dostupná řešení

Webových systému pro zpracování ankety je na trhu opravdu nepřeberné množství. Představme si proto alespoň některé z nich, jelikož byly inspirací pro mou vlastní implementaci.

2.1 Google Forms

Google Forms (česky Google Formuláře) je webová aplikace určená k tvorbě online průzkumů a kvízů. Je součástí balíčku webových aplikací od Googlu, ve kterých nalezneme např. Google Sheets či Google Docs. Při tvorbě těchto formulářů máme možnost pracovat s různými předpřipravenými šablonami nebo s předdefinovanými typy otázek. Do formulářů můžeme zakomponovat i různá multimédia jako např. videa či obrázky a samozřejmě zde nechybí i vizualizace výsledků. Jediné, co potřebujeme k tvorbě formulářů na této platformě, je vytvoření a přihlášení do Google účtu. [1][2]

2.2 Microsoft Forms

Microsoft Forms (česky Microsoft Formuláře) je služba umožňující tvorbu a sdílení dotazníků a kvízů určená jak pro soukromé účely, tak pro firemní sféru. Je součástí balíčku aplikací Microsoft Office, kam patří např. Microsoft Word nebo Microsoft Excel. I zde máme při tvorbě ankety možnost vybrat z předdefinovaných typů otázek společně s např. možností nahrání souboru či zobrazení výsledků. K tvorbě ankety se stačí jen přihlásit do svého Microsoft účtu. [3][4]

2.3 Survio

Survio je anketní nástroj pro měření zákaznické spokojenosti, marketingový průzkum či jiné účely původem z Česka. Kromě samotného tvoření formuláře nabízí i funkce jak např. export dotazníku do PDF, souhrnnou statistiku ve grafech a tabulkách či zpracování výsledků v reálném čase. Tvoření ankety se základními funkcemi je zdarma, služba ale nabízí i možnost prémiových funkcí. K tvorbě formuláře je znovu třeba se přihlásit. [5]

3 Použité technologie

Tato kapitola shrnuje a představuje hlavní technologie využité při řešení a implementaci projektu.

3.1 Frontend

Za frontend je v souvislosti s programováním a návrhem aplikace považováno to, co běžně vidí uživatel dané aplikace. Jedná se o vizuální stránku programu, díky které lze komunikovat s tzv. backendem (viz dále) – to znamená, že musí jednak správně interagovat s uživatelem (např. umožnit uživateli si v e-shopu vybrat zboží, zpracovat další potřebné údaje poskytnuté uživatelem a následně zboží objednat), ale i se serverem (např. posílat validní data a instrukce od uživatele na server).

Lze ji také označit za „klientskou stranu“ aplikace, jelikož je tvořena primárně pro běžnou a intuitivní obsluhu lidmi, kteří nemusí a ani nepotřebují rozumět technologiím, na kterých tato část funguje. [6]

3.1.1 JavaScript

JavaScript je multiplatformní, objektově orientovaný, událostmi řízený skriptovací jazyk, který je používán primárně při tvorbě webových stránek. Je určen především pro tvorbu klientské strany, ale např. díky prostředí Node.js lze v něm dnes psát i serverovou část aplikace. [7][8]

Psát web v čistém JavaScriptu dnes v souvislosti s pokročilejšími webovými aplikacemi s největší pravděpodobností postrádá smysl, jelikož zde existují tzv. javascriptové frameworky, které za nás řeší nejrůznější problémy (jak technické, tak i bezpečnostní), které bychom museli implementovat v čistém JavaScriptu ručně. Tento projekt není výjimkou, a proto byl při tvoření klientské strany použit framework VueJS, který je představen v sekci 3.1.2.

3.1.2 VueJS

VueJS je progresivní open-source framework určený k tvorbě uživatelských rozhraní. Jedná se o framework psaný v jazyce JavaScript, ale lze ho psát i v TypeScriptu. Tento framework je vhodný primárně jen pro účely viditelné části aplikace – nelze v něm např. programovat chování serveru. [9]

Svou popularitu si získal především tím, že je vyvíjen komunitou, a strukturou kódu, ve kterém je psán (viz obrázek 3.1). Kód jednotlivých komponent je tvořen třemi základními částmi:

- **Template** - část určená pro návrh struktury komponenty v HTML
- **Script** - část určená pro tvoření logiky a manipulaci s daty v komponentě v JavaScriptu
- **Style** - část určená pro stylování komponenty v CSS

Nutno zmínit, že se jedná o framework, který je vytvořen právě pro práci s komponentami – znamená to, že je vhodné jednotlivé části aplikace rozdělovat na jednotlivé komponenty, které lze použít později v jiném místě aplikace. Tento přístup vede k udržitelnosti aplikace. [10][11]

```
<template>
    <div>Content</div>
</template>

<script>
    export default {
        name: "Component",
        mounted() {
            console.log("Rendered successfully.")
        }
    }
</script>

<style scoped>
    div {display: block;}
</style>
```

Obrázek 3.1: Ukázka struktury kódu VueJS komponenty

3.1.3 Bootstrap

Bootstrap je open-source framework určený především pro stylování běžných komponent webových aplikací. Styly se aplikují na HTML ve formě tříd, kterým jsou interně předepsány parametry, jakým způsobem má být daný element nastylován. Také obsahuje předem hotové styly pro běžné komponenty jako jsou např. formuláře a jejich prvky, navigační lišty, modální okna atp. [12][13][14]

3.1.4 NPM

NPM je rozsáhlý open-source správce balíčků pro JavaScript. Je využíván zejména k přidávání komunitou vytvořených balíčků do vlastních projektů či k sdílení vlastních balíčků - ty jsou uloženy v tzv. registru, což je veřejná databáze javascriptového softwaru. K instalaci balíčků se nejčastěji používá CLI (Command Line Interface), kterým správce NPM disponuje. [15]

V projektu byl využit zejména k přidání a správě balíčků na frontendové části společně s VueJS.

3.2 Backend

Termínem backend označujeme část aplikace, která pro běžného uživatele zpravidla není vidět. Jedná se o administrátorskou část, ve které administrátor konfiguruje, jak se aplikace bude při jednotlivých situacích chovat. Backend, jako samostatná část aplikace, poté slouží ke zpracování dat – např. vyřizuje příchozí požadavky od frontendu nebo na frontend sama zasílá konkrétní data.

Backend může být také zamýšlen jako prostředí pro správu např. obsahu webu. Příkladem může být e-shop, kde na backendu bude moci administrátor např. přidávat jednotlivé zboží nebo spravovat veškeré objednávky. [16][17]

3.2.1 PHP

PHP (zkr. pro Hypertext Preprocessor) je open-source multiplatformní skriptovací jazyk. Je používán především pro vývoj webových aplikací a může být vložen přímo v souborech HTML. V rámci základních informací lze zmínit, že podporuje procedurální i objektově-orientované programování a dokáže pracovat s velkým množstvím databázových systémů. Oproti JavaScriptu je kód spouštěn přímo na serveru, kde generuje HTML, které je předáváno dále na klientskou stranu uživateli. [18][19]

Hlavním důvodem, proč byl pro tento projekt jazyk PHP vybrán, byla jednoduchost syntaxe jazyka a obecně nižší provozní náklady. V jazyce PHP je napsán framework Laravel, ve kterém je tvořena backendová část projektu.

3.2.2 Laravel

Laravel je open-source framework pro tvorbu webových aplikací v jazyce PHP. Řeší za vývojáře mnoho běžně implementovaných funkcí jako např. autentifikaci, přesměrovávání, správu sessions a cookies souborů, tvoření databázových migrací nebo testování aplikace. Je inspirován frameworky jako Ruby on Rails, ASP.NET MVC nebo Sinatra a snaží se o to, aby vývoj webových aplikací byl pro vývojáře co nejpohodlnější a nejintuitivnější. [20] Framework vychází z návrhového vzoru MVC [21].

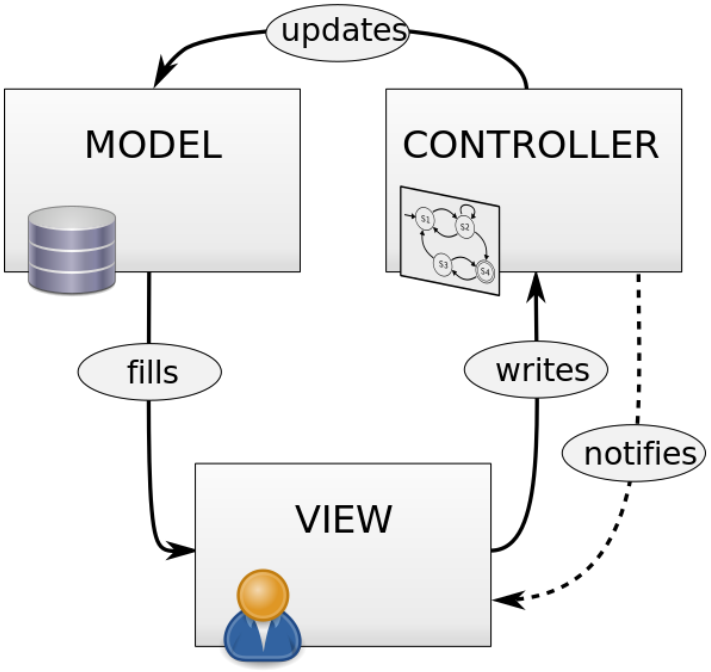
Dále samotný Laravel disponuje velkým ekosystémem přídavků, funkcí a knihoven, díky kterým si můžeme při vývoji ještě více usnadnit práci. [22] V mém projektu ze zmíněného ekosystému využívám např. Laravel Sanctum pro autentifikaci single-page aplikací.

3.2.2.1 Návrhový vzor MVC

MVC (zkr. pro Model-View-Controller) je architektonický vzor, jehož základní myšlenkou je oddělení logiky od výstupu. Obsahuje (jak už z názvu vyplývá) tři základní komponenty:

- **Model** – obsahuje veškerou logiku spojenou s daty (např. databázové dotazy); neřeší, odkud data přišla a ani jak budou dále předána uživateli
- **View** (v překladu Pohled) – řeší, jak budou data vyobrazena uživateli; neřeší, odkud mu data přišla
- **Controller** (v překladu Řadič nebo Kontrolér) – tzv. prostředník mezi modely a pohledy – propojuje je; reaguje na podněty uživatele

Jeho účelem je zamezení tvorby tzv. „špagetového kódu“ – tj. dlouhý soubor, který obsahuje implementaci výše zmíněných funkcionalit (vše na jednom místě), což je v konečném důsledku velmi nepřehledné. Také nám usnadňuje myšlení při vývoji projektu – jednotlivé části kódu se umisťují tam, kam patří (např. stylování a konečný výstup pro uživatele do pohledu). [23] Pro lepší představu je zde přiložena grafická podoba tohoto konceptu (resp. diagram), který je znázorněn v obrázku 3.2.



Obrázek 3.2: MVC diagram, převzato z [24]

3.2.3 Composer

Composer je nástroj pro správu závislostí pro jazyk PHP. Umožňuje nám v projektu deklarovat, které cizí knihovny (resp. balíčky) budou v projektu použity. Ty poté dokáže nainstalovat nebo aktualizovat. Funkcionalitou je velmi podobný jako NPM (viz sekce 3.1.4). [25]

V tomto projektu byl použit primárně ke správě balíčků pro backendovou část.

3.3 Úložiště dat

Samotná data, která jsou spravována a vyměňována mezi backendem a frontendem, musí být samozřejmě někde ukládána. Kromě zmíněných dat samozřejmě musí být i samotná aplikace někde uložena a veřejně dostupná. K těmto účelům slouží právě úložiště dat – o využitých úložištích v tomto projektu je psáno níže.

3.3.1 Webhosting

Webhosting je služba, díky které si můžeme pronajmout prostor na cizím vzdáleném serveru pro naše webové stránky. Poskytovatelé těchto služeb většinou nabízí i služby spojené s provozem databází nebo e-mailové schránky. Hlavní výhodou této služby je fakt, že nemusíme fyzicky vlastnit žádný server a že nemusíme řešit administrativní úkony s ním spojené (např. instalace specializovaného softwaru na serveru). [26]

Pro účely tohoto projektu bylo potřeba mít jak webový server pro nahrání celé aplikace, tak i databázový server pro uchovávání dat. O využitých poskytovatelích těchto služeb je psáno v práci dále.

3.3.1.1 DigitalOcean

DigitalOcean je americký poskytovatel cloudových služeb s datovými centry po celém světě. Nabízí obrovské množství technologií a služeb – např. správu webového serveru, virtuální stroje či možnost zprostředkování databázového serveru. Celkově se společnost snaží především o poskytování služeb, které jsou uživatelsky přívětivé. [27][28]

Z balíčku služeb, které DigitalOcean nabízí, byla pro tento projekt použita takzvaná „App Platform“ [29], do které lze jednoduše nahrát soubory webové aplikace a následně využívat webové služby.

3.3.1.2 Heroku

Heroku je cloudová platforma, která je využívána k nasazení, správě a škálování moderních aplikací. V rámci nabízených služeb se velmi podobá DigitalOceanu – nabízí tedy např. možnost využití virtuálních strojů a služby s tím spojené nebo možnost zprostředkování nejrůznějších databázových serverů. [30][31]

V rámci této implementace byla od Heroku využita služba zprostředkovávající databázové služby – konkrétně PostgreSQL server.

3.3.2 Databáze

Databáze je organizovaná kolekce strukturovaných dat, která je v dnešní době běžně uchovávána elektronicky. Je většinou řízena tzv. database management systémem (DBMS), který zajišťuje veškerou administrativu nad daty. Data a DBMS dohromady tvoří databázový systém. Cílem moderní databáze je např. zajistit zpracování a uchovávání velkého množství dat, zabezpečení dat, dostupnost dat, možnost správy a údržby systému či škálovatelnost. Za předchůdce databáze lze považovat např. kartotéku u lékaře. [32]

Databáze můžeme dělit podle modelu, který svou strukturou a funkcionalitou splňují. Základní modely jsou uvedeny níže: [33]

- **Hierarchický model** – data jsou uspořádána do stromové struktury, kde jsou omezena vztahem „jeden k více“ (záznam z vyšší úrovně lze spojit jen se záznamy nižší úrovně), jsou identifikovány tzv. ukazatelem (ten ukazuje na místo v databázi, kde jsou data uložena). [34]
- **Síťový model** – data jsou uspořádána jako uzly rovinného grafu, jsou identifikovány tzv. ukazatelem; oproti hierarchickému modelu lze spojovat záznamy libovolně. [35]
- **Relační model** – data jsou uspořádána v tabulkách do relací (tj. spojení dat, která obsahují nějaký společný atribut), jsou identifikovány primárním klíčem. [36]
- **Objektově-orientovaný model** – data jsou reprezentována jako objekty, které vychází z objektově-orientovaného programování. [37]

K zajištění možnosti ovládání databáze musí být také zajištěn způsob, jakým lze s databází komunikovat. K těmto účelům vznikly tzv. dotazovací jazyky, které právě slouží ke komunikaci uživatele s příslušným programem [38] - zde databázovým systémem. Mezi nejpoužívanější jazyky tohoto druhu patří jazyk SQL, který je primárně používán s relačními databázemi [39].

Dalším důležitým faktorem je spolehlivost databáze. K tomuto účelu slouží tzv. zásady ACID - ty garantují, že data po jednotlivých úkonech nebo transakcích (zjednodušeně skupiny úkonů [40]) - např. aktualizace dat - budou přesná a konzistentní i při selhání systému. Zkratku ACID lze rozdělit na tyto části:

- **Atomicity** (v překladu Atomicita) – buď se transakce provede celá, nebo se data nijak nezmění [41]
- **Consistency** (v překladu Konzistence) – data musí být validní a konzistentní - tzn. že musí splňovat požadavky databázového systému např. platný typ vstupní hodnoty pro určitý atribut [41]
- **Isolation** (v překladu Izolace) – všechny transakce musí být vzájemně nezávislé a nesmí se nějak ovlivňovat [41]
- **Durability** (v překladu Odolnost) – po provedení transakce musí být data permanentně uložena v databázi [41]

V projektu byla využita relační databáze PostgreSQL, která je popsána v sekci 3.3.2.1.

3.3.2.1 PostgreSQL

PostgreSQL je open-source objektově-relační databáze (spojení relačního a objektově orientovaného modelu) splňující ACID zásady. Je rozšířená a oblíbená kvůli její architektuře, výkonnosti, spolehlivosti, datové integritě, rozšířitelnosti, škálovatelnosti a velkému množství funkcí. Využívá jazyk SQL, který rozšiřuje o další funkce a syntaxi. [42]

4 Vlastní implementace

4.1 Uchovávání dat

Jak již bylo zmíněno, uživatelská data a samotná aplikace musí být někde uchovávány. Webová aplikace (tj. VueJS frontend a Laravel backend) je nasazena pomocí služby „App Platform“ od poskytovatele DigitalOcean (viz sekce 3.3.1.1). PostgreSQL databáze pro administraci nad uživatelskými daty a formuláři, jejíž struktura bude rozebrána v této části, je nasazena na Heroku serveru (viz sekce 3.3.1.2).

4.1.1 Struktura databáze

K tomu, aby byla struktura databáze vysvětlena, je v příloze přiloženo vygenerované schéma pomocí programu DataGrip (viz Příloha 1: Schéma databáze), které by mělo sloužit pro lepší vizualizaci.

Ze schématu vyplývá, že je databáze tvořena tabulkami, které jsou provázány určitými relacemi. Všechny tabulky (kromě tabulky *password_resets*) mají unikátní primární klíč (atribut *id*), kterým lze identifikovat jednotlivé instance. Většina z nich má také tzv. cizí klíč, který slouží k propojení s jinou tabulkou na základě shody tohoto klíče s cizím primárním klíčem. Také většina obsahuje předgenerované atributy timestamps (v překladu časová razítka) *created_at* a *updated_at*, které nám umožňují sledovat, kdy byla položka vytvořena či upravena.

V práci nebudou popsány tabulky *personal_access_tokens*, *failed_jobs*, *migrations* a *password_resets*, jelikož se jedná o předgenerované tabulky Laravelem. Výjimkou je tabulka *users*, která byla cíleně využita.

4.1.1.1 Uživatelé

Základním objektem databáze je samotný uživatel (tabulka *users*), který obsahuje standardní vlastnosti jako jsou: uživatelské jméno (*name*), emailovou adresu (*email*), která také slouží jako identifikátor (musí být unikátní), heslo v zašifrované podobě (*password*) a další předgenerované atributy, které jsou využívány primárně samotným frameworkem Laravel. V rámci relací pro uživatele platí, že může mít více formulářů.

4.1.1.2 Formuláře

Dalším důležitým základním objektem je formulář (tabulka *forms*). Tento objekt reprezentuje samotný dotazník a jeho vlastnosti. Základním atributem je unikátní řetězec (*slug*), který má podobnou funkci jako primární klíč - ukazuje na konkrétní formulář. Dalšími atributy jsou: název formuláře (*name*), nepovinný popis formuláře (*description*), čas zveřejnění a čas ukončení zveřejňování (*start_time* a *end_time*), proměnná vyjadřující dostupnost formuláře - zda je veřejný nebo privátní s nutností vyplnění přes pozvánku (*has_public_results*) - a proměnná ukazující, zda má formulář veřejně dostupné výsledky (*has_public_results*). Tento objekt je vázán na další databázové objekty těmito vztahy: formulář musí mít jednoho uživatele (vlastníka) a musí mu náležet přístupové tokeny v případě privátního formuláře (tabulka *form_private_access_tokens*), samotné elementy formuláře (tabulka *form_elements*) a záznamy vyplnění formuláře (tabulka *form_completions*).

4.1.1.3 Elementy formuláře

Podstatným prvkem po objektu samotného formuláře jsou tzv. elementy formuláře (tabulka *form_elements*) - ty zaobalují další meziprvky jako jsou element pro uživatelský vstup (tabulka *input_elements*) a element pro označení nové stránky (tabulka *new_pages*) - těm přiřazují pořadí (atribut *order*), jak se mají ve formuláři zobrazovat. Tento prvek vznikl právě pro sjednocení hodnoty pořadí. Platí pro něj vztah, že mu musí náležet buď element pro uživatelský vstup nebo element označující novou stránku. Zároveň, jak již bylo zmíněno, sám musí náležet samotnému formuláři.

Jedním ze zmíněných meziprvků elementu formuláře je element označující novou stránku (tabulka *new_pages*). Jeho využití je prosté - odděluje od sebe elementy pro uživatelský vstup, což při vykreslování formuláře slouží ke stránkování. Má jedinou vazbu - musí náležet elementu formuláře (tabulka *form_elements*).

Dále přichází na řadu objekt elementu pro uživatelský vstup (tabulka *input_elements*), jehož účelem je hlavně zaobalení jednotlivých typů uživatelských vstupů. Také ale obsahuje atributy jako samotnou otázku (*header*), proměnnou popisující, zda musí být otázka povinně vyplněna (*is_mandatory*), a proměnnou ukazující, zda jsou veřejně publikovány výsledky zodpovězení příslušné otázky (*has_public_results*). Kromě vazby, že jako element nové stránky náleží elementu formuláře, mu musí náležet příslušný objekt typu vstupu (např. objekt z tabulky *text_inputs*).

4.1.1.4 Typy vstupů

Dalšími prvky, které zde budou rozebrány, jsou právě objekty typu vstupu. Těchto typů existuje v databázovém modelu několik:

- **Textový vstup** (tabulka *text_inputs*) - typ zaznamenávající textovou odpověď, kterému lze přiřadit validační atributy jako minimální, maximální či striktní délku (*min_length*, *max_length*, *strict_length*)
- **Číselný vstup** (tabulka *number_inputs*) - typ zaznamenávající odpověď ve formě čísla, kterému lze nastavit validační atributy jako minimální a maximální hodnota (*min* a *max*) či zda může být zadaná hodnota desetinné číslo (*can_be_decimal*)
- **Vstup pro datum** (tabulka *date_inputs*) - typ pro zaznamenávání odpovědí ve formě data, kterému lze přiřadit validační parametry jako minimální a maximální datum (*min* a *max*)
- **Vstup pro odpověď typu Ano/Ne** (tabulka *boolean_inputs*)
- **Vstup pro odpověď z výběru předepsaných možných odpovědí** (tabulka *select_inputs*) - typ zaznamenávající, jakou odpověď (pocházející z tabulky *select_input_choices*) dotyčný respondent pro příslušnou otázku vybral; lze u něj validovat zda se jedná o tzv. multiselect (možnost výběru více odpovědí - v tomto případě lze omezit i minimální, maximální nebo striktní počet odpovědí pomocí atributů *min*\|*max*\|*strict_amount_of_answers*) či singleselect (možnost zvolení pouze jedné odpovědi) podle atributu *is_multiselect* a zda má skrytý popisek používaný pro export dat podle atributu *has_hidden_label*

V rámci vztahů pro všechny platí, že jsou podřízení elementu formuláře (tabulka *form_elements*). Dále platí, že každému jednotlivému typu vstupu náleží tabulka pro zaznamenávání odpovědi (např. tabulka *number_input_answers*).

Speciálním případem je poslední zmíněný vstup pro odpověď z výběru předepsaných možných odpovědí. K němu je vytvořena tabulka *select_input_choices*, sloužící k ukládání jednotlivých možností odpovědí k nadřazenému vstupu pro odpověď. Jsou zde obsaženy tyto atributy: text vyjadřující možnou odpověď (*text*), skrytý popisek pro export (*hidden_label*) a pořadí, jak mají být možnosti otázky seřazeny.

4.1.1.5 Odpovědi k otázkám

Dalšími objekty jsou odpovědi k otázkám. Ty slouží k samotnému zaznamenávání uživatelských vstupů při vyplnění formuláře - tedy odpovědí. Je jich stejné množství jako počet typů vstupů a s každým tímto vstupem jsou samozřejmě relačně provázány - musí náležet jak příslušnému typu vstupu, tak i objektu vyplnění formuláře (tabulka *form_completions*). Obsahuji kromě standardních atributů jen atribut hodnoty (*value*), ve kterém je uložena právě odpověď na otázku.

Jedinou výjimkou je znovu objekt pro zaznamenávání odpovědí pro otázky s výběrem předepsaných možných odpovědí. V něm není hodnota uložena ve sloupci *value*, ale jako cizí klíč odkazující na příslušnou možnost otázky.

4.1.1.6 Vyplnění formuláře

Posledním popsaným objektem, který je nutné evidovat, je záznam o vyplnění nebo-li souhrn všech odpovědí na formulář od jednoho respondenta (tabulka *form_completions*). Ten seskupuje odpovědi z jednotlivých otázek ve formuláři pro jednu odpověď na formulář. Je přímo vázán se samotným formulářem, kterému musí vždy patřit, a musí mu náležet jednotlivé odpovědi k otázkám.

4.2 Backend

V rámci backendové nebo-li serverové části projektu je prvně nutné zmínit, že funguje na principu API - tzn. že na ní přicházejí požadavky od klienta, na které náležitě odpovídá. Nevrací ani nijak nespravuje vizualizaci dat - jen předává data na frontendovou část, o které je psáno v sekci 4.3.

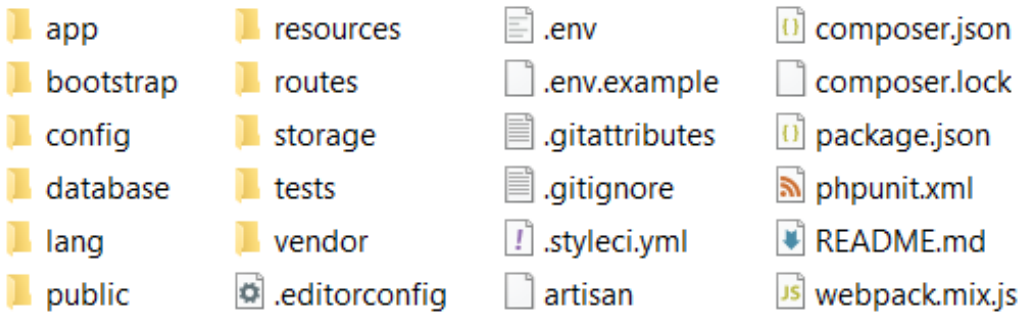
Zde jsou zobecněné základní úkony, které v této implementaci právě backend vykonává:

- Zpracování a vyřizování požadavků z webové frontendové části
- Administraci databáze - tzn. vytváření, úpravu, mazání a čtení jednotlivých objektů a migrace databáze (tj. automatizovaná deklarace struktury databázových objektu)
- Rozesílání emailů (např. pozvánky na neveřejný formulář)
- Exportování dat do Excel tabulek
- Autentifikaci uživatele

V následující části jsou popsány jednotlivé backendové komponenty a principy, díky kterým je celý projekt implementován.

4.2.1 Obecná struktura Laravel projektu

Prvně je nutné rozebrat obecnou strukturu Laravel projektu.



Obrázek 4.1: Obecná struktura nově vygenerovaného Laravel projektu

Jak můžeme na obrázku 4.1 vidět, celý projekt je složen z mnoha složek a souborů. Jelikož podrobný rozbor jednotlivých částí není předmětem této práce, tak jsou nejdůležitější části pro implementaci projektu popsány níže jen obecně.

- Složka *app* obsahuje většinu souborů jádra aplikace (obsahuje např. modely či kontrolery). [43]
- Složka *bootstrap* obsahuje soubory pro zavedení a spuštění aplikace. [43]
- Složka *config* obsahuje konfigurace jednotlivých částí aplikace. [43]
- Složka *database* obsahuje soubory spojené s prací s databází. [43]
- Složka *lang* obsahuje soubory jazyků a překladů (zde nevyužita). [43]
- Složka *public* obsahuje soubory, které jsou veřejně dostupné při uživatelské interakci s aplikací (např. při načtení webu v prohlížeči). [43]
- Složka *resources* obsahuje pohledy a nezkompilované soubory pro celý frontend. [43]
- Složka *routes* obsahuje všechny definice cest aplikace (např. cestu */api/forms/create* pro vyřízení požadavku na vytvoření nového formuláře). [43]
- Složka *storage* obsahuje primárně záznamy o chodu aplikace a jiné aplikací vygenerované soubory. [43]
- Složka *tests* obsahuje automatizované testy aplikace (zde nevyužita). [43]
- Složka *vendor* obsahuje závislosti a soubory přídavných balíčků spravované Composerem (viz sekce 3.2.3), které aplikace používá. [43]

- Soubor .env drží veškeré důležité administrativní hodnoty jako např. přihlašovací údaje k databázi.
- Soubor artisan obsahuje důležité příkazy pro sestavení a spuštění aplikace. [44]
- Soubor composer.json drží informace o přídavných balíčcích Composeru (viz sekce 3.2.3) pro Laravel projekt. [45]
- Soubor package.json drží informace o přídavných balíčcích spravovaných pomocí NPM (viz sekce 3.1.4) pro frontendovou část projektu. [46]
- Soubor webpack.mix.js obsahuje informace pro kompilaci souborů pro frontend. [47]

4.2.2 Cesty

Cesty (volně přeložen původní název Routes) jsou abstraktně využívány k rozdělení aplikace - to je myšleno tak, že na každé cestě lze provádět jen úkon, který jí je přiřazen. Tyto cesty mají tvar URL adresy - skládají se z názvu výchozí domény, který je rozšířen o příslušnou cestu (ve formě textového řetězce děleného lomítky), a jsou využívány např. v prohlížeči pro přístup na specifickou stránku. Příkladem může být cesta vytvoření formuláře s příkladovou výchozí doménou *localhost* s portem 8000 *„localhost:8000/forms/create“*. Po zadání této cesty do webového prohlížeče se za splnění podmínek jako spuštění lokálního serveru s aplikací a přihlášení do uživatelského účtu zobrazí požadovaná stránka.

Běžně se v Laravelu tyto cesty definují v souboru *web.php* - v tomto projektu tomu tak ale úplně není, jelikož cesty, kam se uživatel může v aplikaci dostat, jsou definovány ve frontendové části (viz sekce 4.3.4) - to také vyplývá z předem dané koncepce aplikace - Laravel má sloužit jen jako API, proto neřeší, jaká data budou v prohlížeči vykreslena - k tomu slouží frontendová část ve VueJS. K těmto účelům je ve zmíněném souboru deklarováno, že na jakékoli cestě se má vrátit prázdný pohled *app* (popsán v části 4.2.8), který je pak doplněn o příslušný obsah, který zajišťuje oddělený frontend (viz obrázek 4.2). Jedinou výjimkou jsou zde předdefinované cesty pro resetování hesla pro samotný Laravel.

```
Route::get('{any}', function () {
    return view('app');
})->where('any', '.*');
```

Obrázek 4.2: Část kódu *web.php* vracející pohled *app* na všech cestách

Cesty, které Laravel specificky obsluhuje, jsou v souboru *api.php* (viz. obrázek 4.3). Na tyto cesty uživatel v adresním řádku webového prohlížeče vůbec nepřistupuje - jsou využívány ve frontendové části na pozadí (viz sekce 4.3.2). Až zde je vlastně definováno abstraktní rozdělení aplikace - dle cesty se vykonává určitý úkon.

Tyto API cesty jsou oproti běžným cestám aplikace rozlišeny tak, že mezi výchozí doménu a cestu k dané službě je vložen řetězec „*/api/*“ - to je výchozí chování cest v souboru *api.php*. K tomu, aby tyto cesty fungovaly s dalšími službami, bylo nutné je specifikovat v konfiguračních souborech *sanctum.php* a *cors.php*.

```php
Route::post('/form/complete/{slug}', [FormCompletionController::class, 'store']);
Route::post('/form/private_complete/{token}', [FormCompletionController::class, 'privateStore']);

Route::get('/form/public_results/{slug}', [FormCompletionController::class, 'publicIndex']);

//require user privileges
Route::group(['middleware' => ['auth:sanctum']], function() {
    Route::get('/form/authenticated/{slug}', [FormController::class, 'showWithAuth']);
    Route::get('/forms', [FormController::class, 'index']);
    Route::put('/forms/update/{slug}', [FormController::class, 'update']);
    Route::post('/forms/create', [FormController::class, 'store']);
```

Obrázek 4.3: Část kódu *api.php*

Zmíněné cesty jsou rozděleny do dvou skupin - s veřejným přístupem (bez přihlášení) nebo s privátním přístupem (s přihlášením). Veřejné cesty jsou obslouženy vždy, privátní cesty jsou obslouženy jen pokud jsou společně s příslušnými daty odeslány údaje o přihlášení (tj. autorizační tokeny) - ty si autonomně spravuje balíček „Laravel Sanctum“ pro zabezpečení single-page aplikací. Ke každé cestě je přidělen úkon (tj. funkce obsahující, co se má s požadavkem provést), který je deklarován v příslušném kontroléru.

Zmíněné cesty mají také deklarováno, jakou metodu komunikace používají. V této implementaci byly využity zejména tyto metody: standardní GET a POST, ale také PUT pro aktualizaci dat nebo DELETE pro mazání dat.

V rámci popisu aplikace jsou zde tabulky s jednotlivými cestami a jejich úkony - tabulka 4.1 obsahuje veřejné cesty a tabulka 4.2 obsahuje cesty s přihlášením. Hodnoty u cest ve složených závorkách značí určitou proměnnou přenášenou přes URL adresu (např. *slug* jako identifikátor formuláře).

Cesta (/api/ + řetězec níže)	**Metoda**	**Funkce**
/login	POST	Přihlášení uživatele
/register	POST	Registrace uživatele
/logged_user	GET	Vrácení přihlášeného uživatele
/forgot_password	POST	Zaslání emailu pro resetování hesla
/reset_password	POST	Resetování hesla
/form/{slug}	GET	Vrácení veřejného formuláře
/private_form/{token}	GET	Vrácení privátního formuláře
/form/complete/{slug}	POST	Odeslání odpovědi na veřejný formulář
/form/private_-complete/{token}	POST	Odeslání odpovědi na privátní formulář
/form/public_results/{slug}	GET	Vrácení veřejných výsledků formuláře

Tabulka 4.1: Veřejné cesty

Cesta (/api/ + řetězec níže)	**Metoda**	**Funkce**
/form/authenticated/{slug}	GET	Vrácení formuláře pro administrátorský přístup
/forms	GET	Vrácení formulářů, které patří uživateli
/forms/update/{slug}	PUT	Aktualizace formuláře
/forms/create	POST	Vytvoření formuláře
/forms/{slug}	DELETE	Odstranění formuláře
/form/duplicate	POST	Duplikování formuláře
/form/results/{slug}	GET	Vrácení výsledků formuláře
/form/results/{slug}/download	GET	Export výsledků formuláře ke stažení
/form/results/{slug}/publish_results	POST	Nastavení veřejných výsledků
/forms/update_access/{slug}	PUT	Úprava přístupu k formuláři
/logout	POST	Odhlášení uživatele
/change_password	PUT	Změna hesla
/delete_account	POST	Smazání účtu uživatele

Tabulka 4.2: Cesty s přihlášením

4.2.3 Modely

Modely slouží k mapování jednotlivých dat z databáze a jsou zprostředkovány pomocí objektově-relačního mapovacího balíčku Eloquent. Pro správnou funkčnost musí být zajištěno, že má každá tabulka v databázi vlastní model. Díky tomuto přístupu lze s daty manipulovat tak, že vytvoříme instanci příslušné třídy a na ní voláme příslušné metody (např. *update* pro změnu záznamů) - nemusíme tedy vytvářet jednotlivé SQL příkazy pro všechny požadované úkony [48].

V rámci tohoto projektu vzniklo mnoho modelů, které mapují většinu tabulek databáze (v kontextu mnou vytvořených modelů nejsou vytvořeny modely např. pro předgenerované tabulky jako *migrations* nebo *failed_jobs*). Kromě samotné funkce mapování dat z databáze jim můžeme přiřazovat různé vlastnosti a metody, díky kterým lze měnit např. chování při předávání dat uživateli. Nejdůležitější a v projektu použité jsou v následujícím seznamu:

- Metody vytvářející vazby mezi modely - Ty určují jednotlivé vztahy mezi modely a slouží k jednodušší práci s daty. Existuje mnoho metod - např. *belongsTo* (označuje, komu samotný model náleží) či *hasOne*/*hasMany* (označují, které model/y samotnému modelu patří). [48]
- Vlastnost *fillable* - Ta určuje, do kterých atributů může být na vrstvě webové aplikace zapisováno. Nenapíšeme sem tedy např. atribut *id*, který většinou chceme doplnit až při uložení do databáze samotnou databází. [48]
- Vlastnost *visible* - Ta určuje, které atributy jsou ve vrácených datech z databáze viditelné a tedy poslané dále (myšleno na frontendovou část aplikace). Např. z bezpečnostního hlediska sem nenapíšeme atribut uživatelského hesla, který by neměl být obecně předáván mimo server. [48]
- Vlastnost *table* - Ta slouží k přesnému určení tabulky pomocí jejího jména, ke které se vytvořený model vztahuje. [48]
- Vlastnost *with* - Ta slouží k výchozímu připojení dalších dat z jiného modelu ke stávajícímu setu dat modelu. K takovému úkonu je nutné mít vytvořené vazby pomocí příslušných metod zmíněných výše. [48]

Každý model většinou nevyužívá všechny zmíněné činnosti, ale jen ty, které jsou pro jeho typ důležité. Příkladem je zde uveden poměrně rozsáhlý model formuláře *Form*. Podobným způsobem jsou vytvořeny i ostatní modely.

Třída modelu *Form* reprezentuje jednotlivé formuláře. Odkazuje se na stejnojmennou tabulku *forms*, ze které přebírá data. Má definované vlastnosti (viz obrázek 4.4): *fillable* (např. jméno formuláře, popis formuláře, čas spuštění, ID uživatele, kterému formulář

patří,...), *visible* (např. atributy jako jméno formuláře, odkaz na formulář či popis formuláře, ale také názvy atributů, které jsou vytvářeny a přidávány v průběhu práce s daty, a názvy metod vytvářejících vazby mezi některými dalšími modely (viz obrázek 4.5) jako např. pro model uživatele, pro který platí, že mu formulář musí náležet) a *with* (zde jen název vazbové metody pro model elementu formuláře, u kterého platí, že musí formuláři náležet a že jednomu formuláři může náležet i více elementů). Samotná definice provázání modelu s tabulkou v databázi pomocí vlastnosti *table* zde není potřeba - Laravel umí u jednoduše pojmenovaných modelů vygenerovaných pomocí Artisan skriptů tuto vlastnost vnitřně doplnit.

```php
protected $visible = [
    'slug',
    'name',
    'description',
    'start_time',
    'end_time',
    'has_private_token',
    'has_public_results',

    //visible props, not DB columns
    'formElements',
    'formCompletions',
    'user',
    'is_expired',
    'was_already_published',
    'is_opened',
    'results_table',
    'formPrivateAccessTokens'
];
```

```php
protected $fillable = [
    'slug',
    'name',
    'description',
    'end_time',
    'start_time',
    'has_private_token',
    'user_id',
    'has_public_results'
];

protected $with = ['formElements'];
```

Obrázek 4.4: Zmíněné vlastnosti modelu formuláře

```php
public function user()
{
    return $this->belongsTo(User::class);
}

public function formElements()
{
    return $this->hasMany(FormElement::class);
}

public function formCompletions()
{
    return $this->hasMany(FormCompletion::class);
}

public function formPrivateAccessTokens() {
    return $this->hasMany(FormPrivateAccessToken::class);
}
```

Obrázek 4.5: Zmíněné metody modelu formuláře

4.2.4 Kontroléry

Kontroléry jsou jednou z nejdůležitějších částí celé aplikace - probíhají v nich všechny požadované činnosti, které voláme přes specifické cesty. Obecně zajišťují veškerou administrativu nad příslušným modelem - tzn. např. model formuláře by měl mít svůj kontrolér. Většinou v nich najdeme metody pro vrácení všech prvků nebo konkrétního prvku příslušného modelu či metody pro ukládání, aktualizaci a mazání jednotlivých prvků.

Kontroléry v této aplikaci (viz obrázek 4.6), resp. jejich metody, jsou nastaveny tak, aby v případě chyby vrátily příslušný stavový kód pro jednoznačnou identifikaci chyby. Pokud se chyba nevyskytne, jsou navrácena příslušná data se stavovým kódem 200, který značí úspěch akce.

```
class FormController extends Controller
{
    /**...
    public function index()
    {
        $userId = Auth::user()->id;
        if ($userId) {
            return Form::where('user_id', $userId)
                ->without('formElements', 'user')
                ->orderBy('created_at', 'desc')
                ->get();
        } else return response("Unauthorized", 401);
    }

    /**...
    public function store(Request $request, $updateSlug = null, $updateHasPrivateToken = null)
    {...
    }

    /**...
    public function show($slug)
    {...
    }

    public function privateShow($token)
    {...
    }
```

Obrázek 4.6: Ukázka části kódu kontroléru formuláře

Konkrétní postupy, jak se jednotlivé činnosti provádí, jsou popsány v sekci 4.2.9.

4.2.4.1 Kontrolér formuláře

Kontrolér formuláře, jak už z názvu vyplývá, pracuje s daty jednotlivých formulářů. Kromě výše zmíněných běžných metod obsahuje metodu pro vrácení privátního formuláře, metodu pro duplikaci formuláře, metodu pro vrácení formuláře s dodatečnými daty pro majitele formuláře, metodu pro nastavení veřejných výsledků či metodu pro úpravu přístupu k formuláři (nastavení veřejného nebo privátního formuláře).

4.2.4.2 Kontrolér vyplnění formuláře

Kontrolér vyplnění formuláře je oproti předchozímu kontroléru chudší. V rámci běžných metod neobsahuje např. metodu pro úpravu či smazání odpovědi. Oproti tomu ale disponuje speciální metodou pro export všech odpovědí do stáhnutelného souboru ve formátu MS Excel, metodou pro uložení odpovědi z privátního formuláře či metodou pro zobrazení veřejných výsledků.

4.2.4.3 Kontrolér uživatele

Kontrolér uživatele slouží ke správě uživatelů. Obsahuje základní metody pro registraci, přihlášení a odhlášení, ale i pro změnu hesla (i při zapomenutém hesle) či smazání účtu.

4.2.5 Exporty

Exporty, resp. jejich třídy, slouží k definování vzhledu a způsobu uložení dat v určitém formátu. Exportovat můžeme různá data do různých formátů (např. obrázek do formátu PNG, list nějakých záznamů do tabulky formátu CSV,...).

V rámci tohoto projektu byl export potřeba jen v jednom případě - pro exportování výsledků vyplnění formuláře pro majitele formuláře do formátu MS Excel. K tomuto účelu byl využit přídavný balíček „maatwebsite/excel“, který zajišťuje veškerou administrativu nad věcmi, které jsou s tímto úkonem spojené. Jakým způsobem jsou tato data exportována je popsáno v sekci 4.2.9.2.5.

4.2.6 Maily

Maily, resp. jejich třídy, slouží k definování vzhledu a obsahu emailu. Poté, co se vytvoří instance této třídy, která je hydratována (tzn. naplněna daty), jsou data předána příslušnému pohledu, který je poté poslán na požadované emaily.

Způsoby a principy, kterými jsou emaily rozesílány, zde nejsou popsány - jednak jsou předdefinované Laravelem a zároveň nejsou předmětem této práce. Jediné, co bylo potřeba k zprovoznění emailového klienta, bylo vyplnit příslušné konfigurační údaje (např. protokol pro přenos nebo port na emailový server) a přihlašovací údaje k emailu, ze kterého jsou emaily rozesílány. K těmto účelům byla v implementaci projektu na produkční verzi použita bezplatná emailová služba „Google Gmail“.

4.2.7 Migrace databáze

Migrace databáze jsou třídy určené k deklaraci struktury jednotlivých tabulek v databázi. Obecně mají dvě metody: *up* pro vytvoření předdefinované tabulky a *down* pro odstranění tabulky z databáze. Tyto migrace se v Laravelu běžně generují pomocí Artisan skriptů. [49]

V implementaci tohoto projektu se využívala primárně metoda *up*, kde se specifikovaly jednotlivé názvy a typy atributů tabulky (viz obrázek 4.7). Těmto atributům lze přidávat další různé vlastnosti (v migraci zapsány jako metody) jako např. *nullable* (možnost vložení prázdné hodnoty *null* do atributu), *default* (možnost nastavení výchozí hodnoty atributu při vytvoření) či *unique* (možnost nastavení unikátní hodnoty atributu ve sloupci pro celou tabulku).

Kromě zmíněných vlastností bylo ještě nutné specifikovat tzv. cizí klíče, které slouží k vytvoření relací mezi ostatními databázovými objekty a k celkovému provázání dat. Jedná se o samostatné atributy, které většinou obsahují ID jiného objektu v databázi - tím je zaručeno, který řádek v tabulce patří k řádku/řádkům jiné tabulky.

```php
class CreateFormsTable extends Migration
{
    public function up()
    {
        Schema::create('forms', function (Blueprint $table) {
            $table->id();
            $table->string("slug")->unique(); //use type uuid
            $table->string("name", 128);
            $table->string("description", 512)->nullable();
            $table->dateTime("start_time");
            $table->dateTime("end_time");
            $table->boolean("has_public_results")->default(false);
            $table->boolean("has_private_token");
            $table->timestamps();

            //foreign key: user_id
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
        });
    }
```

Obrázek 4.7: Ukázka části kódu migrace (tabulka pro formuláře)

V Laravel migraci je toto deklarováno pomocí vlastnosti *constrained*, ve které specifikujeme, se kterou tabulkou lze data ze současné tabulky spojovat. Pokud jsou i ostatní tabulky vytvořeny vygenerovanými migracemi, nemusíme ani specifikovat, který sloupec bude z jiné tabulky sloužit k provázání - k tomu Laravel standardně používá sloupec *id*.

Poslední věcí, která byla nutná k zachování referenční integrity dat v databázi pro implementaci projektu, bylo u několika tabulek specifikovat chování při smazání řádku z tabulky, který je provázán s jinou tabulkou. K tomu slouží vlastnost *onDelete*, kde toto chování deklarujeme. V tomto projektu je použit způsob chování „kaskáda“. Ta funguje

v případě mazání tak, že pokud je smazán rodičovský (nadřazený) objekt, tak jsou smazány všichni potomci (podřazené objekty) [50].

4.2.7.1 Testovací data

K tomu, aby bylo možné aplikaci rozumně testovat, je nutné mít v databázi nějaká data. Pokud bychom vždy ručně tato data přidávali, trvalo by to enormní množství času - k těmto účelům v Laravelu existují tzv. Factories a Seeders.

Factories (volně přeloženo jako Továrny) slouží k vygenerování výplňových dat, která jsou použita primárně při vývoji aplikace. Tyto továrny na vymyšlená data jsou poté použity v Seederu (volně přeloženo jako Rozsévač), který provádí všechny úkony jako vymazání všech současných dat databáze a následné naplnění daty z továren.

4.2.8 Pohledy

Pohledy jsou šablony ve formátu PHP (spravované balíčkem „Blade“), pomocí kterých Laravel nativně vykresluje data uživateli. Poté, co jsou takové šabloně data předána (např. pomocí kontroléru), je šablona následně převedena na HTML formát, který je čitelný např. pro webový prohlížeč. Obvykle také obsahuje závislosti na různé přídavné soubory - např. JavaScript nebo CSS soubory.

V tomto projektu (jak již bylo zmíněno) je pro vykreslení ve webovém prohlížeči použit pouze jeden pohled s názvem *app* (viz obrázek 4.8). Tato šablona kromě standardní HTML hlavičky a běžných referenci na javascriptové nebo CSS soubory obsahuje pouze tělo obsahující jediný oddíl s ID *app*. Na tento oddíl díky referenci na ID poté oddělený frontend ve VueJS připojuje další jednotlivé elementy stránky - tyto elementy a další postupy jsou dále popsány ve frontendové části práce v sekci 4.3.

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <title>Title</title>
        <link href="{{ asset('css/app.css') }}" type="text/css" rel="stylesheet">
    </head>
    <body>
        <div id="app"></div>
        <script src="{{ asset('js/app.js') }}"></script>
    </body>
</html>
```

Obrázek 4.8: Šablona pohledu *app* (pro účely této práce upravena)

4.2.9 Průběh jednotlivých činností

V této části jsou popsány postupy, jak se jednotlivé akce zpracovávají na serveru a jak jsou data předávána databázi či uživateli webové stránky. Úkony jsou rozděleny tak, jak jsou zapsány v kontrolérech. Z hlediska kódu je každý úkon metodou kontroléru.

Každá z těchto metod vrací po provedení úkonu většinou předdefinovaný stavový kód označující, zda se činnost provedla úspěšně či nikoliv. Zde (tabulka 4.3) je seznam přímo využitých stavových kódů, které jsou serverem v této implementaci obvykle předávány.

Stavový kód	**Hláška**	**Popis hlášky**
200	OK	Operace proběhla úspěšně.
204	No Content	Operace proběhla úspěšně, ale žádná data nebyla klientovi poslána.
400	Bad Request	Server nerozumí požadavku.
401	Unauthorized	Server vyžaduje autorizaci.
404	Not Found	Požadovaná stránka nebyla nalezena.
410	Gone	Požadovaná stránka již není dostupná.
423	Locked	Požadovaný zdroj je uzamčen.
500	Internal Server Error	Obecná chybová hláška serveru - došlo k neočekávané chybě.

Tabulka 4.3: Využité stavové kódy HTTP [51][52]

Nutno zmínit, že stavový kód „200 (OK)“ je v této implementaci projektu automaticky odesílán v případě, že není v průběhu vyřizování požadavku zaznamenána žádná chyba. Dále v práci proto nebude jeho odesílání zmiňováno.

4.2.9.1 Manipulace s formuláři

Tato část popisuje veškerou administrativu nad formuláři.

4.2.9.1.1 Vytvoření formuláře

Metoda s názvem *store* slouží k vytvoření a uložení formuláře s jeho elementy do databáze. Kromě samotného vytvoření se stará o integritu formátu jednotlivých otázek a validitu příslušných atributů formuláře. Při chybě validace předaných dat (pokud není chyba cíleně ignorována) je nejčastěji vrácen stavový kód 400.

Prvním krokem, který funkce provádí, je ověření, zda je uživatel přihlášen (pokud není, je vrácen stavový kód 401). Poté se provádí rozsáhlá validace, která sestává z kontroly základních informací o formuláři (např. název formuláře či data zveřejněni a ukončení

zveřejnění), samotných elementů formuláře a dalších parametrů validního formuláře. Před těmito úkony je zkontrolováno, zda požadavek od uživatele vůbec obsahuje nějaká data.

Validace základních informací kontroluje název a popisek formuláře (zda obsahují validní řetězec, u názvu samotné vyplnění - jedná se o povinný atribut), data zveřejnění a ukončení zveřejnění formuláře (zda mají hodnoty správný formát, zda datum ukončení zveřejnění je dále v čase než datum zveřejnění a zda se netvoří už expirovaný formulář s časem ukončení zveřejnění dříve v čase, než je aktuální serverový čas), hodnotu vyjadřující přístupnost formuláře (zda bude veřejný nebo privátní) a zda požadavek vůbec obsahuje kolekci otázek. U privátního formuláře jsou také validovány emaily, na které se rozešle pozvánka (tzn. jejich formát).

Další částí je validace elementů. Ty jsou procházeny cyklem, ve kterém se nejprve zjišťuje, zda je vybraný element elementem otázky nebo nové stránky, a kontroluje se správné pořadí na základě vlastnosti *order*. Pokud se jedná o novou stránku, přeskakuje se validace atributů otázky, a pokud má stránka validní hodnotu pro pořadí, je rovnou přemístěna mezi validní elementy.

V případě elementu otázky je to komplikovanější. Kromě validace pořadí přichází na řadu kontroly vlastností, jako jsou typ samotné otázky (zda se jedná o platný typ podporovaný systémem), samotný textový řetězec vyjadřující otázku (zda se jedná o text) či atribut vyjadřující, zda se jedná o otázku povinnou, či nikoliv (kontrola typu). Poté, na základě typu, jsou validovány atributy související s typem otázky. Atributy, které jednotlivé typy otázek mají, jsou rozebrány v sekci 4.1.1.4 - u některých ze samotné podstaty vyplývá, co a jak se u nich bude validovat.

Obecně platí, že u rozsahů délky (tzn. minimální a maximální hodnoty) musí platit, že minimální hodnota je menší než maximální (a to nezávisle na typu - tzn. platí např. i pro datum - i přesto, že jsou data vkládána jako textový řetězec, musí být příslušně převedena a zvalidována). Striktní délka musí být větší než nula. U atributů, kde má být zadána hodnota *true* nebo *false*, kontrolovat validní hodnotu popř. předanou hodnotu převést na validní tvar. Obvykle, pokud se nejedná o povinnou hodnotu, se chyby ignorují - proces pokračuje dále, žádné hodnoty nejsou do těchto atributů uloženy.

Nejkomplikovanější validace probíhá u typu vstupu pro odpověď z výběru předepsaných možných odpovědí. Zde musí být validovány i jednotlivé možnosti odpovědi pro otázku - u nich se kontroluje počet (možností musí jich být více než dvě), zda se jedná o škálu (na základě *true/false* hodnoty *has_hidden_label* - pokud je tato hodnota *true*, očekává se u všech možností číselný popisek, který je pro danou kolekci možností unikátní), vnitřní pořadí (zda se jedná o čísla tvořící posloupnost 0,1,2,3,...) a samotný text vyjadřující odpověď na otázku (zda se jedná o platnou textovou hodnotu). V případě selhání validace chyby nejsou ignorovány a je vrácen příslušný chybový stavový kód. Po této validaci už následuje jen validace rozsahu počtu odpovědí či striktního počtu odpovědí (pokud se jedná o tzv. multiple-choice otázku) - u těchto hodnot platí kromě dříve popsaných

pravidel, že musí být v symbióze s počtem možností odpovědí - tzn. zadané hodnoty musí být menší než celkový počet možností.

Po validaci zmíněných dat přichází na řadu kontrola dalších parametrů. Tím je validace pořadí (elementy musí mít v atributu pořadí číslo, které s ostatními tvoří posloupnost 0,1,2,3,...) a kontrola, zda v kolekci otázek existuje alespoň jedna otázka povinná. Dále je zavedena kontrola umístění elementů nové stránky - pro ty platí, že nesmí stát na začátku a na konci kolekce elementů a že v kolekci nesmí stát dva a více vedle sebe. V případě výskytu chyby je vrácen příslušný chybový stavový kód.

Pokud jsou všechna data validní, tak se začne vykonávat kód pro předejití poškození integrity dat v databázové transakci, kde je v případě tvoření formuláře nejprve vygenerován unikátní identifikátor pomocí balíčku „ramsey/uuid“ ve tvaru UUID4. Poté se postupně vytváří příslušné objekty pro jednotlivé prvky formuláře, kterým se předávají zvalidovaná data. Tyto objekty (resp. jejich data) jsou poté ukládány do databáze. Po tomto úkonu se v případě privátního formuláře rozešlou pozvánky na příslušné emaily.

4.2.9.1.2 Úprava formuláře

Metoda s názvem *update* slouží k editaci formuláře, který zatím nebyl zveřejněn. Díky této metodě lze kromě základních informací měnit i samotné otázky. Co v této metodě ale nelze provést, je úprava přístupnosti formuláře a případné přidání emailů k rozeslání pozvánky (k tomu slouží metoda v sekci 4.2.9.1.4).

Při volání metody se nejprve ověří, zda je uživatel přihlášen (pokud není, je vrácen stavový kód 401) a zda požadovaný formulář existuje (pokud ne, je vrácen stavový kód 404). Poté přichází na řadu kontrola časů - kontroluje se, zda už nebyl formulář zveřejněn na základě data zveřejnění formuláře (atribut musí být dále v čase než je serverový čas). Pokud tato podmínka není splněna, je vrácen stavový kód 400.

Dále je spuštěna databázová transakce, ve které se nejprve vygeneruje dočasný identifikátor *slug* formuláře ve tvaru UUID4. Poté se všechna data z požadavku uživatele předají metodě *store* (popsána v sekci 4.2.9.1.1), která přidá do databáze formulář s předanými úpravami. Kromě dat z požadavku se metodě poskytnou i další data: dočasný identifikátor, který slouží k identifikaci přidaného upraveného formuláře a informace o tom, zda byl upravovaný formulář veřejný nebo privátní. Dočasný identifikátor je použit místo vygenerovaného identifikátoru použitého přímo v metodě *store*. Předávaná informace o přístupnosti slouží k tomu, aby upravený formulář tuto vlastnost neztratil a aby nebyly znovu rozeslány pozvánky na příslušné emaily, což je nativní chování metody *store*.

Poté, pokud nebyl uživatelský požadavek validní dle metody pro přidání formuláře, je vrácen příslušný stavový kód pocházející přímo ze zmíněné metody. Pokud ale vše v metodě *store* proběhlo bez problému, následují další činnosti.

Pokud se jedná o veřejný formulář, původní (neupravený) formulář (a s ním i další objekty na něj navázané) je smazán pomocí metody *destroy* (popsána v sekci 4.2.9.1.6) a u upraveného formuláře je přepsán identifikátor na hodnotu identifikátoru, kterou disponoval neupravený formulář. V případě, že je upravený formulář privátní, se prvně do dočasné proměnné uloží všechny přístupové tokeny. Poté se původní formulář smaže a nahradí se identifikátor stejně jako u veřejného formuláře. Nakonec se vytvoří kopie tokenů na základě dat v dočasné proměnné zmíněné dříve, které se liší jen v hodnotě ID formuláře (to, jelikož byl de facto vytvořen nový formulář, musí odkazovat na upravený formulář). Při neočekávaných chybách je vrácen stavový kód 500.

4.2.9.1.3 Duplikování formuláře

Metoda s názvem *duplicateWithAuth* slouží k duplikování formuláře - tzn. lze vytvořit nový formulář, který obsahuje stejné otázky a vlastnosti (tj. jméno formuláře, datum zveřejnění, datum ukončení zveřejnění,...), jako formulář, u kterého byla požadována duplikace. Zmíněné vlastnosti při tomto procesu lze modifikovat, otázky ne.

Metoda funguje tak, že prvně ověří, zda je uživatel přihlášen (pokud ne, vrátí stavový kód 401), poté zkontroluje zda požadovaný formulář existuje a zda patří přihlášenému uživateli. Dále jsou validovány všechny předané hodnoty vlastností formuláře, které již byly zmíněné - tzn. zda vůbec v požadavku na duplikaci existují a zda jsou správného typu (v případě jakékoliv očekávané chyby je vrácen kód 400).

U hodnot pro čas zveřejnění a ukončení zveřejnění se validuje příslušný formát data, zda není čas zveřejnění větší (déle v čase) než čas ukončení zveřejnění či zda není vytvářen už expirovaný formulář (tzn. aktuální serverový čas je déle v čase než čas ukončení zveřejnění).

Poté probíhá kontrola nastavení přístupnosti formuláře - při duplikaci lze změnit tento atribut. Pokud je tento atribut nastaven jako *true* (tzn. formulář je privátní, lze k němu přistupovat jen s tokenem), kontrolují se předané emaily (zda mají validní tvar), na které se bude rozesílat pozvánka.

Nakonec se vygeneruje nový identifikátor ve tvaru UUID4 a metoda se pokouší příslušná data uložit do databáze. Zde vzniká nový objekt formuláře, kterému jsou předány příslušné vlastnosti (tj. ID uživatele, vygenerovaný identifikátor a data z požadavku duplikace). S tím podobně vznikají i příslušné objekty pro uložení jednotlivých elementů formuláře (tj. otázky nebo nové stránky), jejichž vlastnosti jsou přebírány a kopírovány z duplikovaného formuláře. Nakonec, pokud je formulář nastaven jako privátní, jsou rozeslány pozvánky na zvalidované emaily. Tento proces probíhá v databázové transakci.

Při úspěchu je vrácena odpověď s identifikátorem nově vzniklého formuláře (duplikátu). Ten je určen pro frontendovou část projektu k přesměrování.

4.2.9.1.4 Úprava přístupnosti formuláře

Metoda s názvem *updateAccess* slouží k úpravě přístupnosti formuláře. V ní lze nastavit, zda je formulář veřejný, nebo privátní (s tím je možné nastavit, na které emaily lze dodatečně poslat pozvánku či kterým emailům s pozvánkou odebrat přístup ve formě invalidace tokenu).

V rámci zpracování dat je nejprve kontrolováno, zda požadavek obsahuje vůbec nějaká data (pokud ne, je vrácen stavový kód 400). Poté se kontroluje přihlášení uživatele (pokud není přihlášen, je vrácen stavový kód 401) a zda požadovaný formulář s příslušným vlastníkem existuje (pokud ne, je vrácen stavový kód 404). Další akce jsou prováděny jen pokud existuje v požadavku uživatele atribut *has_private_token* s platnou hodnotou (pokud neexistuje, je vrácen stavový kód 400). Po těchto kontrolách se dále celý proces dělí na dvě logické větve.

První větev je podmíněna tím, že se zmíněný předaný atribut *has_private_token* se rovná *true* (tzn. formulář bude privátní). V takovém případě se nejprve zvalidují předané emaily, na které má být odeslána nová pozvánka (pokud dojde k chybě ve validaci samotných emailů, je vrácen kód 400). Poté, pokud byl měněný formulář veřejný (tzn. před tímto úkonem se jeho hodnota *has_private_token* rovnala *false*), se formuláři v databázi přepisuje zmíněný atribut *has_private_token* na *true* a vytváří se nové tokeny pro předané emaily, na které je poté rozeslána pozvánka. Pokud byl ale měněný formulář už privátní, tak jsou, společně s vytvořením nových tokenů a rozesláním pozvánek, zneplatněny pozvánky, které byly zaslané na majitelem předané emaily - tzn. po validaci těchto dat jsou z databáze vymazány tokeny, ke kterým jsou vázány příslušné emaily z uživatelského požadavku. Uživatelům, kterým byla pozvánka zneplatněna, se žádný oznamovací email nezasílá.

Druhá větev je podmíněna tím, že zmíněný předaný atribut *has_private_token* se rovná *false* a atribut požadovaného formuláře *has_private_token* se rovná *false* (tzn. formulář bude převeden na veřejný). V takovém případě proběhne triviální smazání všech privátních tokenů, které se na příslušný formulář váží. Žádný oznamovací email se uživatelům s pozvánkou neposílá.

Veškeré databázové operace spojené se zmíněnými úkony probíhají pro zachování integrity dat v transakcích.

4.2.9.1.5 Zveřejnění výsledků formuláře

Metoda s názvem *publishResults* slouží k modifikaci přístupu k výsledkům formuláře pro běžnou veřejnost. Zde lze nastavit, zda jsou výsledky veřejné či nikoliv. Pokud jsou výsledky zveřejněny, lze zvolit, které otázky jsou skryté a které jsou zveřejněné.

Funkce nejprve zkontroluje, zda požadavek obsahuje vůbec nějaká data (pokud ne, je vrácen stavový kód 400). Poté kontroluje přihlášení uživatele (pokud není přihlášen, je vrácen stavový kód 401) a zda požadovaný formulář s příslušným vlastníkem existuje (pokud ne, je vrácen stavový kód 404). Po těchto kontrolách jsou validována předaná data v požadavku na server.

V rámci zmíněných dat se nejprve validuje předaný atribut *has_public_results*, který stanovuje, zda má formulář veřejné výsledky - tzn. zda má platnou hodnotu. Poté, pokud je předchozí hodnota *true* (tzn. výsledky budou zveřejněny), jsou procházena a validována všechna data z uživatelova požadavku, která představují jednotlivé otázky a jejich atribut zveřejnění. Zde se kontroluje, zda otázky vůbec náleží požadovanému formuláři (pomocí *id*) - pokud ano, jsou dále předávány společně s informací, zda mají být tyto otázky veřejné, nebo ne (neplatná data v této části jsou ignorována). Zároveň musí platit, že v případě zveřejnění výsledků formuláře musí být alespoň jedna otázka veřejná (pokud toto neplatí, je vrácen stavový kód 400).

Poté se metoda pokouší tato data uložit do databáze. Nejprve přepisuje atribut formuláře *has_public_results*, poté, na základě tohoto atributu, manipuluje s jednotlivými elementy. Pokud je zmíněný atribut *true*, tak jsou jednotlivé elementy modifikovány podle zvalidovaných dat (tzn. pokud mají být veřejné, je jim nastaven stejnojmenný atribut *has_public_results* na true, v opačném případě na *false*). Pokud je ale zmíněný atribut formuláře *false*, je všem elementům zmíněný atribut zveřejnění nastaven na *false* (tzn. všechny odpovědi jsou skryty). Celý proces probíhá v transakci.

4.2.9.1.6 Mazání formuláře

Metoda s názvem *destroy* slouží k odstranění formuláře a jeho přidružených elementů.

V koncepci je metoda velice jednoduchá - poté, co se zkontroluje, zda je uživatel přihlášen (pokud ne, je vrácen stavový kód 401) a zda vůbec formulář s předaným *slug* identifikátorem patřící přihlášenému uživateli v databázi existuje (pokud ne, je vrácen stavový kód 404), je pomocí příslušné nativní metody *delete* smazán. Tento proces také probíhá v transakci.

4.2.9.1.7 Vrácení všech formulářů uživateli

Metoda s názvem *index* je určena k vrácení všech formulářů, které patří přihlášenému uživateli. To slouží k hydrataci úvodní stránky uživatele - jedná se o přehled všech vlastněných formulářů.

Tato funkce nejprve ověří, zda je uživatel přihlášen. Pokud přihlášen je, vrátí uživateli všechny jeho formuláře bez elementů formuláře a bez dat o uživateli, které se obecně k datům připojují. Tento set formulářů je seřazen podle doby vytvoření formuláře sestupně. Pokud přihlášen není, je vrácen stavový kód 401.

4.2.9.1.8 Vrácení informací o formuláři majiteli

Metoda s názvem *showWithAuth* má za účel vrátit celý formulář a data s ním spojená pro jeho vlastníka. Tato data poté slouží k hydrataci stránky s přehledem informací o formuláři.

Funkce nejprve zkontroluje, zda je uživatel přihlášen. Pokud není, je navrácen stavový kód 401. V případě, že je uživatel přihlášen, se v databázi vyhledává formulář na základě předaného identifikátoru *slug*. Pokud formulář není nalezen, je vrácen stavový kód 404, a pokud nepatří přihlášenému uživateli, je vrácen kód 401.

Po zmíněných kontrolách se k objektu formuláře připojují tři atributy: *is_expired* (rovná se *true*, pokud je serverový čas dále v čase než čas ukončení zveřejnění), *was_already_published* (rovná se *true*, pokud je serverový čas dále v čase než čas zveřejnění formuláře) a *is_opened* (rovná se *true*, když neplatí *is_expired* a zároveň platí *was_already_published* - tzn. když je formulář dostupný k vyplnění). Po připojení těchto vlastností je celý formulář s příslušnými vlastnosti vrácen.

Pokud je formulář nastaven jako privátní, tak jsou k celému objektu formuláře ještě připojeny záznamy o privátních tokenech. Tyto záznamy slouží pro vlastníka formuláře k výpisu emailů, kterým byla poslána pozvánka a které mají v době vykonávání této metody oprávnění k vyplnění formuláře. Připojené záznamy o tokenech samotný token neobsahují (resp. je skrytý pro vlastníka formuláře) - to slouží k tomu, aby nebylo možné dohledat, který token patří ke kterému emailu. Zároveň to znemožňuje jednoduše odpovědět za člověka, kterému byla odeslána pozvánka k vyplnění.

4.2.9.1.9 Vrácení konkrétního veřejného formuláře

Metoda s názvem *show* je určena k vrácení konkrétního veřejného formuláře. To slouží k hydrataci stránky určené k vyplnění veřejného formuláře.

V rámci celého procesu funkce nejprve ověří, zda v databázi existuje formulář s požadovaným identifikátorem *slug*. Pokud ne, je vrácen stavový kód 404. Pokud ale nalezen je, provádí se několik kontrol - když kontrolami projde, je navrácen formulář s příslušnými elementy.

První se kontroluje, zda zadaný identifikátor neukazuje na privátní formulář - na tento formulář lze přistoupit jen přes token, nikoliv přes zmíněný identifikátor. Pokud se potvrdí přístup přes identifikátor, odešle se jen stavový kód 400.

Dalším faktorem je čas, kdy k formuláři chce uživatel přistoupit. Ověřuje se podle uložených dat o formuláři z databáze, zda je už formulář zveřejněn (pokud není, je vrácen stavový kód 423), nebo zda už formulář neexpiroval (pokud expiroval, je vrácen stavový kód 410). Aktuální čas se bere podle serveru, na kterém je aplikace nasazena.

4.2.9.1.10 Vrácení konkrétního privátního formuláře

Metoda s názvem *privateShow* funguje podobně jako metoda *show* popsaná v sekci 4.2.9.1.9 a slouží k hydrataci stránky určené k vyplnění privátního formuláře.

Oproti zmíněné předchozí metodě se místo identifikátoru *slug* nejprve vyhledává uživatelem předaný token v databázi. Pokud nalezen není, je vrácen stavový kód 400. Pokud nalezen je, provádí se stejné kontroly časů jako v metodě v sekci 4.2.9.1.9. Liší se ale v kontrole tokenu - zde je zjišťováno, zda není token už expirovaný resp. zda už pro tento token nebyla zaznamenána odpověď. Pokud expirovaný je, tak se odesílá jen stavový kód 410. Pokud požadavek na privátní formulář projde bez chyby, je odeslán formulář s příslušnými elementy - identifikátor *slug* se pro koncového uživatele skrývá.

4.2.9.2 Interakce s odpovědmi na formuláře

Tato část popisuje veškerou administrativu nad zaznamenáváním odpovědí k formuláři.

4.2.9.2.1 Zaznamenávání odpovědi na veřejný formulář

Metoda s názvem *store* slouží k zaznamenání všech odpovědí na příslušný veřejný formulář. V případě očekávané chyby je nejčastěji vrácen chybový kód 400 - proto dále v textu není zmiňován.

První věcí, která se provádí, je ověření, zda existuje formulář, na který lze odpovědět, na základě předaného identifikátoru *slug* (v případě této chyby je vrácen stavový kód 404). Také se kontroluje, zda se nejedná o privátní formulář - na ten nelze přes identifikátor

odpovědět (k tomu je určena metoda *privateStore* popsána v sekci 4.2.9.2.2). Dále se kontroluje, zda je formulář už zveřejněný a zároveň zda není expirovaný (serverový čas v okamžiku, kdy je odeslána odpověď, musí být v intervalu mezi datem zveřejnění a datem ukončení zveřejnění). Po těchto operacích je formulář uložen v dočasné proměnné pro další úkony popsané níže.

Poté je inicializována vnitřní metoda *getCorrespondingValue* sloužící k nalezení příslušných dat a ucelení formátu předaných informací z požadavku na server. Té jsou předávány tyto parametry: *request* (celý požadavek od uživatele), *type* (řetězec identifikující typ otázky) a *specificInputId* (*id* otázky, pro kterou je hledána odpověď). Metoda na základě předaných dat projde celý požadavek, ve kterém jednotlivé záznamy odpovědí zpracuje (tzn. zkontroluje jejich formát). Pokud je formát platný, testují se proti sobě příslušná data z požadavku a předaná data v parametrech metody (tzn. zda se typ a ID otázky z požadavku a z parametru rovnají). Pokud je nalezena taková shoda, jsou vrácena data (tj. ID otázky, typ otázky a předaná hodnota/odpověď) připravená ve formátu asociativního pole určená k další validaci. Pokud shoda nalezena není, jsou vrácena stejná data, jen bez předané hodnoty (pro případy otázek, u kterých není odpověď povinná).

Dále jsou cyklem procházeny všechny elementy zodpovězeného formuláře. Cyklus postupně zkouší najít příslušnou hodnotu pro každou otázku. Krok cyklu je větven do několika částí, které se odvíjejí od typu otázky. V každé větvi je poté, na základě známého typu otázky, prohledáván požadavek od uživatele pomocí metody *getCorrespondingValue* k nalezení příslušné odpovědi, která je převedena do pracovního formátu. Ta je poté validována podle předepsaných pravidel příslušné otázky, jež byla deklarována při tvoření formuláře (např. maximální délka předané textové odpovědi) v metodě *store* ze sekce 4.2.9.1.1 - tzn. pokud je povinná, musí mít platnou hodnotu, pokud povinná není, může mít kromě platné hodnoty hodnotu *null*. Komplikovanější, ale v podstatě stejná, je validace vstupu pro odpověď z výběru předepsaných možných odpovědí, kde se podobně jako v metodě *getCorrespondingValue* v dalším vnořeném cyklu testují i příslušné možnosti odpovědi (zda vůbec k formuláři patří a zda se jedná o platnou hodnotu).

Po tomto validačním procesu následuje databázová transakce, ve které se celá odpověď ukládá do databáze. Pro celou odpověď je nejprve vytvořen záznam o vyplnění formuláře, kterému je předáno ID formuláře, na který byla odpověď směřována. Poté se vytváří na základě validního setu odpovědí k příslušným vstupům příslušné databázové objekty, které jsou zaobaleny záznamem o vyplnění formuláře a do databáze uloženy.

4.2.9.2.2 Zaznamenávání odpovědi na privátní formulář

Metoda s názvem *privateStore* slouží k zaznamenání všech odpovědí na privátní formulář.

V rámci funkčnosti je tato metoda opravdu velmi spjatá s metodou popsanou v sekci 4.2.9.2.1. Prvním krokem je ověření, zda použitý přístupový token existuje (pokud ne, je vrácen stavový kód 400) a zda pomocí tohoto tokenu už nebyla zaznamenána odpověď (pokud ano, je vrácen stavový kód 410). Poté se na základě předaného tokenu musí vyhledat formulář, kterému token náleží, a z něj získat jeho identifikátor *slug*. Následuje použití metody *store* ze sekce 4.2.9.2.1 - té je předán celý uživatelský požadavek, identifikátor nalezený v předchozím kroku a informace o tom, že se odpovídá na privátní formulář (metoda standardně počítá s tím, že se odpovídá na veřejný formulář, takže bez předání této informace by odpověď blokovala). Na základě stavového kódu z vnitřně volané metody *store* se provádí další činnosti.

V případě, že byla odpověď zaznamenána úspěšně (byl vrácen stavový kód 200), se v databázové transakci přepíše atribut tokenu, který vyjadřuje, že už pomocí něho byla zaznamenána odpověď, na hodnotu *true* (tzn. byl použit, nelze s tímto tokenem dále odpovídat). Pokud se ale vyskytla chyba v průběhu metody *store*, tak je uživateli vrácen touto metodou předaný chybový kód.

4.2.9.2.3 Vrácení všech výsledků

Metoda s názvem *index* slouží k vypsání všech odpovědí na formulář majiteli požadovaného formuláře.

Funguje tak, že po kontrole autentizace (u chyby vrácen stavový kód 401) a autorizace (u chyby vrácen stavový kód 404) se k nalezenému formuláři postupně připojí všechny existující odpovědi k příslušným typům otázek pomocí vazeb, které lze vytvářet na Laravel modelech (viz obrázek 4.9). Poté, pokud byla nalezena alespoň jedna odpověď na formulář, je vrácen celý objekt formuláře, ke kterému je připojen set dat obsahující všechny odpovědi. Pokud nebyla zaznamenána žádná odpověď, je vrácen stavový kód 204 (nejedná se o chybu, jen neexistují žádná data k předání).

```
$wantedFormResults = Form::where(['slug' => $slug, 'user_id' => $user->id])->with(
    'formElements.inputElement.textInput.textInputAnswers',
    'formElements.inputElement.numberInput.numberInputAnswers',
    'formElements.inputElement.dateInput.dateInputAnswers',
    'formElements.inputElement.booleanInput.booleanInputAnswers',
    'formElements.inputElement.selectInput.selectInputChoices.selectInputAnswers',
)->first();
```

Obrázek 4.9: Ukázka části kódu metody *index* pro získání odpovědí na formulář

4.2.9.2.4 Vrácení veřejných výsledků

Metoda s názvem *publicIndex* slouží k vypsání všech odpovědí na formulář. Tento výpis je dostupný jen v případě, že majitel formuláře tuto funkci aktivoval (popsáno v sekci 4.2.9.1.5) - poté jsou vybrané výsledky dostupné pro kohokoliv.

Tato metoda funguje velmi podobně jako metoda *index* popsaná v sekci 4.2.9.2.3. Oproti zmíněné metodě nekontroluje přihlášení uživatele, ale zda jsou výsledky dotazovaného formuláře nastaveny jako veřejné. Z podstaty věci také omezuje viditelnost vybraných otázek na základě hodnoty *has_public_results*, kterou musí disponovat všechny otázky - pokud je nastavena na *false*, tak je otázka z výstupního setu odebrána. Pokud jsou všechny nastaveny jako skryté, odesílá se stejný stavový kód jako při situaci, kdy neexistuje žádná odpověď na formulář. Jinak se tato metoda oproti zmíněné metodě *index* neliší.

4.2.9.2.5 Export výsledků

Metoda s názvem *export* je určena k exportování všech odpovědí do souboru formátu MS Excel (s příponou .xlsx). Tato data mají sloužit při správném nastavení formuláře a jeho odpovědí např. ke korelační analýze (především u škálových typů otázek). K tomuto úkonu je použit přídavný balíček „maatwebsite/excel“ (především třída *Excel*), který po předání příslušných dat celý exportní proces obstarává (viz sekce 4.2.5).

Metoda funguje tak, že nejprve volá metodu *index* (popsána v sekci 4.2.9.2.3), které předává identifikátor formuláře z uživatelského požadavku. Poté, pokud je ze zmíněné metody navrácen objekt formuláře (tzn. že celý proces proběhl úspěšně - byla předána platná kolekce odpovědí), jsou příslušná data předána metodě *download* na statické třídě *Excel*, která vygeneruje příslušný soubor. Pokud je z metody *index* vrácen jen stavový kód, tak je uživateli předán společně s příslušnou chybovou hláškou.

Samotný export dat je proveden tak, že se zmíněné metodě *download* předá instance objektu, která zaobaluje předaná data k exportu a definuje, jak budou v souboru sázena. Metoda *download* poté vytvoří soubor, který je poslán uživateli. Instance objektu je tvořena ze třídy *FormCompletionsExport*, která byla k tomuto účelu připravena - implementuje rozhraní *FromArray* (z balíčku pro export) a obsahuje příslušnou metodu *array*, která vybraná data zpracovává a vrací ve formě pole polí (reprezentuje sloupce a řádky v tabulce). Metoda určuje formát dat takto (viz obrázek 4.10):

- Na prvním řádku v první buňce (roh tabulky) je umístěn samotný název formuláře.
- Na druhém řádku je nejprve definován sloupec *Completion ID* (ten obsahuje ID pro každou zaznamenanou odpověď - jedná se o *id* databázového objektu pro vyplnění formuláře viz sekce 4.1.1.6). Poté jsou v řádku sázeny jednotlivé otázky formuláře.
- Na dalších řádcích jsou vypsána data z odpovědi - tzn. na prvním místě ID vyplnění formuláře, poté příslušné odpovědi pod příslušnými otázkami.

Dotazník spokojenosti			
Completion ID	Jakou službu jste u nás využíval? *	Jak byste ohodnotil naši službu? *	Proč jsme podle vás lepší než konkurence?
9	Bagrování děr	2	Skvělá radlice u bagru;Cena
10	Zapujčení auta	5	
11	Bagrování děr	1	Kvalitní přístup;Cena
12	Wellnes	3	Kvalitní přístup
13	Wellnes	1	Skvělá radlice u bagru;Cena
14	Bagrování děr	4	
15	Zapujčení auta	1	Cena
16	Wellnes	3	

Obrázek 4.10: Ukázka vyexportovaných dat (pro potřeby této práce upraveno)

Zpracování odpovědí nejprve probíhá tak, že se shromáždí všechna ID vyplnění formuláře pro požadovaný formulář - tím se vytvoří připravené „chlívečky“ pro odpovědi (jednorozměrné pole), které jsou od sebe oddělené právě pomocí ID vyplnění formuláře (co *chlíveček*, to řádek s odpovědmi). Dále se na první index „chlívečku“ umístí samotné ID vyplnění. Poté se prochází jednotlivé odpovědi, které se postupně ukládají do zmíněných „chlívečků“, přičemž je dbáno na to, aby se odpovědi ukládaly pod správné otázky. U otázek s možnostmi odpovědí v případě multiple-choice jsou data rozdělována středníkem. Pokud je otázka nepovinná a nebyla zodpovězena, buňka ve výstupu bude prázdná.

Nakonec se otázky rozmístěné do „chlívečků“ ještě upraví (tzn. kde se nachází logické hodnoty *true* nebo *false*, tam je umístěn textový ekvivalent „true“ nebo „false“). Podobná úprava probíhá i u samotných otázek v 2. řádku tabulky (pokud je otázka povinná, je jí na konec přidán znak hvězdičky „*“).

4.2.9.3 Interakce s uživateli

Tato část popisuje veškerou administrativu nad uživatelskými účty. Nutno zmínit, že v projektu byly pro tyto účely použity nativní materiály a funkce Laravelu (např. model uživatele, migrace tabulky uživatelů a validační funkce). V rámci validačních funkcí zde nebudou popsány stavové kódy, které při chybě vracejí, jelikož se jedná o funkcionalitu spravovanou přímo frameworkem Laravel.

4.2.9.3.1 Registrace uživatele

Metoda *register* slouží k vytvoření nového uživatelského účtu.

V rámci funkčnosti metoda nejdříve přijme požadavek od uživatele, ve kterém očekává určité parametry (přezdívka uživatele, email, heslo a potvrzení hesla). Pokud tyto parametry dostane a jsou platné (tzn. že splňují požadavky jako že se jedná o platné textové hodnoty, že email musí být u uživatele unikátní a že je heslo alespoň 8 znaků dlouhé), vytvoří se hash hesla (to je zašifrováno pomocí funkce *bcrypt*) a všechna data o uživateli se uloží do databáze.

4.2.9.3.2 Přihlášení uživatele

Metoda *login* slouží k přihlášení uživatele k uživatelskému účtu.

Funkce funguje tak, že nejprve (podobně jako v metodě *register* v sekci 4.2.9.3.1) zvaliduje příchozí data (zde email a heslo) a až poté, co se ověří, že jsou údaje platné (pomocí statické třídy *Auth*), se odesílá objekt s daty o přihlášeném uživateli.

4.2.9.3.3 Odhlášení uživatele

Metoda *logout* slouží k odhlášení uživatele - tedy k invalidování relace. Obsahuje jen volání metody *logout* na statické třídě *Auth*, která zařizuje veškerou administrativu nad přihlášenými uživateli a tedy i nad tímto úkonem.

4.2.9.3.4 Vrácení informací o uživateli

Metoda *show* přihlášenému uživateli vrátí data o jeho účtu resp. jeho osobní údaje jako je např. přezdívka nebo email.

Funguje tak, že pokud je uživatel přihlášen, jsou jednoduše vrácena jeho osobní data z databáze. Pokud není přihlášen, je vrácen stavový kód 401.

4.2.9.3.5 Vymazání účtu uživatele

Metoda *destroy* slouží k odstranění uživatelského účtu na základě požadavku vlastníka profilu. Vlastník k tomuto úkonu samozřejmě musí být přihlášen.

Jelikož k tomuto úkonu musí být předáno současné heslo k účtu, je prvně kontrolováno právě to resp. jestli se jedná o validní textový řetězec. Poté je zjišťováno, zda je vůbec nějaký uživatel přihlášen (pokud není, je vrácen stavový kód 400). Pokud ano, je kontrolováno, zda se heslo shoduje s heslem v databázi (pokud ne, je vrácen stavový kód 400). Pokud se hesla shodují, je uživatel z databáze smazán.

4.2.9.3.6 Změna hesla

Metoda *changePassword* slouží ke změně hesla za předpokladu, že uživatel stávající heslo zná.

V rámci průběhu celého úkonu metody, prvně se kontroluje, zda je vůbec nějaký uživatel přihlášen. Pokud není, je vrácen stavový kód 401. Pokud ale přihlášen je, tak nejprve dochází k validaci dat (podobně jako v sekci 4.2.9.3.1). Dále se kontroluje, zda je původní heslo shodné s předaným původním heslem, zda není staré a nové heslo stejné a zda je nové heslo shodné s potvrzením tohoto hesla. Pokud všemi kontrolami projde, dochází k vytvoření nového hashe hesla, kterým je přepsán starý hash uživatele.

4.2.9.3.7 Resetování zapomenutého hesla uživatele

Tento proces je obhospodařován dvěma metodami - *forgotPassword* a *resetPassword*. Celý proces resetování hesla probíhá tak, že uživatel odešle požadavek na metodu *forgotPassword*, která mu na jeho email pošle odkaz k resetování hesla - ten vede na stránku, kde si zvolí nové heslo. Poté, co nové heslo vyplní, odesílá požadavek na metodu *resetPassword*, která heslo za splnění všech podmínek (tj. validace dat a správnost obnovovacího tokenu) změní.

Metoda *forgotPassword* tedy v požadavku přijímá jen email, který je nejprve validován. Poté je na něj odeslán odkaz k resetování hesla. Pokud se odeslání nezdaří, je vrácen stavový kód 400.

K metodě *resetPassword* je přistupováno právě z odkazu poslaného pomocí metody *forgotPassword*. Metoda *resetPassword* přijímá ověřovací token předaný v odkazu v emailu, samotnou emailovou adresu a heslo s potvrzením. Pokud jsou předaná data validní, je vygenerován nový hash hesla, který přepíše stávající hash. Pokud se při resetování hesla vyskytne chyba, je vrácen stavový kód 400.

4.3 Frontend

Frontendová část tohoto projektu zajišťuje především komunikaci mezi uživatelem a serverem pomocí prostředí, díky kterému je tato komunikace snazší a obecně přívětivější. Serverová část očekává příchozí požadavky od této části na cesty, které byly už dříve definovány (viz sekce 4.2.2).

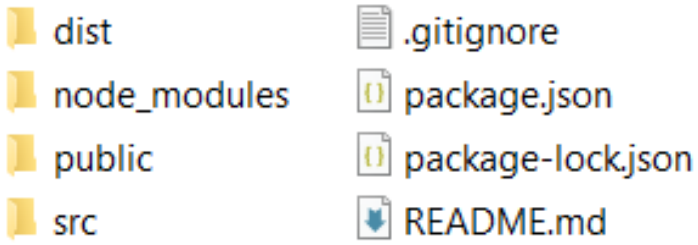
V rámci této implementace (a z podstaty využitých technologií) probíhá komunikace s API (viz sekce 4.2), které se tato část snaží poskytovat validní data. Slovo „snaží“ je zmíněno proto, jelikož se na ni v reálném světe nemůžeme úplně spoléhat - frontendovou část může uživatel s dostatečnými znalostmi a zkušenostmi modifikovat, čímž může i změnit způsob komunikace se serverem. To ale neznamená, že na ní nemůže probíhat jakákoliv validace a kontrola dat - frontend pro správnou funkčnost celé aplikace musí serveru ve výchozím (uživatelem nezměněném) stavu backendu poskytovat validní data - backend je ale musí preventivně validovat taky.

Uživatelské prostředí musí být připraveno tak, aby bylo příjemné k užívání a ergonomické. Zároveň by nemělo být nějak složité a přehlcené ovládacími prvky, což by mohlo uživatele od používání aplikace odradit. Dále by toto prostředí mělo uživatele „vést“ úkonem, který chce uživatel provést - tzn. pokud zadává neplatná data, tak ho upozornit a rozhodně počkat na opravu dat před odesláním požadavku na server. Uživatelské prostředí, které v rámci tohoto projektu vzniklo, je určeno primárně pro vykreslování ve webovém prohlížeči.

V následující části jsou popsány jednotlivé frontendové komponenty a principy, díky kterým může celá aplikace fungovat.

4.3.1 Obecná struktura VueJS aplikace

Nejprve je nutné rozebrat obecnou strukturu VueJS projektu.



Obrázek 4.11: Obecná struktura nově vygenerovaného VueJS projektu

Jak můžeme na obrázku 4.11 vidět, celý projekt je složen z mnoha složek a souborů. Podrobný rozbor všech souborů a složek není předmětem této práce, proto jsou nejdůležitější zmíněné části popsány níže jen obecně.

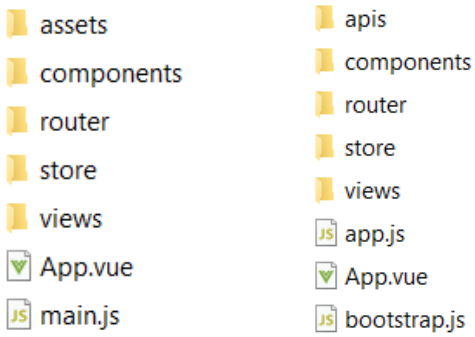
- Složka *dist* obsahuje zkompilovaný kód aplikace. [53]
- Složka *node_modules* obsahuje závislosti a soubory přídavných balíčků NPM (viz sekce 3.1.4), které aplikace používá. [54]
- Složka *public* obsahuje obecně veřejné soubory (např. favicon). [54]
- Složka *src* obsahuje všechny soubory se zdrojovým kódem, ze kterých se poté tvoří zkompilovaný kód celé aplikace. [54]
- Soubor *package.json* obsahuje informace o přídavných balíčcích spravovaných pomocí NPM (viz sekce 3.1.4). [54]

Popsaná struktura výše je standardní pro oddělenou frontendovou aplikaci. To v implementaci tohoto projektu ale neplatí, jelikož je frontend přímo zaintegrován v Laravel projektu (viz sekce 4.2.1). Jsou zde tyto rozdíly:

- Složka *src* je nahrazena složkou *js* ve složce *resources*.
- Složka *public* je umístěna v kořenovém adresáři Laravel projektu a je sloučena se složkou *dist*.
- Složka *node_modules* společně se souborem *package.json* je také umístěna v kořenovém adresáři Laravel projektu.

Celkovou administrativu nad celým VueJS projektem poté zajišťuje modul Laravel Mix podle konfiguračního souboru webpack.mix.js [55]. Ten je také umístěn v kořenovém adresáři projektu. Díky celému tomuto přístupu může backend i frontend běžet jako jedna aplikace na jedné doméně.

Jak již bylo zmíněno, zdrojový kód, tedy nejdůležitější část frontendové aplikace, leží standardně ve složce *src* - v tomto projektu ve zmíněné složce *js*. Oproti běžné samostatné VueJS aplikaci je zde (ve složce *js*) největší rozdíl v pojmenování souborů - soubor *main.js* byl přejmenován pro potřeby Laravelu na *app.js*. Také se navíc v této složce nachází Laravelem vygenerovaný soubor bootstrap.js pro další konfiguraci při kompilaci. Nakonec byla kvůli nevyužití odebrána složka *assets* a pro další účely přidána složka *apis* (viz obrázek 4.12).



(a) (b)

Obrázek 4.12: Porovnání obsahu složek *src* (a) a *js* (b)

Každý soubor a složka ve zmíněném adresáři *js* má určitý význam. Ty jsou představeny níže (soubor bootstrap.js byl již popsán výše). Další rozbor jednotlivých souborů ve zmíněných složkách je obsažen v dalších sekcích této práce.

- Složka *apis* obsahuje prostředky pro komunikaci s API (tedy s backendem).
- Složka *components* obsahuje všechny komponenty, které jsou využívány v celé aplikaci.
- Složka *router* obsahuje prostředky pro přesměrovávání a definování cest ve frontendové aplikaci.
- Složka *store* obsahuje nástroje pro držení dat ve webovém prohlížeči.
- Složka *views* obsahuje veškeré pohledy, na které se můžeme pomocí definované cesty dostat.
- Soubor *app.js* je kořenový soubor aplikace shromažďující všechny reference na ostatní soubory (zabaluje celou aplikaci).

- Soubor *App.vue* je kořenová šablona, která přijímá pohledy na základě cesty, které poté vykresluje (tvoří aplikační kostru HTML).

4.3.1.1 Souhrn využitých balíčků

Pro přehlednost je zde vytvořena kapitola shrnující vybrané externí balíčky, které byly do frontendové části přidány, a jejich základní účel a funkcionalita.

- Balíček „@braid/vue-formulate“ (dále jako „VueFormulate“) je použit pro veškeré formulářové prvky. Kromě zprostředkování hotových komponent pro jednotlivé elementy formuláře také nabízí např. validaci dat.
- Balíček „vue-chartjs“ (zastřešující populární balíček „chart.js“) je použit pro grafické vyobrazení dat ve formě grafů.
- Balíček „randomcolor“ je použit pro generování náhodných barev (využito u grafů k dynamickému rozlišení dat).
- Balíček „uuid“ je použit pro generování unikátního identifikátoru (především funkce *uuidv4*, např. k označování vytvořených otázek při tvorbě nového formuláře).
- Balíček „vue-js-modal“ je použit k vytváření a správě modálních oken.
- Balíček „vue-toasted“ je použit pro zobrazování zpráv o průběhu různých činností (např. zobrazení chybových hlášek).

Dále frontend obsahuje reference na automaticky přidané balíčky, které nejsou předmětem této práce. Lze z nich ale vyjmout např. „vuex“ a „vuex-persist“ (správa držení dat v prohlížeči), „vue-router“ (správa cest mezi jednotlivými částmi aplikace - směrovač) či „axios“ (komunikace s API).

4.3.2 Komunikace s Laravel API

Komunikace mezi frontendovou aplikací a Laravel API je pro celkovou funkčnost projektu klíčová - bez ní by nedávala existence této části žádný smysl. K těmto účelům je do projektu přidán balíček „axios“ - ten se stará o veškerou komunikaci mezi klientskou a serverovou stranou tzn. data dokáže odesílat i přijímat. Komunikace probíhá tak, že nějaký frontendový prvek (obvykle pohled) volá metody zastřešené tímto balíčkem a na základě odpovědi ze serveru dostává data.

Všechny tyto metody a soubory spojené s komunikací se nachází ve složce *apis*. Zde je klíčový soubor *Api.js*, ve kterém nejprve inicializujeme objekt, který celou komunikaci zastřešuje (ten pochází ze zmíněného balíčku) - tomu je vhodné přiřadit výchozí cestu, ze které bude později tvořit absolutní cesty, na které bude posílat specifické požadavky

(v případě této implementace název domény + řetězec */api/*). Poté mu deklarujeme, co má provést v případě příchozí odpovědi ze serveru bez chyby a s chybou. V případě úspěšné akce (tedy navrácení dat bez chyby - obvykle s kódem 200) jsou data bez jakýchkoli úprav předána komponentě, která si je žádala. V opačném případě (tedy při chybě) je kromě vrácení obsahu příchozí zprávy také do konzole vypsána příslušná chyba. Vrácení příchozích dat s chybou tímto způsobem je určeno pro vytvoření chybového oznámení pro uživatele - aby uživatel věděl, proč daná operace selhala. Zároveň to slouží např. samotným pohledům - na základě chybové hlášky nebo chybového stavového kódu přizpůsobí své chování. Výpis do konzole sloužil primárně k vývoji aplikace - ničemu ale nevadil, proto byl v aplikaci ponechán.

Speciálním případem je situace, kdy se vyskytne chyba se stavovým kódem 401 nebo 419 (uživatel pravděpodobně chce přistoupit k obsahu, ke kterému nemá v současnou chvíli oprávnění např. protože mu vypršela relace přihlášení). V takovém případě se aplikace pokusí uživateli zneplatnit přihlášení a odstranit o něm záznam z prohlížeče a uživatele automaticky přesměrovat na pohled pro přihlášení.

Další soubory ve zmíněném adresáři jsou *Form.js* a *User.js*. Ty obsahují konkrétní funkce s referencí na objekt deklarovaný v souboru *Api.js*, přednastavenou metodu komunikace (ta musí korelovat s metodami na API viz sekce 4.2.2) a relativní cestu, na kterou budou požadavek směrovat (ta je poté spojena s předem definovanou výchozí cestou). Zmíněné cesty samozřejmě musí odkazovat na existující cesty definované serverem (viz tabulky 4.1 a 4.2). Funkci je samozřejmě možné předat parametr (obvykle set dat), který je poté předán na server. Jak už z názvů souborů vyplývá, *Form.js* zaštiťuje komunikaci týkající se manipulace s formuláři nebo s odpověďmi na formulář a *User.js* zajišťuje komunikaci v rámci veškeré administrativy nad uživateli.

4.3.3 Komponenty

Komponenty jsou v podstatě „díly k složení celé aplikace“. Díky nim nemusíme např. stránku pro vytvoření formuláře psát do jednoho nepřehledného a obrovského souboru, ale můžeme tento kód rozdělit. Výhodou také je, že když budeme tento komponentový styl zápisu kódu dodržovat, lze jeho části v jiných místech opětovně využít - předcházíme tím redundantnímu kódu, který zbytečně znepřehledňuje celý projekt. Všechny zmíněné komponenty jsou zapsány v souborech s koncovkou *.vue* a disponují stejným jménem jako samotný soubor. Struktura kódu je vyobrazena na obrázku 3.1.

Níže jsou popsány jednotlivé komponenty, které byly v projektu implementovány.

4.3.3.1 Základní komponenty

Mezi základní komponenty jsou řazeny především ty, jejichž použití se opakuje v aplikaci nejčastěji - tedy *Header* a *Loading*.

Komponenta *Header* slouží k vykreslení horního navigačního panelu s nabídkou dalších funkcionalit, která na základě přihlášení uživatele mění svůj obsah. Pokud není nikdo přihlášen, nabídka nabízí odkazy na přihlášení či registraci. Pokud se uživatel přihlásí, může se odsud dostat na stránku svého profilu, na vytvoření nového formuláře nebo se odhlásit ze svého účtu.

Komponenta *Loading* obsahuje jednoduchou animaci, která symbolizuje průběh načítání. Je využita v částech komponent a pohledů, kde je očekáván např. příjem nějakých dat z backendu, na který se musí čekat. Uživateli má vyobrazovat, že v aplikaci probíhá nějaká činnost.

Dále zde lze zmínit i komponentu *App* z kořenového adresáře frontendu, která již byla popsána v sekci 4.3.1. Lze dodat, že se v ní kromě předaných pohledů nachází i reference na komponentu *Header* - ta musí být z podstaty věci vykreslena na všech cestách.

4.3.3.2 Komponenty elementů formuláře

Tyto komponenty představují jednotlivé otázky a k ním příslušný vstup definovaný typem (pokud se nejedná o element nové stránky). Existují zde interaktivní komponenty elementů (použity např. ve vyobrazení celého formuláře k vyplnění) nebo komponenty statické (použity např. ve vyobrazení jednotlivých otázek při vytváření formuláře).

Interaktivní komponenty elementů formuláře nalezneme v podsložce *Elements* - zde se nachází tyto komponenty: *BooleanInput*, *DateInput*, *NumberInput*, *SelectInput* a *TextInput* (vycházející z typů ze sekce 4.1.1.4). Všechny tyto komponenty jsou si navzájem velmi podobné, jelikož využívají komponenty z balíčku „VueFormulate“: *FormulateInput* pro vykreslení samotné otázky a vytvoření vstupního pole podle datového typu pro odpověď (např. pro text je vygenerována oblast, do které lze vkládat textový řetězec) a *FormulateErrors* pro výpis validační chyby (např. pokud má mít vkládaný text délku maximálně deset znaku a do oblasti vstupu je vložen text delší, tak se objeví chybové hlášení pod otázkou). V logické části jsou si taktéž všechny komponenty podobné - při jejich inicializaci se na základě přijmutích dat nastaví validační pravidla pro vstup a ID otázky, ke které samotná komponenta patří.

Tyto interaktivní komponenty jsou poté zabaleny do dalších komponent. Komponenta *FormElement* na základě příchozích dat (tj. typ otázky) vrací rodičovské komponentě *FormElementsComponent* příslušný element formuláře rozebraný v předchozím odstavci. Zmíněná rodičovská komponenta poté figuruje především při vykreslování celého formuláře - má za úkol vykreslovat veškeré otázky formuláře s funkcí stránkování. Při inicializaci této komponenty se otázky nejprve rozdělí do příslušných polí, které představují stránky,

na základě výskytu elementu nové stránky (tzn. pokud se při procházení všech elementů formuláře najde element nové stránky, tak je vytvořeno nové pole, do kterého se přesouvají další otázky kolekce). Po vzniku těchto polí představujících stránky dojde k jejich vykreslení pomocí zmíněné komponenty *FormElement*. Stránkování je zde zajištěno pomocí skrývání a odkrývání jednotlivých stránek pomocí CSS stylů ovládaných příslušnými tlačítky - takto to je řešeno kvůli udržení dat při přesunu na další stránku (nutné opatření kvůli zmíněnému balíčku k tvoření formulářových prvků).

V rámci statických komponent zde existuje komponenta *FormElementControl*. Ta slouží k vykreslování reprezentace otázek nebo elementu nové stránky v pohledech vytváření nebo úpravě formuláře. Jedná se o komponentu, která na základě předaného typu elementu (tj. otázka s příslušným typem nebo nová stránka) vrací příslušnou reprezentaci elementu (např. v případě textového vstupu vyobrazení textového pole, do kterého nelze vkládat odpověď, a samotné otázky). Při kliknutí na element otázky se objeví modální okno pro úpravu vybrané otázky - toto modální okno je popsáno v sekci 4.3.3.3.1. U elementu nové stránky toto provést nelze, proto je zde připraven jen prokliknutelný text „Remove“, který vysílá specifickou událost do rodičovské komponenty, která ho používá (slouží k odebrání elementu).

4.3.3.3 Modální okna

Zmíněné komponenty představují předpis vykreslení pro modální okna spravovaná balíčkem „vue-js-modal“. Na jejich funkčnosti závisí několik pohledů. Tyto komponenty se nachází v podsložce *Modals* - zde existují ještě další tři podsložky pojmenované podle oblasti, ve které jsou používány. Ty poté obsahují příslušné soubory těchto komponent. Jedná se tedy o tyto složky: *CreateForm*, *FormResults*, *FormReview*.

4.3.3.3.1 Manipulace s elementy formuláře

První adresář *CreateForm* obsahuje příslušné komponenty modálních oken a podkomponenty využívané především v pohledu vytváření formuláře. Později byly tyto komponenty využity i v pohledu pro úpravu formuláře - název tohoto adresáře se ale neměnil. Adresář obsahuje komponenty modálních oken *ItemModal* a *SelectChoiceModal*.

Komponenta *ItemModal* slouží k samotné tvorbě nového elementu nebo úpravě stávajícího elementu otázky ve formuláři - to je možné pomocí manipulace s dočasným úložištěm dat *createFormStore* (viz sekce 4.3.5.2), se kterým se pracuje např. v rodičovském pohledu pro vytváření formuláře. K manipulaci s daty využívá příslušné metody inicializovaného úložiště.

Celé modální okno je zobrazeno na základě vyžádání jiné komponenty (např. pohledu pro vytvoření formuláře) a je tvořeno zejména komponentou *FormulateForm* (z balíčku „VueFormulate“), ve které jsou vnořeny všechny ovládací prvky (tvořené komponentami

FormulateInput a *FormulateErrors*) - ty shromažďují veškeré informace o elementu otázky - tedy typ odpovědi na otázku (např. text nebo číslo), samotnou otázku, zda je otázka povinná a další validační pravidla, která jsou vázána na typ otázky (tato pravidla vychází ze sekce 4.1.1.4). Ovládací prvky dalších validačních pravidel jsou další samostatné komponenty připojované do modálního okna - každý typ otázky má pro tyto účely svou podkomponentu (jedná se o *DateItem*, *NumberItem*, *SelectItem* a *TextItem* ve stejné složce).

Zmíněné prvky mohou být už předvyplněny v situaci, kdy je upravován existující element (komponenta může přijímat vstupní hodnoty). Data jsou poté na základě stisknutí příslušného tlačítka (např. s účelem vytvoření nového elementu s textem „Create“) z modálního okna předána příslušné metodě úložiště, která si data sama zpracuje.

Známou komplikací je samozřejmě zpracování otázek s možnostmi odpovědi. K těmto účelům je v komponentě *SelectItem* vnořena další podkomponenta *SelectChoicesComponent*. Tato komponenta poskytuje administrativu nad možnostmi odpovědi - k tomu používá vlastní dočasné úložiště *createFormChoicesStore*, které má stejné metody jako již zmíněné úložiště *createFormStore* (data z celé této komponenty samozřejmě podléhají i manipulaci s daty rodičovské komponentě *ItemModal*). V komponentě jsou kromě tlačítka pro přidání možnosti odpovědi vykreslovány samotné možnosti odpovědi. Ty lze samozřejmě modifikovat - při kliknutí na ně se zobrazí další modální okno *SelectChoiceModal*, ve kterém jim lze upravit příslušné hodnoty (toto okno je používáno i k přidávání těchto možností). Toto okno poté data předává zmíněnému poskytnutému úložišti. Zajímavou funkcí, kterou lze u *SelectChoicesComponent* zmínit, je hlídání hodnoty *hasHiddenLabel* pocházející z rodičovské komponenty (ta značí, zda možnosti odpovědi mají skrytý popisek - tj. škála) - pokud je tato hodnota změněna z *true* na *false*, tak jsou automaticky tyto skryté popisky odstraněny. V opačném případě komponenta automaticky očísluje možnosti odpovědi čísly 1,2,3,...

4.3.3.3.2 Výsledky formuláře

Dalším adresářem je *FormResults* obsahující pouze jedno modální okno s názvem *FormResultsPublicationModal*. To je využito zejména v pohledu pro zobrazení výsledků formuláře. Toto okno je znovu tvořeno především prvky z balíčku „VueFormulate“.

Před inicializací přijímá všechny otázky formuláře, hodnotu vyjadřující zda je formulář už zveřejněn a identifikátor formuláře. Poté, na základě přijatých dat, vypisuje jeho současný stav a možnosti, co lze s formulářem provést. Klíčová je hodnota vyjadřující zda jsou už výsledky veřejné - pokud se rovná *false*, je zobrazeno jen prázdné zaškrtávací políčko vyjadřující, že výsledky zveřejněny nejsou. Pokud se hodnota rovná *true*, jsou vypsány i všechny otázky, u kterých lze označit, které z nich budou veřejně viditelné. Zmíněnou hodnotu můžeme pomocí zaškrtávacího políčka samozřejmě měnit - s tím se i dynamicky změní rozložení okna. Poté, co navolíme příslušné parametry a klikneme na tlačítko pro uložení nastavení, je na server poslán požadavek s příslušnými údaji. Ten je zpracován

metodou *publishResults*, která je popsána v sekci 4.2.9.1.5. Po úspěšném provedení činnosti se okno zavírá a celá stránka je přenačtena.

4.3.3.3.3 Manipulace s celým formulářem

Posledním adresářem je *FormReview*. Ten obsahuje dvě modální okna použitá v pohledu pro zobrazení souhrnu informací o formuláři. Jedná se o komponenty *FormAccessibilityModal* a *FormDuplicationModal*.

Komponenta *FormAccessibilityModal* slouží k úpravě přístupnosti formuláře. Je také tvořena především komponentami z balíčku „VueFormulate“ a ke svému správnému chodu potřebuje při vytvoření nejprve dostat hodnotu vyjadřující současnou přístupnost (zda je formulář veřejný nebo privátní) a identifikátor formuláře - pokud se jedná o privátní formulář, musí být předány i emaily, na které byla už zaslána pozvánka. Na základě těchto dat je vykresleno příslušné rozložení modálního okna - toto rozložení lze měnit na základě změny hodnoty vyjadřující přístupnost. Pokud je zmíněná hodnota *false*, zobrazí se jen související prázdné zaškrtávací políčko. Pokud se ale rovná *true*, zobrazí se zde i další dva ovládací prvky. V prvním lze vidět emaily, na které již byla zaslána pozvánka - ty lze pro zneplatnění pozvánky označit. V druhém můžeme zase přidat emaily, na které bude poslána nová pozvánka k vyplnění formuláře. Nakonec (po kliknutí na tlačítko vyjadřující provedení změny) se příslušná data pošlou na server. Ta jsou zpracována metodou *updateAccess*, která je popsána v sekci 4.2.9.1.4. Modální okno se poté zavře a stránka je přenačtena.

Druhá komponenta *FormDuplicationModal* nám umožňuje duplikovat vybraný formulář. I toto modální okno je tvořeno komponentami z balíčku „VueFormulate“ a při inicializaci očekává základní data o formuláři - tedy jeho název, popisek, datum zveřejnění, datum ukončení zveřejnění a zda se jedná o formulář veřejný či privátní (u privátního také emaily, které dostali pozvánku). Tato data jsou dočasně uložena v úložišti *duplicateFormStore*, které je popsáno v sekci 4.3.5.2. Tyto údaje lze libovolně měnit - pokud jsou zadány neplatné hodnoty (např. datum zveřejnění je dále v čase než datum ukončení zveřejnění), tak nás okno bude informovat příslušným upozorněním. Poté, co klikneme na příslušné tlačítko pro duplikaci formuláře, je odeslán na server požadavek, který je obsloužen metodou *duplicateWithAuth*, jež je popsána v sekci 4.2.9.1.3. Po tomto úkonu je modální okno zavřeno a stránka přenačtena.

4.3.3.4 Komponenty pro zobrazení výsledků

Dalšími komponentami, které jsou zde popsány, jsou komponenty pro grafické znázornění dat v pohledu pro zobrazení výsledků formuláře (pro vyobrazení dat ve formě grafů zde byl použit balíček „vue-chartjs“). Nachází se v adresáři *ResultComponents*, který obsahuje komponenty *DetailedResultsTable*, *ResultComponent* a *ResultRender*. Také se zde nachází podsložka *Views*, ve které lze najít předpřipravené podkomponenty pro konkrétní styl vykreslení dat - *Bar* pro vykreslování sloupcových grafů, *List* pro vykreslování jednoduché tabulky stylem klíč/hodnota (resp. ID odpovědi/odpověď) a *Pie* pro vykreslování koláčových grafů. Tyto podkomponenty jsou poté předávány dál pomocí zmíněné komponenty *ResultRender*, která je vrací na základě předaného typu a dat (tzn. výsledky k otázkám s možnostmi odpovědi nebo k otázkám Ano/Ne jsou zobrazovány jako grafy - lze si mezi zmíněnými styly grafu po načtení vybrat, ostatní jako zmíněná tabulka).

Popsaná komponenta *ResultRender* je poté využita ve zmíněné komponentě *ResultComponents*. Pro správnou funkcionalitu komponenta *ResultComponents* musí pro získání příslušného vyobrazení dat samozřejmě předaná data zpracovat do validního formátu a poté je poskytnout komponentě *ResultRender*. Nakonec zmíněná rodičovská komponenta *ResultComponents* přidá k vyobrazení dat (např. ve formě grafu) samotnou otázku a celý obsah je předán do pohledu pro zobrazení výsledků.

Nelze opomenout komponentu *DetailedResultsTable*, která slouží jako alternativní zobrazení všech výsledků formuláře. (ve výchozím nastavení se zobrazují data ve zmíněných grafech nebo v tabulce pro každou otázku). V podstatě data zobrazuje v podobném formátu, jako jsou vysázeny při exportu dat (tzn. v tabulce viz sekce 4.2.9.2.5). Oproti exportu se vyobrazení dat liší v tom, že v buňkách, kde není žádná hodnota (*null*), je umístěn znak pomlčky („-“).

4.3.3.5 Komponenta karty formuláře

Jednoduchá komponenta, kterou nebylo možné zařadit mezi ostatní výše, s názvem *FormCard* slouží jako předpis pro vykreslování karty formuláře. Ta je použita v pohledu úvodní stránky v případě, že je uživatel přihlášen a že už nějaké formuláře vlastní. Po kliknutí na kartu se přihlášený uživatel dostane na pohled se zobrazením souhrnu informací o požadovaném formuláři. Sama o sobě karta zobrazuje název a popisek formuláře či data spuštění a ukončení zveřejnění formuláře.

4.3.4 Směrovač

K tomu, abychom mohli přecházet mezi částmi aplikace, je nutné, aby v aplikaci existoval nějaký komplexní systém, který by nad tímto problémem měl kontrolu. To zde řeší tzv. směrovač (běžně známý pod anglickým názvem Router), který je importován z balíčku „vue-router“. Tento směrovač má za úkol na základě definovaných možných cest v aplikaci vracet příslušný pohled uživateli.

Směrovač implementovaný v tomto projektu je umístěn ve složce *router* v souboru *index.js*. V tomto souboru jsou nejprve vydefinovány všechny relativní cesty, ke kterým je přiřazen příslušný pohled, který je při vstupu na nějakou z cest vrácen uživateli - některé mohou disponovat i metadaty (např. zda cesta vyžaduje pro vykreslení přihlášení uživatele). Poté je inicializován objekt tohoto směrovače, kterému jsou předány především data jako výchozí cesta (v této implementaci název domény), ke které jsou relativní cesty ve výsledku připojeny, a samotná kolekce vydefinovaných cest.

Nakonec lze určit i specifické chování pro zmíněná metadata u cest, což je zde také využito a popsáno níže:

- Pokud cesta vyžaduje přístup bez přihlášení a uživatel je přihlášen, je přesměrován na domovskou stránku (např. pokus o přístup na přihlašovací stránku). Pokud není přihlášen, je přesměrován na stránku na zvolené cestě.
- Pokud cesta vyžaduje přístup s přihlášením a uživatel je přihlášen, je přesměrován na stránku na zvolené cestě. Pokud není, je přesměrován na pohled pro přihlášení (např. pokus o přístup na pohled zobrazující souhrn informací o formuláři, který je určen jen přihlášenému majiteli formuláře).
- Pokud nemá cesta žádná známá metadata, je uživatel přesměrován na stránku na zvolené cestě.

4.3.5 Držení dat v prohlížeči

Možnost držení dat v prohlížeči je další ze základních atributů pro správnou funkčnost aplikace. Ať už se jedná o soubory cookies, nebo o dočasné úložiště v paměti alokované přímo JavaScriptem - zkrátka je to potřeba. V tomto projektu byly využity oba zmíněné přístupy.

Soubory spojené s držením dat pomocí JavaScriptu (označovány jako Stores) jsou umístěné ve složce *store* - jedná se o soubor *index.js* a soubor *MyStoreClass.js*.

4.3.5.1 Přihlášení uživatele

V rámci přihlášení uživatele lze držení dat rozdělit na dvě větve: pomocí cookies a pomocí držení dat v JavaScriptu za využití balíčku „vuex“.

V cookies jsou drženy šifrované tokeny, které jsou spravované přímo backendovým balíčkem pro zabezpečení single-page aplikací „Laravel Sanctum“ a slouží k ověřování přihlášení (jsou autonomně posílány společně s každým požadavkem na server - ten na základě jejich platnosti provede požadovaný úkon).

Naproti tomu přihlášení drženo pomocí „vuex“ je zde primárně pro funkčnost metadat u směrovače (viz sekce 4.3.4) - na základě tohoto drženého stavu může přihlášený uživatel přistupovat na stránky, které jsou přístupné pouze s přihlášením. Toto je definováno v souboru *index.js*, kde probíhá inicializace objektu z balíčku „vuex“, který má deklarován příslušné proměnné držící stav (*states*), metody pro vrácení hodnot proměnných držících stav (*getters*), metody pro cílenou změnu hodnot (*mutations*) a další procedury ve formě metod (*actions* - např. procedura pro přihlášení uživatele). K tomu, aby data nebyla smazána při přenačtení stránky, musí být přidán zmíněnému objektu plugin *VuexPersistence* z balíčku „vuex-persist“, který toto chování zprostředkovává.

4.3.5.2 Data pro manipulaci s formuláři

K tomu, abychom mohli vytvářet, upravovat nebo duplikovat formuláře, bylo nutné jejich data dočasně (do odeslání požadavku na server) držet v prohlížeči. K těmto účelům vznikla tři dočasná „skladiště“ dat - *createFormStore*, *createFormChoicesStore* a *duplicateFormStore*.

První dva zmíněné objekty (tedy *createFormStore* a *createFormChoicesStore*) jsou určené pro vytváření a úpravu formuláře a jsou ve své podstatě naprosto identické, jelikož se jedná o instance pro projekt vytvořené třídy *MyStore*, která se nachází v souboru *MyStoreClass.js*.

Třída *MyStore* obsahuje proměnnou pro ukládání dat, což je objekt, ve kterém je vytvořeno pole pro ukládání hodnot kolekce dat. Kromě této proměnné třída disponuje různými metodami: *getItems* vrací všechny prvky kolekce, *addItem* ukládá předaná data z parametru do kolekce, *refreshItemsOrder* obnovuje hodnotu pořadí každému prvku v kolekci (tzv. postupně očísluje prvky), *sortItemsByOrder* řadí prvky v kolekci podle hodnoty pořadí, *changeItem* dokáže nahradit existující prvek v kolekci na základě shody ID, *deleteItem* smaže prvek z kolekce na základě ID, *clearStore* vymaže všechny prvky z kolekce a *setItems* přepíše celou kolekci předanou kolekcí.

Třetí objekt (*duplicateFormStore*) slouží k držení dat pro duplikaci formuláře. Oproti zmíněným instancím se jedná o „skladiště“, které je připraveno pro konkrétní data související s duplikací. V rámci proměnných, ve kterých dočasně drží data, jsou zde vlastně deklarovány pole pro příslušné ovládací prvky modálního okna pro duplikaci (viz sekce

4.3.3.3.3). Dále obsahuje standardní metody pro smazání, přepsání a vrácení uložených dat (*clearData*, *setData* a *getData*), ale také metodu pro validaci těchto dat (*validateData*, která na vrací případnou chybovou hlášku pro zobrazení v modálním okně) či metodu pro zjištění, zda je úložiště prázdné neboli ve výchozím stavu (*isStoreEmpty*).

Lze namítat, že pro ukládání těchto dočasných dat je možné využít lokální úložiště VueJS komponenty *data* - zde to ale možné nebylo, jelikož bylo potřeba předávat data mezi více komponentami. Zároveň, díky tomuto přístupu, bylo možné s daty manipulovat v určitých situacích jednodušeji za pomocí předpřipravených metod.

4.3.6 Pohledy

Pohledy (anglicky Views) jsou v rámci struktury kódu stejné jako komponenty, které jsou popsány v sekci 4.3.3. Nacházejí se ve složce *views* a ve své podstatě i oni jsou komponentami - rozdíl je jen ve využití, jelikož pohledy jsou skládány ze zmíněných komponent. Poté, co je pohled složen z těchto komponent, je vykreslen jako plnohodnotná stránka. U samostatných komponent nemá význam, aby byly vykreslovány samostatně - potřebují data ke zpracování, která jim předává právě příslušný pohled.

Jak již bylo řečeno, pohled musí podřadným komponentám, které implementuje, dodat data ke správné funkčnosti. Tato data získává pomocí komunikace s API na základě volané metody a předaných dat (viz sekce 4.3.2). Poté, co získá odpověď ze serveru, tak před předáním dat komponentám data obvykle zpracuje do takového formátu, který komponenty očekávají. Zároveň řeší i situace, kdy dojde k chybě při přijmu dat - pohled musí dát samozřejmě uživateli vědět příslušným způsobem, co se v aplikaci stalo resp. proč např. nedostal data, která požadoval. Zároveň pohled obsahuje příslušné ovládací prvky, které nám umožňuji se pohybovat mezi pohledy - toto je zajištěno směrovačem, který je popsán v sekci 4.3.4.

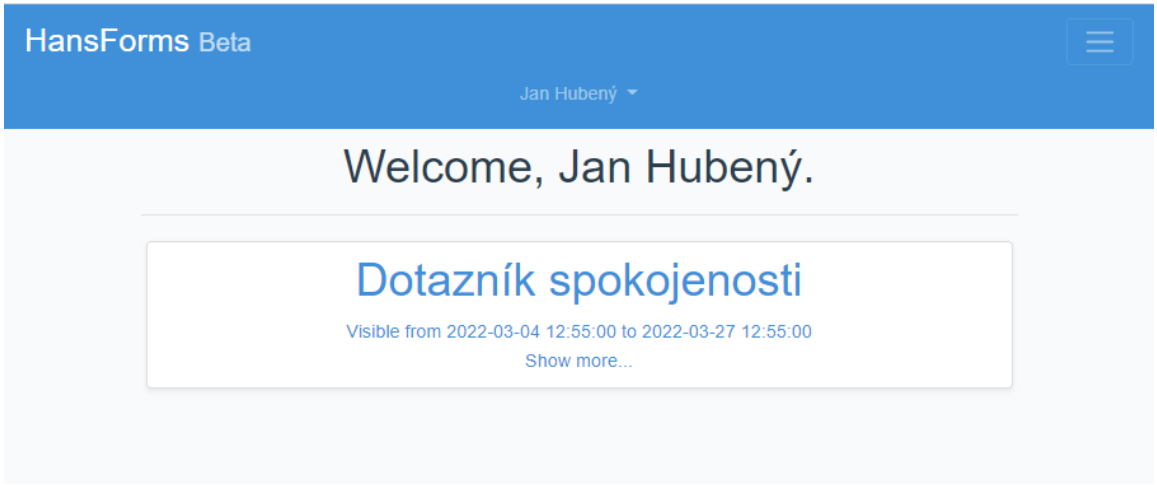
V této části jsou obecně popsány všechny komponenty a jejich účel. V částech, kde jsou použity vstupní ovládací prvky (např. pro zadání textové hodnoty), jsou data před posláním na sever také náležitě validována (např. validací, kterou poskytují komponenty z balíčku „VueFormulate“). Pohledy jsou responzivní - obrázky v této práci byly kvůli rozložení tohoto dokumentu snímány v režimu pro tablet.

4.3.6.1 Domovská stránka

První pohled, který je zobrazen uživateli především při prvním použití aplikace nebo ihned po přihlášení, je právě pohled domovské stránky (viz obrázek 4.13). Tento pohled je rozdělen na dva způsoby vykreslení - bez přihlášení a s přihlášením.

Pokud uživatel přihlášený není, je zobrazena úvodní stránka, která obsahuje kromě úvodního textu jen odkazy na přihlášení a na registraci - jedná se jen o statickou stránku.

Pokud ale uživatel přihlášený je, jsou dvě možnosti, jak je pohled vykreslen. Pokud ještě nevlastní žádný formulář, je mu zde poskytnut odkaz na pohled, kde formulář může vytvořit. Pokud už ale nějaké formuláře vlastní, jsou mu na úvodní stránce všechny vráceny ve formě katalogu karet za použití komponenty *FormCard* ze sekce 4.3.3.5. Data těchto formulářů jsou získána ze serveru po ověření přihlášení pomocí metody *index* ze sekce 4.2.9.1.7.



Obrázek 4.13: Pohled domovské stránky (přihlášený uživatel vlastnící formulář)

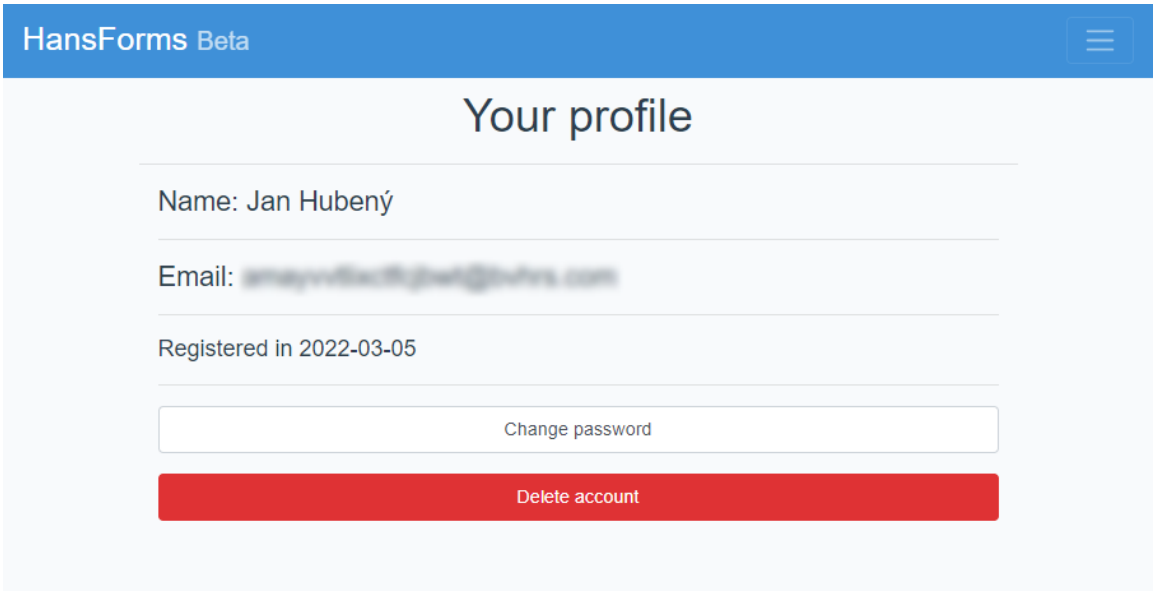
4.3.6.2 Správa uživatele

Co se týče administrativy nad uživatelem, existuje zde šest pohledů, které slouží k různým účelům. Jedná se o pohledy *Login*, *Register*, *Profile*, *ChangePassword*, *ForgotPassword* a *ResetPassword*.

Pohled *Login*, jak už z názvu vyznívá, slouží k přihlášení uživatele. Disponuje vstupními poli pro email a heslo. Poté, co uživatel klikne na tlačítko přihlášení, je na server poslán požadavek se získanými daty ze vstupních polí (na metodu *login* viz sekce 4.2.9.3.2). Pohled pak čeká na odpověď serveru - pokud obsahuje stavový kód 200 a informace o uživateli, je přihlášený uživatel přesměrován na úvodní stránku aplikace. V opačném případě je upozorněn na chybu, která se vyskytla (např. zpráva o chybných údajích). Stránka také disponuje odkazem na stránku pro obnovu hesla. Díky metadatům na tento pohled nelze po přihlášení uživatele přistoupit.

Oproti předchozímu pohledu pohled *Register* slouží k registraci uživatele. Obsahuje vstupní pole pro uživatelské jméno, email a heslo společně s jeho ověřením. Po kliknutí na tlačítko pro registraci je poslán požadavek na server - konkrétně metodě *register* (viz sekce 4.2.9.3.1). Pokud je registrace úspěšná, je uživatel přesměrován na přihlašovací stránku. Pokud není, je mu vráceno oznámení s chybou. Stejně jako v pohledu *Login* nelze po přihlášení uživatele na tento pohled přistoupit.

Dalším pohledem je *Profile* (viz obrázek 4.14), který obsahuje náležité údaje o přihlášeném uživateli (pokud uživatel není přihlášen, nelze k němu přistoupit). Data o uživateli získává ze serveru při načtení stránky pomocí metody *show* (viz sekce 4.2.9.3.4). Na tomto pohledu lze také změnit heslo (obsahuje tlačítko pro přesměrování na pohled pro změnu hesla) nebo celý účet smazat (při kliknutí na příslušné tlačítko po vyplnění aktuálního hesla pro ověření se odešle požadavek na server na metodu *destroy* viz sekce 4.2.9.3.5).



Obrázek 4.14: Pohled profilu přihlášeného uživatele

Již zmíněný pohled pro změnu hesla se jmenuje *ChangePassword*. Na ten lze přistoupit pouze s přihlášením. Obsahuje pole pro staré heslo, a pole pro heslo nové s ověřením. Po kliknutí na příslušné tlačítko pro změnu hesla se odešle požadavek na server na metodu *changePassword* ze sekce 4.2.9.3.6. Následně, na základě serverové odezvy, při úspěchu akce uživatele přesměruje na pohled *Profile*. V opačném případě je uživatel vyzván k úpravě vyplněných údajů.

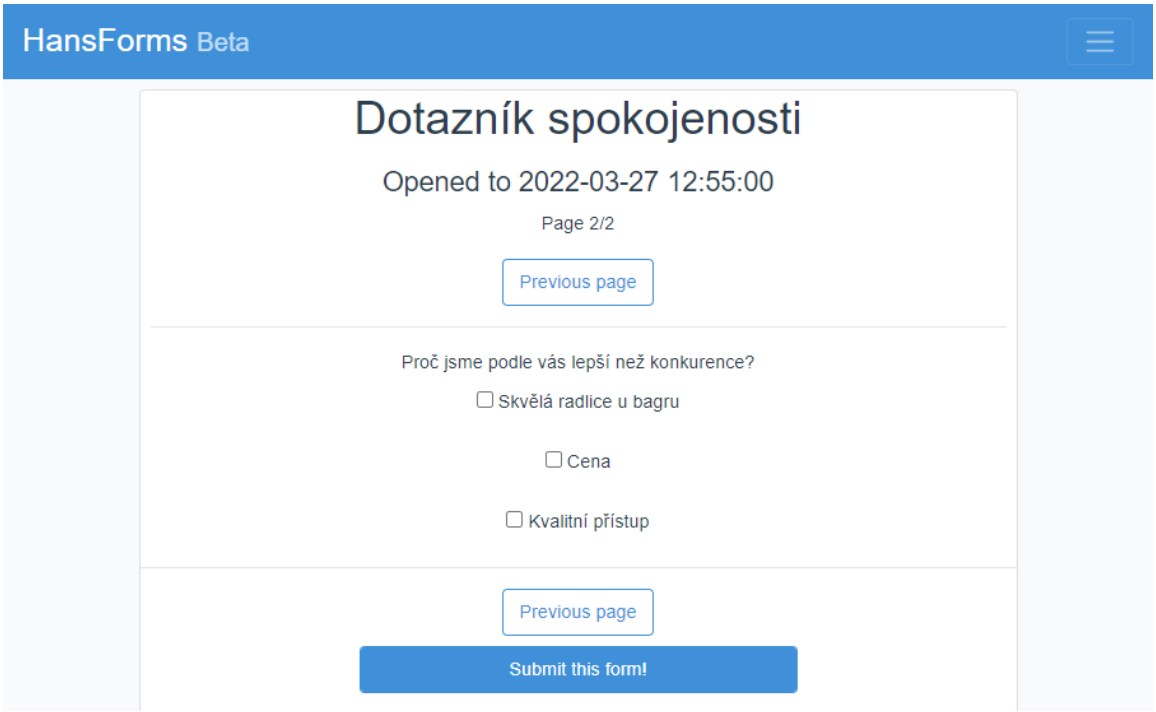
Předposlední pohled v této sekci je *ForgotPassword*. Jedná se o stánku, na kterou lze přistoupit z pohledu *Login*. Jak už z názvu vyplývá, na tomto pohledu lze v případě zapomenutého hesla požádat o resetování - obsahuje vstupní pole pro vyplnění emailu, ke kterému je vázán účet se zapomenutým heslem, a tlačítko pro odeslání požadavku na server. Žádost o obnovení se zasílá na metodu *forgotPassword*, která je popsána v sekci 4.2.9.3.7. Pokud vše proběhne bez problému, je uživatel přesměrován na domovskou stránku. Pokud se vyskytne chyba, je uživatel adekvátně upozorněn.

Posledním pohledem je *ResetPassword*, který slouží k resetování hesla tím, že se zvolí heslo nové. K pohledu se přistupuje především prostřednictvím odkazu, který uživatel dostane na email (nelze k pohledu přistoupit, když je uživatel přihlášen). V pohledu je zobrazen email, ke kterému se vztahuje obnovení přístupu, kolonky pro vyplnění nového hesla s potvrzením a příslušné tlačítko k potvrzení úkonu - tedy k odeslání požadavku na server. Po zpracování požadavku metodou *resetPassword* (viz sekce 4.2.9.3.7) je vrácen

příslušný stavový kód, na základě kterého je buď uživatel přesměrován na pohled pro přihlášení (tj. v případě úspěchu - kód 200), nebo upozorněn na chybu (tj. v případě neúspěchu).

4.3.6.3 Formulář

Pohled *Form* slouží k zobrazení formuláře na základě jeho identifikátoru a zároveň nám umožňuje na zmíněný formulář odpovědět (viz obrázek 4.15). Pohled je přizpůsoben na zobrazení jak veřejného formuláře, tak i privátního (to je rozlišeno na základě tvaru URL adresy). K vykreslování všech otázek využívá komponentu *FormElementsComponent* ze sekce 4.3.3.2.



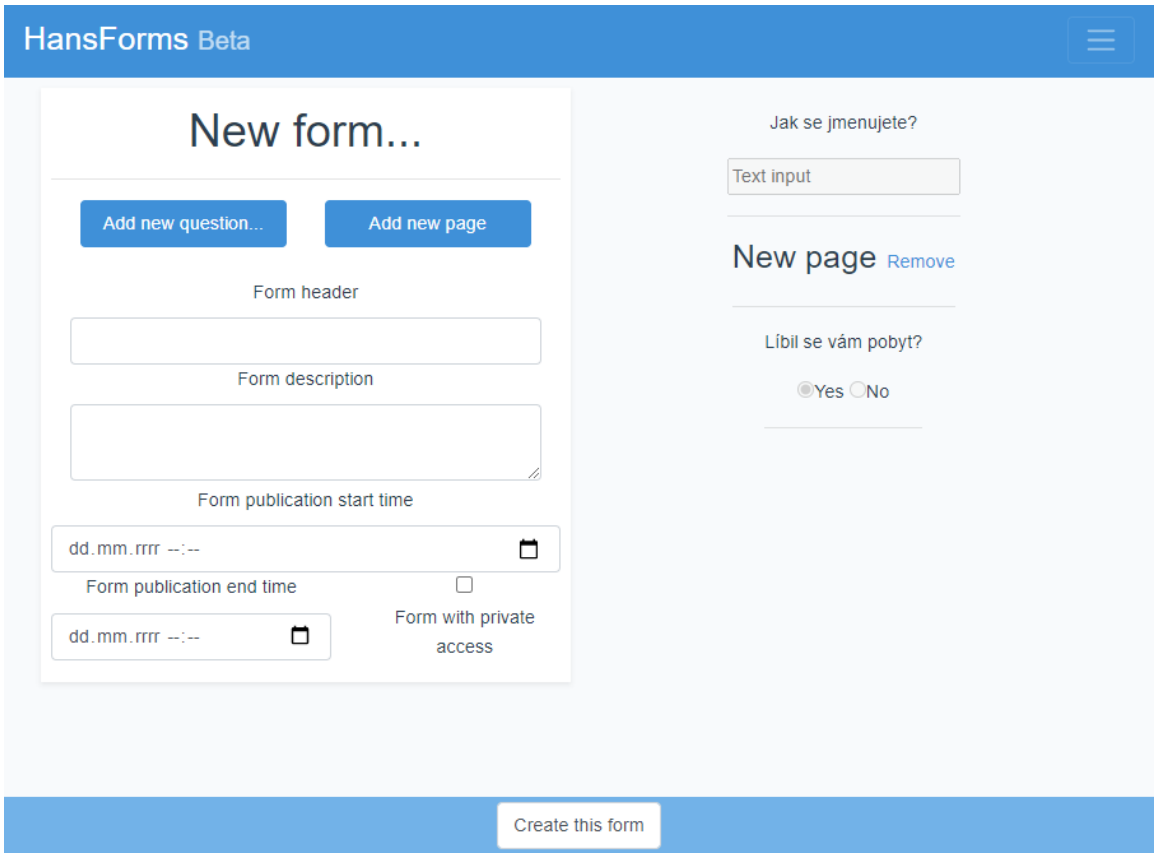
Obrázek 4.15: Pohled formuláře

Celý životní cyklus tohoto pohledu začíná tím, že pohled požádá server o data formuláře na základě předaného identifikátoru (u veřejného formuláře) nebo tokenu (u privátního formuláře) v URL adrese (u veřejného formuláře pomocí metody *show* ze sekce 4.2.9.1.9, u privátního pomocí metody *privateShow* ze sekce 4.2.9.1.10). Pokud požadovaný formulář existuje, jsou příslušné otázky seřazeny a vykresleny (pokud ne, je oznámena chyba).

Poté, co uživatel vyplní příslušná data a klikne na tlačítko pro uložení odpovědi, je odeslán další požadavek na server (pokud se jednalo o veřejný formulář, tak směrující na metodu *store* ze sekce 4.2.9.2.1, pokud se jednalo o privátní formulář, tak na metodu *privateStore* ze sekce 4.2.9.2.2). Na základě odpovědi serveru je respondent buď obeznámen s úspěšným vyplněním formuláře a přesměrován na domovskou stránku, nebo je upozorněn na chybu a je vyzván k opravě předávaných dat.

4.3.6.4 Tvorba formuláře

Pohled *CreateForm* slouží k vytvoření nového formuláře přihlášenému uživateli (nepřihlášený uživatel k tomuto pohledu nemá přístup).



Obrázek 4.16: Pohled pro vytvoření nového formuláře

Z grafické stránky je rozdělen na dvě části (viz obrázek 4.16) - první část obsahuje tlačítka pro přidání elementu otázky a elementu nové stránky a vstupní pole pro vyplnění základních údajů o formuláři (tj. název a popisek formuláře, datum zveřejnění, datum ukončení zveřejnění, zda je formulář veřejný nebo privátní a u privátního formuláře emaily, na které se zašle pozvánka), druhá část obsahuje grafický náhled na otázky. Ve spodní části je tlačítko určené k ukončení úprav a odeslání dat vytvořeného formuláře na server (ta jsou zpracována metodou *store* ze sekce 4.2.9.1.1). Pokud tento požadavek projde bez problému, je uživatel přesměrován na domovskou stránku - v opačném případě mu je předána příslušná chybová hláška.

Pro úpravu otázek v tomto návrhovém zobrazení je zřízeno modální okno *ItemModal* (popsáno v sekci 4.3.3.3). To poté s daty manipuluje pomocí dočasného úložiště *createFormStore* a *createFormChoicesStore* (ze sekce 4.3.5.2). Data z tohoto úložiště jsou průběžně předávána do pohledu, který je dynamicky vykresluje. Před odesláním jsou data validována a zpracována tak, aby splňovala podmínky zmíněné serverové metody, která je přijímá.

4.3.6.5 Úprava formuláře

Pohled *UpdateForm* slouží k úpravě existujícího formuláře, který ještě nebyl publikován. K tomuto pohledu má samozřejmě přístup jen přihlášený vlastník existujícího formuláře.

V rámci vizuální i funkční stránky je skoro totožný s pohledem *CreateForm* - grafické rozložení stránky má stejné, používá stejná modální okna i dočasné úložiště. Jediné, v čem se liší, je přijímání dat existujícího formuláře, která jsou příslušně zpracována pro účely úprav formuláře, a způsob uložení úprav ve formuláři. Také zde nelze měnit přístupnost k formuláři (to je možné v pohledu se souhrnem informací o formuláři, který je popsán v sekci 4.3.6.6).

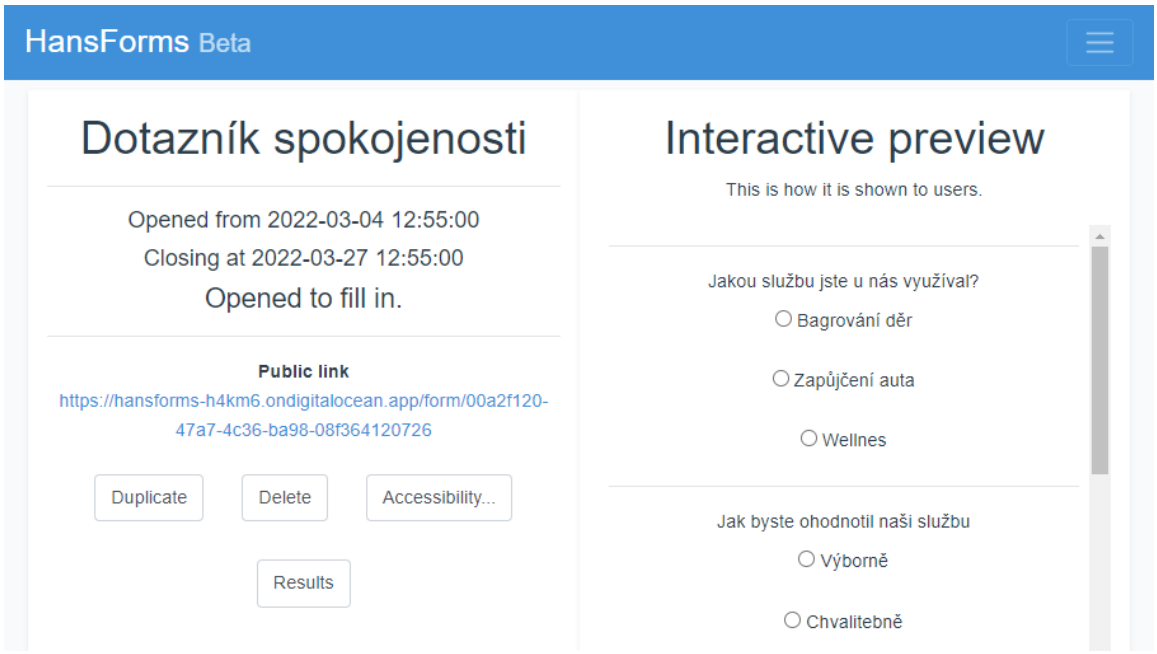
Při inicializaci pohledu je nejprve vyhledán formulář, který má být upraven (pomocí serverové metody *showWithAuth* ze sekce 4.2.9.1.8). Pokud je nalezen, všechna jeho data jsou zpracována do stejného tvaru, který je používán i v pohledu pro vytvoření formuláře (nutné kvůli využívání stejného dočasného úložiště a modálního okna). Poté, co je formulář uživatelem upraven dle jeho představ, je posílán požadavek se změněným formulářem na metodu *update* (viz sekce 4.2.9.1.2). Zpracování chyb je stejné jako ve zmíněném pohledu pro vytvoření formuláře.

4.3.6.6 Souhrn informací o formuláři

Pohled *FormPreview* slouží k zobrazení souhrnu dat o formuláři (viz obrázek 4.17). K tomuto pohledu je přístup možný jen jako přihlášený uživatel (pro vykreslení příslušného souhrnu je samozřejmě nutné, aby požadovaný formulář přihlášenému uživateli patřil). Data o formuláři jsou získána prostřednictvím metody *showWithAuth* ze sekce 4.2.9.1.8 - v případě jakékoliv chyby je uživatel upozorněn konkrétním oznámením. V rámci grafického rozložení je pohled rozdělen na dvě části.

V první části jsou vypsány všechny základní údaje o formuláři (tj. název a popisek, datum zveřejnění a datum ukončení zveřejnění, zda se jedná o formulář veřejný nebo privátní (u veřejného formuláře i odkaz k vyplnění, u privátního zase seznam emailů, na které byla zaslána pozvánka) a status zveřejnění - tzn. zda čeká na zveřejnění, zda je právě otevřený k zodpovězení či zda už zveřejňování bylo ukončeno). Také zde nalezneme ovládací prvky ve formě několika tlačítek, díky kterým můžeme formulář duplikovat (dále pomocí modálního okna *FormDuplicationModal* ze sekce 4.3.3.3 a dočasného úložiště *duplicateFormStore* ze sekce 4.3.5.2), smazat (posláním požadavku na serverovou metodu *destroy* ze sekce 4.2.9.1.6), upravovat (možné jen v případě, že formulář ještě nebyl zveřejněn - zajištěno přesměrováním na pohled *UpdateForm* popsaný v sekci 4.3.6.5), měnit jeho přístupnost (dále pomocí modálního okna *FormAccessibilityModal* ze sekce 4.3.3.3) či prohlížet jeho výsledky (možné jen v případě, že byl formulář už zveřejněn - zajištěno přesměrováním na pohled *FormResults* ze sekce 4.3.6.7).

Druhá část funguje jako interaktivní zobrazení formuláře (jsou zde vykresleny všechny otázky formuláře společně s označením, kde začínají nové stránky) - lze zde např. testovat uživatelem definovaná validační pravidla pro vstupní data. Tato část je tvořena z dat elementů formuláře, která jsou po přijetí předána komponentě *FormElement* (viz sekce 4.3.3.2), která příslušné elementy vykreslí. Nelze pomocí této části přidávat data mezi odpovědi formuláře - data, která jsou v tomto interaktivním zobrazení použita, se nikam neukládají.



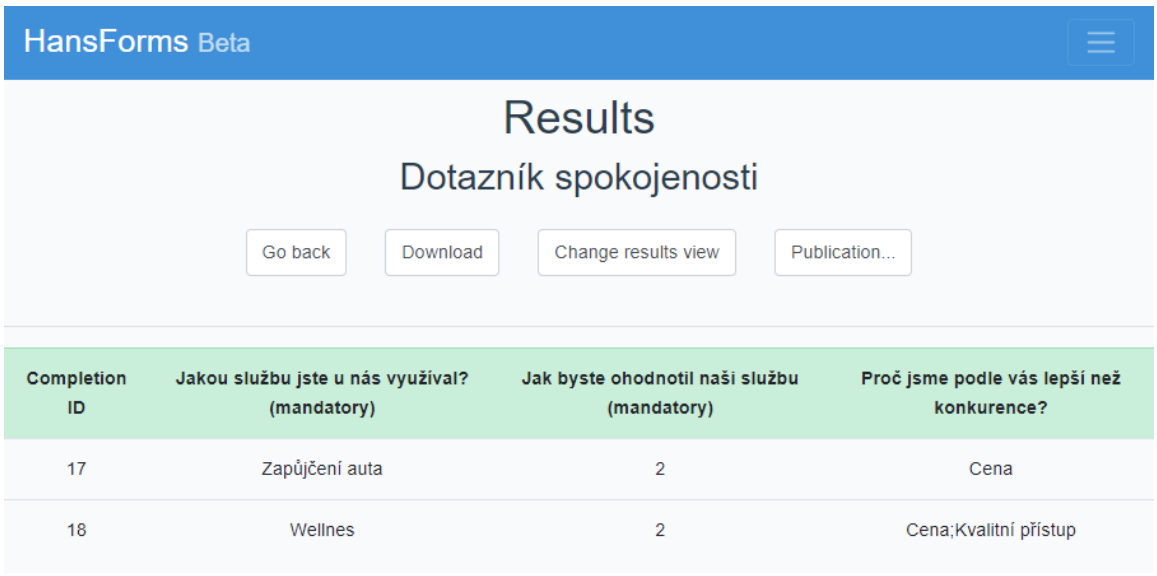
Obrázek 4.17: Pohled souhrnu informací o formuláři

4.3.6.7 Výsledky formuláře

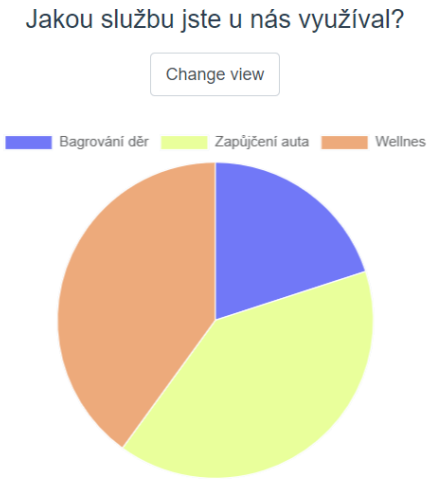
Pohled *FormResults* slouží k zobrazení všech odpovědí na formulář (tedy výsledků), a to i těch pro veřejnost (pokud je k tomu formulář příslušně nastaven). Získání odpovědí pro vlastníka formuláře je zajištěno metodou *index* ze sekce 4.2.9.2.3, pro veřejnost pomocí metody *publicIndex*, která je popsána v sekci 4.2.9.2.4. O tom, jaká data mají být vrácena (zda má být vrácen výběr veřejných odpovědí nebo všechny odpovědi), rozhoduje tvar URL adresy. Pokud mají být vrácena všechna data, musí být vlastník samozřejmě přihlášen. V případě chyby je uživatel příslušně obeznámen se stavem aplikace.

Pokud byla data úspěšně přijata, můžeme na stránce vidět několik ovládacích prvků (tlačítko pro přesměrování na pohled domovské stránky nebo na pohled souhrnu informací o formuláři, v případě výsledků určených pro vlastníka formuláře tlačítko pro stažení vyexportovaných dat ve formátu MS Excel pomocí metody *export* ze sekce 4.2.9.2.5, tlačítko pro změnu vykreslení výsledků - lze vybrat mezi komponentou *ResultComponent* a *DetailedResultsTable* ze sekce 4.3.3.4 - a tlačítko pro nastavení veřejných výsledků pomocí modálního okna *FormResultsPublicationModal* ze sekce 4.3.3.3.2) a samotné výsledky (viz

obrázek 4.18). Způsob vykreslení výsledků lze měnit mezi komplexní tabulkou se všemi odpovědmi *DetailedResultsTable* a způsobem vykreslení výsledků „co otázka, to všechny příslušné odpovědi“ pomocí komponenty *ResultComponent* (zde mohou být data zobrazena ve formě grafů - viz obrázek 4.19 - nebo ve formě tabulky klíč/hodnota - záleží na typu otázky).



Obrázek 4.18: Pohled výsledků formuláře (pomocí komponenty *DetailedResultsTable*)



Obrázek 4.19: Komponenta *ResultComponent* vykreslující koláčový graf

4.3.6.8 Ostatní pohledy

Poslední pohledy, které v aplikaci existují, jsou *NotFound* a *About*. Jedná se o statické pohledy - obsahují jen předepsaný obsah (nedisponují nějakou specifickou funkcionalitou).

Pohled *About* je použit pro stručný popis projektu. Pohled *NotFound* je určen k vykreslení v případě, že směrovač (viz sekce 4.3.4) požadovanou cestu nezná. Obsahuje jen text indikující, že požadovaná stránka nebyla nalezena.

5 Závěr

Podařilo se vytvořit webovou aplikaci pro zpracování ankety se všemi chtěnými funkcemi. V současné době lze formuláře vytvořit, upravovat, duplikovat, mazat, prohlížet získané odpovědi (které lze také příslušným způsobem exportovat např. pro potřeby korelační analýzy) či nastavovat jejich přístupnost. Také je možné, jako respondent, na formulář odpovědět, a to jak přes veřejný odkaz, tak i přes pozvánku z emailu. Součástí projektu se samozřejmě správa uživatelských účtu - kromě registrace uživatele a přihlášení do uživatelského účtu lze např. resetovat zapomenuté nebo změnit známé heslo či smazat účet. V rámci projektu také vznikla rešerše existujících řešení, jež byly inspirací pro tuto implementaci.

Aplikace je v současnou chvíli otestována v základním rozsahu a je považována za funkční. Jsou ale známy záležitosti, které lze zlepšit nebo přidat - např. úprava algoritmu pro modifikaci nezveřejněného formuláře či přidání dalších funkcí do uživatelského rozhraní (např. možnost měnit pořadí otázek pomocí tzv. funkce „Drag and drop“).

6 Literatura

[1] Google Forms: Free Online Form Creator | Google Workspace. *Google* [online]. [cit. 2022-01-15]. Dostupné z: https://www.google.com/forms/about/

[2] DEMAREST, Abigail Abesamis. What are Google Forms? Everything you need to know about Google Workspace's online form builder. *Business Insider* [online]. 2021, 22.1.2021 [cit. 2022-01-15]. Dostupné z: https://www.businessinsider.com/what-is-google-forms

[3] Microsoft Forms | Průzkumy, hlasování a kvízy. *Microsoft Forms* [online]. [cit. 2022-01-30]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/online-surveys-polls-quizzes

[4] Microsoft Forms — Užitečný nástroj, díky němuž se mnoho dozvíte. *Microsoft Forms* [online]. 2021, 3.2.2021 [cit. 2022-01-30]. Dostupné z: https://manica.cz/microsoft-forms-uzitecny-nastroj-diky-nemuz-se-mnoho-dozvite/

[5] Dotazník zdarma: Vytvořit online dotazník. *Survio®* [online]. [cit. 2022-01-30]. Dostupné z: https://www.survio.com/cs/

[6] VERNEROVÁ, Sára. Front end vs. Back end - jaký je mezi nimi rozdíl?. *Apitree* [online]. 2021, 12.9.2021 [cit. 2022-02-04]. Dostupné z: https://www.apitree.cz/blog/front-end-vs-back-end-jaky-je-mezi-nimi-rozdil

[7] JavaScript. *Wikipedia* [online]. 2021, 16.12.2021 [cit. 2022-02-04]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript

[8] KOĎOUSKOVÁ, Barbora. JavaScript pro začátečníky: co to je a jak funguje. *Rascasone* [online]. 2022, 28.01.2022 [cit. 2022-01-23]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-javascript-pro-zacatecniky

[9] Introduction. *Vue.js* [online]. [cit. 2022-01-23]. Dostupné z: https://vuejs.org/v2/guide/#What-is-Vue-js

[10] SFC Syntax Specification. *Vue.js* [online]. [cit. 2022-01-23]. Dostupné z: https://v3.vuejs.org/api/sfc-spec.html#custom-blocks

[11] Úvod do frameworku Vue.js. *Vue manuál* [online]. [cit. 2022-01-23]. Dostupné z: https://vue.baraja.cz/uvod-do-vue

[12] The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web. *GitHub* [online]. [cit. 2022-01-30]. Dostupné z: https://github.com/twbs/bootstrap

[13] Bootstrap (front-end framework). *Wikipedia* [online]. 2022, 2022 [cit. 2022-01-30]. Dostupné z: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

[14] About · Bootstrap v5.1. *Bootstrap* [online]. [cit. 2022-01-30]. Dostupné z: https://getbootstrap.com/docs/5.1/about/overview/

[15] About npm. *Npm Docs* [online]. [cit. 2022-03-03]. Dostupné z: https://docs.npmjs.com/about-npm

[16] ŠTRÁFELDA, Jan. Co je backend. *Jan Štráfelda* [online]. [cit. 2022-01-30]. Dostupné z: https://www.strafelda.cz/backend

[17] Backend. *Shoptet.cz* [online]. [cit. 2022-01-30]. Dostupné z: https://www.shoptet.cz/slovnik-pojmu/backend/

[18] PHP: What is PHP?. *PHP.net* [online]. [cit. 2022-02-22]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php

[19] PHP: What can PHP do?. *PHP.net* [online]. [cit. 2022-02-22]. Dostupné z: https://www.php.net/manual/en/intro-whatcando.php

[20] Introduction. *Laravel* [online]. [cit. 2022-01-30]. Dostupné z: https://laravel.com/docs/4.2/introduction

[21] How Laravel implements MVC and how to use it effectively. *Pusher* [online]. 2018, 22.8.2018 [cit. 2022-01-31]. Dostupné z: https://blog.pusher.com/laravel-mvc-use/

[22] Laravel Ecosystem Tools — How Development Can be Faster and More Effective. *ASPER BROTHERS* [online]. 2019, 20.4.2019 [cit. 2022-01-30]. Dostupné z: https://asperbrothers.com/blog/laravel-ecosystem-tools/

[23] ČÁPKA, David. MVC architektura. *Itnetwork.cz* [online]. [cit. 2022-01-31]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor

[24] XINFE a DELTACEN. MVC Diagram (Model-View-Controller). *Wikimedia Commons: the free media repository* [online]. 2013, 27.5.2013 [cit. 2022-03-05]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/b/b4/MVC_Diagram_%28Model-View-Controller%29.svg

[25] Introduction. *Composer* [online]. [cit. 2022-03-03]. Dostupné z: https://getcomposer.org/doc/00-intro.md

[26] Webhosting. *Wikipedia* [online]. 2021, 27.10.2021 [cit. 2022-01-30]. Dostupné z: https://cs.wikipedia.org/wiki/Webhosting

[27] SALEEM, Salman. What is DigitalOcean and why should you host apps on it?. *The Official Cloudways Blog* [online]. 2021, 20.12.2021 [cit. 2022-01-30]. Dostupné z: https://www.cloudways.com/blog/what-is-digital-ocean/

[28] DigitalOcean Products | Designed for Developers, Built for Businesses. *Digitalocean.com* [online]. [cit. 2022-01-30]. Dostupné z: https://www.digitalocean.com/products

[29] DigitalOcean App Platform: Build, Deploy and Scale Apps Quickly. *Digitalocean.com* [online]. [cit. 2022-01-30]. Dostupné z: https://www.digitalocean.com/products/app-platform

[30] About Heroku. *Heroku* [online]. [cit. 2022-01-30]. Dostupné z: https://www.heroku.com/about

[31] The Heroku product suite. *Heroku* [online]. [cit. 2022-01-30]. Dostupné z: https://www.heroku.com/products

[32] What is a database?. *Oracle | Cloud Applications and Cloud Platform* [online]. [cit. 2022-01-31]. Dostupné z: https://www.oracle.com/cz/database/what-is-database/

[33] KUČEROVÁ, Helena. Databázový model. *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. Praha: Národní knihovna ČR, 2003 [cit. 2022-02-04]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000091&local_base=KTD

[34] KUČEROVÁ, Helena. Hierarchická databáze. *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. Praha: Národní knihovna ČR, 2003 [cit. 2022-02-04]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000103&local_base=KTD

[35] KUČEROVÁ, Helena. Síťová databáze. *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. Praha: Národní knihovna ČR, 2003 [cit. 2022-02-04]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000128&local_base=KTD

[36] KUČEROVÁ, Helena. Relační databáze. *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. Praha: Národní knihovna ČR, 2003 [cit. 2022-02-04]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000126&local_base=KTD

[37] MEHRA, Puran. What Are Object-Oriented Databases And Their Advantages. *C# Corner* [online]. 6.9.2019 [cit. 2022-02-04]. Dostupné z: https://www.c-sharpcorner.com/article/what-are-object-oriented-databases-and-their-advantages2/

[38] KUČEROVÁ, Helena. Dotazovací jazyk. *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. Praha: Národní knihovna ČR, 2003

[cit. 2022-02-05]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000098&local_base=KTD

[39] BRITANNICA, The Editors of Encyclopaedia. SQL. *Encyclopedia Britannica* [online]. 16.6.2021 [cit. 2022-02-05]. Dostupné z: https://www.britannica.com/technology/SQL

[40] What is a Transaction? - Win32 apps. *Microsoft* [online]. 2021, 1.7.2021 [cit. 2022-02-05]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/ktm/what-is-a-transaction

[41] IAN. What does ACID mean in Database Systems?. *Database.Guide* [online]. 2016, 20.6.2016 [cit. 2022-02-05]. Dostupné z: https://database.guide/what-is-acid-in-databases/

[42] About. *PostgreSQL* [online]. [cit. 2022-02-05]. Dostupné z: https://www.postgresql.org/about/

[43] Directory Structure. *Laravel* [online]. [cit. 2022-02-19]. Dostupné z: https://laravel.com/docs/8.x/structure

[44] Artisan Console. *Laravel* [online]. [cit. 2022-02-19]. Dostupné z: https://laravel.com/docs/8.x/artisan

[45] Package.json. *Npm Docs* [online]. [cit. 2022-02-19]. Dostupné z: https://docs.npmjs.com/cli/v8/configuring-npm/package-json

[46] Basic usage. *Composer* [online]. [cit. 2022-02-19]. Dostupné z: https://getcomposer.org/doc/01-basic-usage.md

[47] JavaScript & CSS Scaffolding. *Laravel* [online]. [cit. 2022-02-19]. Dostupné z: https://laravel.com/docs/7.x/frontend

[48] Eloquent: Getting Started. *Laravel* [online]. [cit. 2022-02-19]. Dostupné z: https://laravel.com/docs/8.x/eloquent

[49] Database Migrations. *Laravel* [online]. [cit. 2022-02-19]. Dostupné z: https://laravel.com/docs/8.x/migrations

[50] PILOT, Data. How to use the PostgreSQL DELETE CASCADE. *ObjectRocket* [online]. 2020, 28.2.2020 [cit. 2022-02-22]. Dostupné z: https://kb.objectrocket.com/postgresql/how-to-use-the-postgresql-delete-cascade-1369

[51] Stavové kódy HTTP. *Wikipedia* [online]. 2021, 13.11.2021 [cit. 2022-02-22]. Dostupné z: https://cs.wikipedia.org/wiki/Stavov%C3%A9_k%C3%B3dy_HTTP

[52] JEKEL, Milan. Stavové kódy a hlášení v odpovědi protokolu HTTP. *Interval.cz* [online]. 2002, 29.5.2002 [cit. 2022-02-22]. Dostupné z: https://www.interval.cz/clanky/stavove-kody-a-hlaseni-v-odpovedi-protokolu-http/

[53] What is the meaning of the /dist directory in open source projects?. *Stack Overflow* [online]. [cit. 2022-03-03]. Dostupné z: https://stackoverflow.com/questions/22842691/what-is-the-meaning-of-the-dist-directory-in-open-source-projects

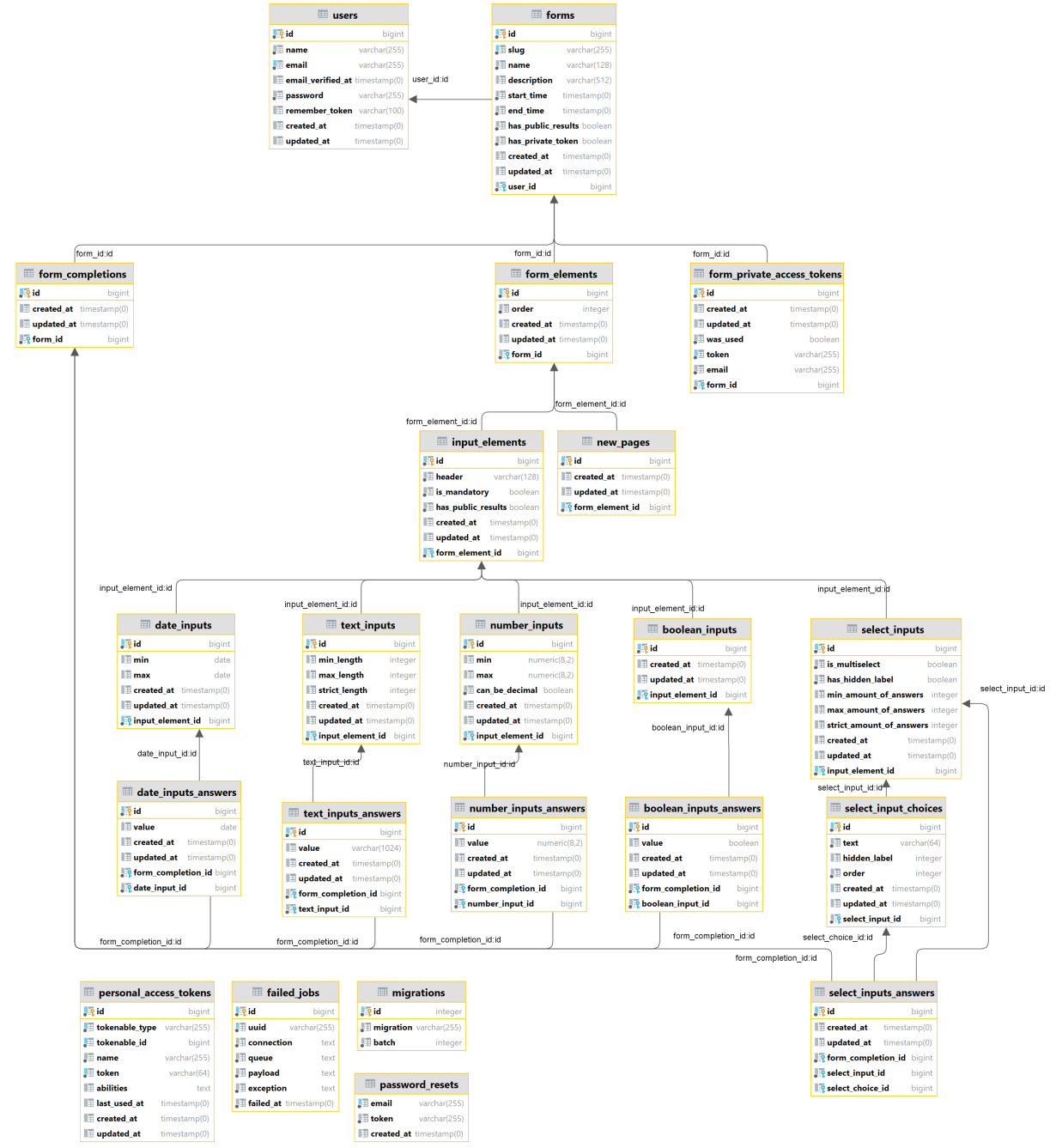
[54] TGUGNANI. Project tour of new Vue CLI App. *5 Balloons* [online]. 2020, 8.5.2020 [cit. 2022-03-03]. Dostupné z: https://5balloons.info/project-tour-of-vue-cli-app/

[55] Vue. *Laravel Mix Documentation* [online]. [cit. 2022-03-03]. Dostupné z: https://laravel-mix.com/docs/6.0/vue

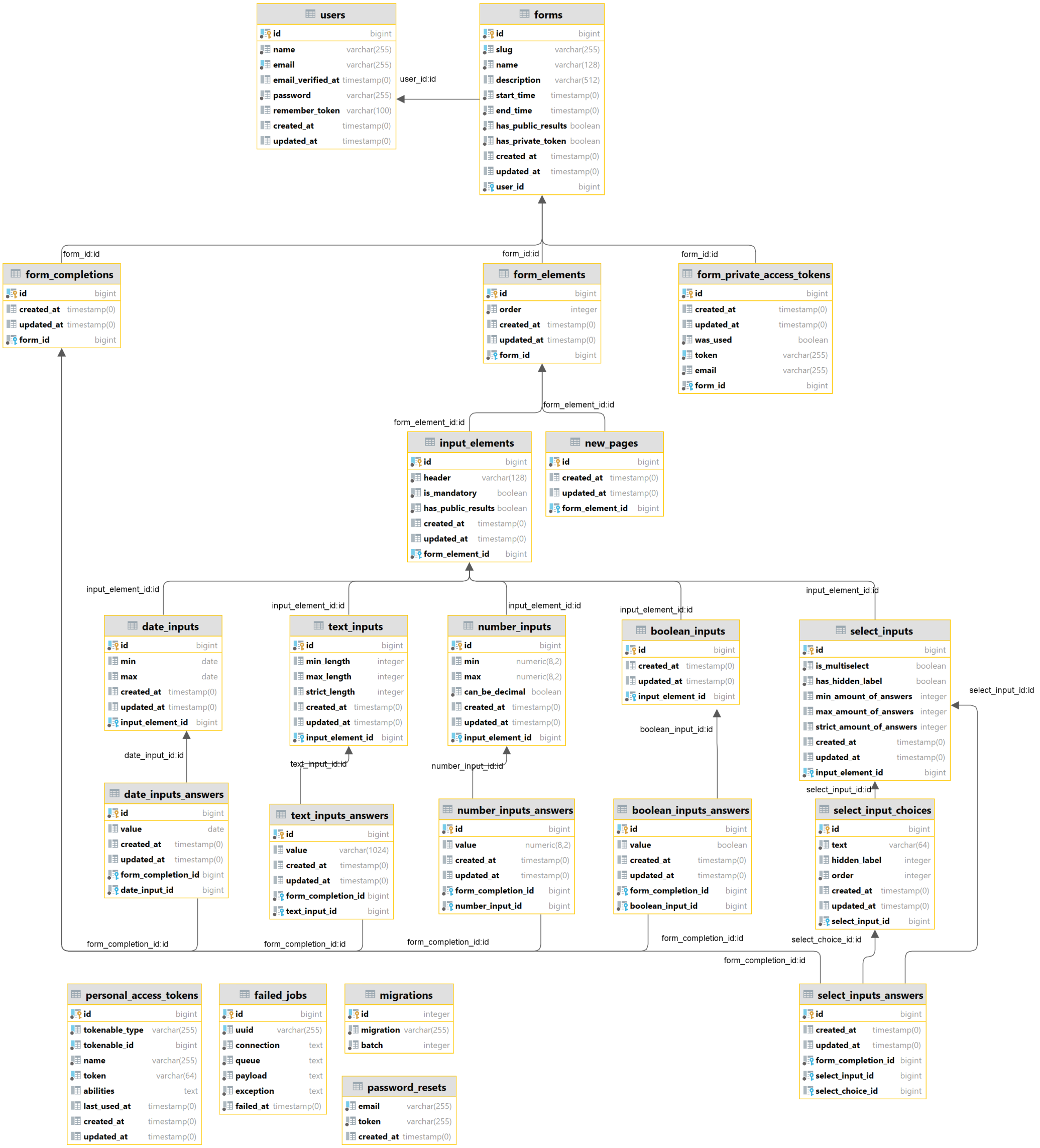
7 Seznam obrázků

8 Seznam tabulek

9 Příloha 1: Schéma databáze



Příloha 1: Schéma databáze



Zadání maturitního projektu z informatických předmětů

Jméno a příjmení:	Jan Hubený
Školní rok:	2021/2022
Třída:	4. B
Obor:	Informační technologie 18-20-M/01
Téma práce:	Systém pro vkládání a zpracování ankety/zpětné vazby
Vedoucí práce:	RNDr. Jan Koupil, Ph.D.

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:
Cílem práce je vytvořit webovou aplikaci pro zpracování a tvorbu ankety, která bude svou funkčností bohatší než běžné formulářové systémy.

Tvůrce formuláře bude mít v systému k dispozici běžné funkcionality jako je přehled vlastních formulářů, jejich správa, modifikace a duplikování nebo omezení viditelnosti datem spuštění a ukončení, základní textové, logické a číselné odpovědi a stránkování formuláře. Systém bude ale také obsahovat předpřipravené komponenty pro sofistikovanější typy hodnocení (typicky škály či multiple-choice odpovědi). Zajímavou ale nepovinnou možností je také větvení formuláře (podmíněné zobrazení otázky či stránky) na základě předchozích odpovědí.

Nabízen bude textový a grafický pohled na výsledky (četnosti odpovědí v případě škál či rozhodovacích odpovědí), ale také takový export dat, včetně škál, aby bylo snadné provádět např. korelační analýzu. Samotné nástroje pro vyhledávání korelací nejsou povinnou částí tohoto zadání. Standardní výsledky formuláře bude také možné přímo v systému publikovat veřejně (s možností skrytí konkrétních otázek).

Systém bude nabízet nejen veřejně dostupné formuláře, ale také možnost automatizovaného rozeslání pozvánky a vyplnění s unikátním kódem, které zajistí jedinečnost odpovědí při zachování anonymity.

1. Autor provede rešerši za účelem nalezení a porovnání existujících řešení anketních systémů
2. Autor navrhne webovou aplikaci pro zpracování ankety společně se správou uživatelských účtů a celkovou administrací, komponenty a formát ankety a způsob zobrazování a exportování dat
3. Autor vytvoří a implementuje navržené řešení a zdokumentuje ho

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

- **Září:** Rešerše dostupných anketních systému, výběr vhodných technologií pro tento projekt
- **Říjen:** Návrh architektury aplikace a databázového modelu
- **Listopad–prosinec:** Tvorba webové aplikace a práce související s implementací
- **Leden–únor:** Testování, tvorba dokumentace
- **Březen:** Dokončení dokumentace