FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Databáze zvířat pro neziskovou organizaci
Student:	Tomáš Taro
Vedoucí:	Ing. Lukáš Maleček
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

Cílem práce je vytvořit novou verzi webové aplikace Plemenná kniha (databázi zvířat), která slouží k evidenci informací o zvířatech v aktivním chovu, evidenci odchovů, schvalování vrhů, pomáhá členům neziskové organizace v plánování chovu a umožňuje jednoduché generování průkazů původu pro odchovy.

* Analyzujte současný stav aplikace, seznamte se s její vnitřní implementací, současnou strukturou databáze a již implementovanými funkcemi.
* Na základě analýzy navrhněte optimálnější strukturu databáze (zejména vyřešte současné duplicitní ukládání dat).
* Vytvořte zcela novou aplikaci pro správu plemenné knihy v PHP s použitím MySQL databáze. V aplikaci využijte současné návrhové a architektonické vzory a postupy.
* Implementujte nové funkce na základě jejich specifikace dodaných vedoucím práce.
* Ověřte správnou funkci aplikace pomocí unit a případně i integračních testů.
* Migrujte data z původní aplikace do nové a navrhněte postup, jak zajistit kontrolu a ověření správnosti importu.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 21. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Databáza zvierat pre neziskovú organizáciu

Tomáš Taro

Katedra softwarového inženýrství

Vedúci práce: Ing. Lukáš Maleček

23. mája 2020

Poďakovanie

Touto cestou by som sa chcel poďakovať pánovi Ing. Lukášovi Malečkovi za vedenie a trpezlivosť pri riešení problematiky týkajúcej sa tejto bakalárskej práce. Taktiež by som sa chcel poďakovať rodine za neustálu podporu počas celého štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 23. mája 2020 . . . . . .

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na túto prácu

Taro, Tomáš. *Databáza zvierat pre neziskovú organizáciu.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

V niekoľkých vetách zhrňte obsah a prínos tejto práce v slovenčine. Po prečítaní abstraktu by mal čitateľ mať dosť informácií pre rozhodnutie, či Vašu prácu chce čítať.

Klíčová slova Nahraďte zoznamom kľúčových slov v slovenčine oddelených čiarkou.

Abstract

Sem doplňte ekvivalent abstraktu Vašej práce v angličtině.

Keywords Nahraďte zoznamom kľúčových slov v angličtine oddelených čiarkou.

Obsah

Úvod **1**

1 Analýza **3**
1.1 Analýza súčasnej aplikácie 3
1.1.1 Architektúra aplikácie 3
1.1.2 Databázový model aplikácie 5
1.2 Analýza požiadaviek 11
1.2.1 Funkčné požiadavky 11
1.2.2 Nefunkčné požiadavky 15
1.3 Používateľské role 17
1.3.1 Bežný používateľ 17
1.3.2 Registrátor zvierat 17
1.3.3 Schvalovateľ vrhov 18
1.3.4 Administrátor 18
1.4 Prípady použitia 18
1.4.1 Prípady použitia zvierat 18
1.4.2 Prípady použitia vrhov 21

2 Návrh **25**
2.1 Návrh architektúry 25
2.1.1 Architektúra MPA aplikácie 25
2.1.2 Architektúra SPA aplikácie 27
2.1.3 Voľba architektúry 29
2.2 Voľba technológií 29
2.2.1 PHP 29
2.2.2 HTML 5 30
2.2.3 CSS 3 30
2.2.4 JavaScript 30
2.3 Návrh databázy 31

2.3.1 Normalizácia databázy 31
2.3.2 Návrh databázového modelu 33

3 Implementácia **39**
3.1 Serverová časť 39
3.1.1 Štruktúra projektu 39
3.1.2 MVC architektúra 40
3.2 Klientská časť 44
3.3 Migrácia dát 47

4 Testovanie **51**
4.1 Jednotkové testy 51
4.2 Integračné testy 51

Literatúra **55**

A Používateľské rozhranie **59**

Zoznam obrázkov

1.1 Ukážka súboru vrhy_detail_zvirete.php 4
1.2 Logický model súčasnej databázy 10

2.1 Diagram znázorňujúci architektúru MPA aplikácií 26
2.2 Diagram znázorňujúci architektúru SPA aplikácií 27
2.3 Logický model navrhnutej databázy 38

3.1 Ukážka modelovej triedy Animal.php 40
3.2 Ukážka šablóny animal_overview.blade.php 42
3.3 Ukážka controlleru AnimalController.php 43
3.4 Ukážka routeru web.php 44
3.5 Ukážka šablóny komponentu AnimalPage.vue 45
3.6 Ukážka biznis logiky AnimalPage.vue komponentu 46
3.7 Ukážka volania import metód v routri web.php 48

4.1 Ukážka integračného testu zmazania používateľa 52

Úvod

Používanie webových aplikácií sa stalo bežnou súčasťou našich životov vďaka rozšírenému používaniu mobilných zariadení schopných pripojiť sa na internet. Ich používatelia nie sú len bežní ľudia, ale aj záujmové organizácie, medzi ktoré taktiež patrí organizácia „Český klub potkanů“. Táto organizácia používa webovú aplikáciu pre evidenciu informácií o zvieratách v aktívnom chove a ich vrhoch, pomáha jej členom v plánovaní odchovov a umožňuje pre ne generovať jednoduché preukazy. Avšak problémom existujúcej aplikácie je jej zastaranosť, ťažkopádnosť pri jej používaní a použitie zlých návrhových vzorov pri jej vytváraní.

Táto bakalárska práca sa zaoberá vytvorením novej webovej aplikácie, ktorá nahradí starú aplikáciu a tým uľahčí organizácii evidenciu jednotlivých informácií.

Rešeršná časť práce sa venuje analýzou existujúcej aplikácie z pohľadu architektúry aplikácie a jej databázy. Na túto časť nadviaže analýza požiadaviek s popisom prípadov použití, ktoré poslúžia ako základ implementácie jednotlivých funkcií. Následne sa práca zaoberá vybraním vhodnej architektúry potrebnej pre beh aplikácie, technológií, ktoré aplikácia bude používať a návrhom databázy, ktorá vyrieši existujúci problém ukladania duplicitných dát.

Implementačná časť práce sa venuje procesu implementácie aplikácie z pohľadu serverovej a klientskej časti. Následne pokračuje opisom procesu migrácie dát z existujúcej databázy do novej databázy navrhnutej pre novú aplikáciu.

V neposlednom rade sa práca venuje testovaniu aplikácie pomocou integračných testov, ktoré overia, či jej implementácia prebehla korektne podľa požiadaviek definovaných v analytickej časti práce.

Kapitola 1

Analýza

Táto kapitola sa venuje analýze súčasnej webovej aplikácie, ktorá zahŕňa analýzu jej architektúry a databázového modelu. Následne sú vymedzené jednotlivé funkčné a nefunkčné požiadavky novej webovej aplikácie, na ktoré nadväzuje sekcia používateľských rolí. Záver kapitoly je určený pre analýzu prípadov použití, ktoré vyplývajú z funkčných požiadaviek.

1.1 Analýza súčasnej aplikácie

Táto sekcia sa zaoberá analýzou súčasnej aplikácie, ktorá bola vytvorená pre potreby neziskovej organizácie viesť evidenciu zvierat, vrhov, ich registrácií a ďalších informácií. Hlavným cieľom analýzy je zistenie súčasného riešenia aplikácie z pohľadu architektúry a štruktúry súčasnej databázy.

1.1.1 Architektúra aplikácie

Samotný kód aplikácie nie je členený – skladá sa z jednej vrstvy – čo znamená, že kód aplikácie zodpovedný za prístup do databázy, logiku aplikácie a zobrazenie aplikácie nie je od seba oddelený.

```
<?php
get_header();
if (!is_user_logged_in()) {
  echo ("<h1>Neautorizováno</h1>\n");
} else {
?>
<h1>Detail zvířete</h1>
<br>
<?php
  if (!empty($_REQUEST['id'])) {
    $req_id = $_REQUEST['id'];
    if (is_numeric($req_id)) {
      $sql_potkan = "SELECT * FROM `" . $tabulka_zvirata . "`
      WHERE `id_prvni_verze` = " . $req_id .
      " ORDER BY `id` DESC";
      $potkani = $wpdb->get_results(
      ^^I$sql_potkan, ARRAY_A, 0
      );
      $zvire = $potkani[0];
      if ($zvire != null) {
?>
  <div id="zvire">
    <fieldset style="
    border: 1px solid #A0A0A0;
    margin: 2px;
    padding: 5px;">
      <legend style="padding: 5px;">
          Základní informace
      </legend>
      <span id="popis">Pohlaví:</span><span id="hodnota">
      <?php
        if ($zvire['pohlavi'] == 'M') {
          echo ("kluk");
        }
        else if ($zvire['pohlavi'] == 'F') {
          echo ("holka");
        }
        else {
          echo ("????");
        }
      ?>
      </span><br>
      ...
    </fieldset>
  </div>
```

Obr. 1.1: Ukážka súboru vrhy_detail_zvirete.php

Obrázok 1.1 obsahuje ukážku súboru súčasnej aplikácie, ktorý je zodpovedný za zobrazenie detailov zvieraťa v danom vrhu. Z ukážky je možné postrehnúť vyššie spomenuté miešanie logiky zodpovednej za funkcionalitu aplikácie (riadok 1–6), kód zodpovedný za prístup do databázy (riadok 10–19) a v neposlednom rade logiku zobrazovania aplikácie (riadok 21–41).

Ako programovací jazyk súčasnej aplikácie bol použitý jazyk PHP, ktorý bol navrhnutý v roku 1994 Rasmusom Lerdorfom. Tento jazyk je univerzálny programovací jazyk na strane servera a je primárne určený na vývoj webových stránok [1].

Pre pridávanie, spracovanie a získavanie dát v aplikácii slúži open-source SQL databázový systém nazývaný MySQL, ktorý je vyvíjaný, distribuovaný a podporovaný spoločnosťou Oracle Corporation [2].

1.1.2 Databázový model aplikácie

Neoddeliteľnou súčasťou tejto práce je analýza databázového modelu existujúcej aplikácie. Na základe tejto analýzy bude navrhnutá taká štruktúra databázy, ktorá nebude spôsobovať nekonzistentnosť a duplicitu dát v aplikácii.

Databáza súčasnej aplikácie sa skladá z nasledujúcich tabuliek:
`czkp_mimi`, `czkp_vrh`, `czkp_zvire`, `pp_informace`, `pp_miminka` a `pp_zadosti`.

1.1.2.1 Popis tabuliek

V tejto podsekcii sú popísané jednotlivé tabuľky, ich štruktúra a najmä význam stĺpcov, do ktorých sa jednotlivé dáta ukladajú. Pre lepšiu predstavu štruktúry databázového modelu aplikácie sa v podsekcii 1.1.2.2 nachádza logický model databázy v grafickej podobe.

Všetky tabuľky obsahujú umelo-vytvorený unikátny primárny kľúč záznamu zvaný `id`, podľa ktorého sa rozlišujú jednotlivé záznamy.

V nasledujúcich dvoch tabuľkách (`czkp_vrh` a `czkp_zvire`) sa vyskytujú tri rovnaké stĺpce, menovite `id_prvni_verze`, `datum_zaznamu` a `uzivatel`. V stĺpci `id_prvni_verze` je uložený identifikátor (`id`) prvej verzie zvieraťa/vrhu. Stĺpec `datum_zaznamu` obsahuje dátum vytvorenia záznamu spolu s jeho časom a v stĺpci `uzivatel` sa nachádza identifikátor používateľa, ktorý daný záznam vytvoril. Tento identifikátor ukazuje na systémovú tabuľku `wp_users`, ktorá obsahuje informácie o používateľoch súčasného systému.

Tabuľka czkp_vrh

Táto tabuľka má za úlohu správu dát týkajúcich sa jednotlivých vrhov. Stĺpec `id_prvni_verze` odkazuje na prvý záznam daného vrhu.
V stĺpci `wp_id_majitel` je uložený identifikátor majiteľa vrhu ukazujúci na systémovú tabuľku `wp_users`. Stĺpce `otec` a `matka` slúžia k ukladaniu identifikátorov otca, resp. matky. Tento identifikátor pochádza z tabuľky `czkp_zvire`.

Následne sú do tejto tabuľky ukladané aj údaje ako označenie vrhu (v stĺpci `oznaceni`), typ vrhu (`typ_priznani`), línia vrhu (`linie`), gény vrhu (`geny_vrhu`), jeho varietnosť (`varietnost`) či dátum narodenia (`datum_narozeni`). U každého vrhu sa taktiež zaznamenáva chovná stanica (`chov_stanice`), v ktorej sa daný vrh narodil, prípadne kontakt na chovateľa, ktorý je uložený v stĺpci `chov_kontakt`. Dôležité informácie, ako počet narodených a odchovaných mláďat sa ukladajú do stĺpcov `nar_mladat`, prípadne `odchov_mladat`. Počet odchovaných mláďat sa delí na počet odchovaných samcov (`odch_kluci`) a samíc (`odch_holky`). Počet mláďat uvoľnených pre chov je možné nájsť v stĺpci `mlad_chovne`. Rozdiel odchovaných mláďat a mláďat uvoľnených pre chov sa rovná počtu mláďat určených na maznáčika — tento údaj je možné nájsť v stĺpci `mlad_pet`.

Jednotlivé vrhy môžu byť zaregistrované pod klubom ČKP — tieto registrácie sa taktiež nachádzajú v tabuľke `czkp_vrh`. Pre účely registrácie vrhu slúžia stĺpce `datum_registrace`, v ktorom je uložený dátum registrácie, `reg_cislo_vrhu`, ktorý obsahuje registračné číslo vrhu a `rok_reg`, ktorý značí, v ktorom roku bol vrh zaregistrovaný. Registrátor, ktorý zaregistroval daný vrh, môže pripojiť poznámku počas registrácie vrhu, ktorá je uložená v stĺpci `poznamka_reg`.

Chovateľ daného vrhu môže taktiež vytvoriť poznámku k vrhu — táto poznámka sa následne uloží do stĺpca `poznamka_chov`.

Ako posledný stĺpec nachádzajúci sa v tabuľke `czkp_vrh` je stĺpec `kompletni`, ktorý značí, či o uloženom vrhu sú dostupné kompletné informácie, avšak tento údaj sa v aplikácii nepoužíva.

Tabuľka czkp_zvire

Tabuľka `czkp_zvire` slúži na ukladanie informácií o jednotlivých zvieratách, ktoré sú sledované organizáciou.

Skladá sa zo stĺpca `id_prvni_verze`, ktorý odkazuje na prvý záznam daného zvieraťa v rovnakej tabuľke – čo znamená, že pomocou tohto stĺpca sa dajú zistiť všetky úpravy daného vrhu spustením príslušného SQL príkazu.

Jej obsahom je taktiež stĺpec `datum_zaznamu`, ktorý je nastavený na dátum a čas vloženia záznamu do tabuľky. Stĺpec `uzivatel` obsahuje identifikátor používateľa, ktorý daný záznam vytvoril. Následne sa v tabuľke nachádzajú stĺpce `uzamceny_upravy`, `uzamceny_kdy` a `uzamceni_kdo`, ktoré sa ale v súčasnej aplikácii nevyužívali. Pri zvierati je potrebné ukladať jeho pohlavie – to je uložené v stĺpci `pohlavi`. Dátum narodenia zvieraťa sa nachádza v stĺpci `datum_narozeni`.

Informácie o chovateľovi, resp. majiteľovi zvieraťa sa ukladajú do šiestich stĺpcov (3 a 3). V stĺpci `chovatel_ckp_id` je uložený identifikátor chovatela (z tabuľky `wp_users`), v `chovatel` meno chovateľa a v `chov_stanice` chovateľská stanica. Informácie o majiteľovi majú podobnú štruktúru, s tým rozdielom že sa dáta ukladajú do stĺpcov `majitel_ckp_id`, `majitel` a `majet_stanice`. V prípade, že chovateľ resp. majiteľ nepochádza z organizácie, jeho identifikátor zostáva prázdny.

Vonkajšie črty zvieraťa, ako farba očí, typ uší, typ srsti, znaky a farba srsti sa ukladajú do stĺpcov `barva_oci`, `typ_ucha`, `typ_srsti`, `bila_kresba` a `barva_srsti`. Pre ukladanie otca a matky zvieraťa slúžia stĺpce `matka` a `otec`, v ktorých sa nachádza jedinečný identifikátor nachádzajúci sa v tejto tabuľke. Pre účely zaznamenania, z akého vrhu dané zviera pochádza, slúži stĺpec `cis_vrhu`, ktoré obsahuje registračné číslo vrhu, z ktorého dané zviera pochádza. Avšak pre jednoznačnú identifkáciu vrhu, odkiaľ zviera pochádza, mal slúžiť stĺpec `id_ckp_vrhu` — ten sa ale nepoužíva.

Pre potreby organizácie sa do tejto tabuľky ukladajú údaje o registrácii zvieraťa — a to typ registrácie (stĺpec `reg_ckp_typ`), číslo registrácie (`reg_ckp_cislo`) a rok registrácie (stĺpec `reg_ckp_rok`). Toto platí iba v prípade, že zviera je zaregistrované pod klubom ČKP. Ak je zviera zaregistrované pod iným klubom, tak sa jeho registračné číslo ukladá do stĺpca `reg_c_ostatni`. Navyše sa ukladá aj identifikátor registrátora, ktorý dané zviera zaregistroval pod klubom ČKP do stĺpca `registrator`. V neposlednom rade je nutné ukladať aj dátum registrácie zvieraťa — pre túto informáciu slúži stĺpec `datum_registrace`.

Informácie o dátume úmrtia a dôvodu úmrtia zvieraťa sa ukladajú do stĺpcov `datum_umrti`, resp. do stĺpca `duvod_umrti`.

Chovateľ zvieraťa si taktiež môže pridať poznámku k zvieraťu, ktorá je následne uložená v stĺpci `poznamka_chovatel`.

Medzi ďalšie informácie, ktoré sa zbierajú o zvierati, patrí aj informácia ohľadom rizikovosti chovu (stĺpec `rizikovost_chovu`) a prípadná poznámka, prečo je chov rizikový. Táto poznámka sa ukladá samostatne do stĺpca `riziko_pozn`.

Predposledný stĺpec `overeno_pk` obsahuje informáciu, či zviera bolo overené podľa plemennej knihy a stĺpec `pozn_edit` obsahuje poznámku, ktorá mohla byť vytvorená pri editácii zvieraťa.

Tabuľka czkp_mimi

Úlohou tejto tabuľky je ukladať informácie o mláďatách narodených v jednotlivých vrhoch. Rozdiel medzi tabuľkou `czkp_mimi` a `czkp_zvire` spočíva v tom, že narozdiel od tabuľky `czkp_zvire`, v tejto tabuľke sú uložené iba mláďata, ktoré pochádzajú zo známeho vrhu.

Tabuľka obsahuje stĺpec `id_vrhu`, v ktorom je uložený identifikátor záznamu z tabuľky `czkp_vrh`. Podľa tohto stĺpcu vieme určiť, v akom vrhu sa dané mláďa narodilo. Pre ukladanie mena a pohlavia mláďata slúžia stĺpce `jmeno`, resp. `pohlavi`. Chovnosť mláďata sa ukladá do stĺpca `chov`. Stĺpec `chov_omez` ďalej slovne špecifikuje chovné obmedzenie mláďata. Pre organizáciu je potrebné sledovať, kto dané mláďa chová, a pre tento účel slúži stĺpec `chov_kdo`. Pre uloženie dodatočných informácií ohľadom mláďata, ako napríklad typ uší, typ srsti, farba srsti, farba očí a znaky slúžia stĺpce `typ_ucha`, `typ_srsti`, `barva_srsti`, `barva_oci` a `znaky`.

Tabuľka pp_informace

Do nasledujúcej tabuľky — `pp_informace` — sa ukladajú informácie spojené s jednotlivými vrhmi. Táto tabuľka nemá žiadnu spojitosť s tabuľkou `czkp_vrh`. Dôvod vzniku tejto tabuľky bol vývoj nadstavby nad pôvodnou aplikáciou.

Táto tabuľka obsahuje stĺpec `uzivatel`, ktorý obsahuje identifikátor majiteľa vrhu ukazujúci do tabuľky `wp_users`. Následne sa do tabuľky ukladajú údaje o samotnom vrhu, ako napríklad typ vrhu (`typ_vrhu`), jeho označenie (`vrh_oznaceni`), dátum narodenia (`vrh_narozeni`), varietnosť (`vrh_varietnost`), počet narodených a odchovaných mláďat (`vrh_nar_mladat`/`vrh_odch_mladat`) a počet odchovaných samcov a samíc (`vrh_odch_kluci`/`vrh_odch_holky`).

Každý vrh má matku a otca vrhu — tieto údaje sú ukladané do stĺpcov `vrh_id_matka` a `vrh_id_otec`. Ich obsahom sú identifikátory rodičov daného vrhu. Títo rodičia pochádzajú z tabuľky `czkp_zvire`.

Informácie o chovateľovi vrhu sa ukladajú do dvoch stĺpcov, a to `chovatel` a `chov_kontakt`. Prvý z menovaných stĺpcov obsahuje meno chovateľa a druhý jeho kontakt.

Nakoľko každý vrh môže byť zaregistrovaný, je nutné ukladať údaje o ich registrácii. Pre tieto účely slúžia stĺpce `reg_dat_vyplneni`, ktorý značí dátum požiadania o registráciu majiteľom vrhu, `reg_dat_schvaleni`, ktorý obsahuje dátum schválenia vrhu a `reg_cislo_vrhu` obsahujúci registračné číslo vrhu.

Tabuľka pp_miminka

Obsahom tabuľky `pp_miminka` sú mláďatá, ktoré boli narodené vo vrhoch uložených v predchádzajúcej tabuľke.

Aby bolo možné zistiť, ku ktorému vrhu jednotlivé mláďa patrí, bolo potrebné, aby tabuľka obsahovala stĺpec `id_pp`, ktorý obsahuje identifikátor vrhu, v ktorom sa mláďa narodilo. Tento identifikátor ukazuje do predchádzajúcej tabuľky, `pp_informace`. Následne sa do tabuľky ukladajú dáta týkajúce sa predovšetkým samotného mláďata, ako napríklad jeho meno (`mimi_jmeno`), pohlavie (`mimi_pohlavi`), typ uší (`mimi_typ_ucha`), typ srsti (`mimi_typ_srsti`), farba srsti (`mimi_barva_srsti`), či farba očí (`mimi_barva_oci`) a jeho znaky (`mimi_znaky`). K danému zvieraťu taktiež prislúcha majiteľ, ktorý sa ukladá do stĺpca `mimi_majitel`. Okrem mena majiteľa sa ukladá nie len jeho kontakt (`mimi_maj_kontakt`), ale aj číslo preukazu majiteľa, v prípade že je zároveň členom organizácie (`mimi_maj_prukaz`). Medzi posledné informácie ukladajúce sa k danému mláďatu patrí taktiež informácia, či je mláďa určené pre chov (`mimi_chov`), prípadne chovné obmedzenie (`mimi_chov_omezeni`).

Tabuľka pp_zadosti

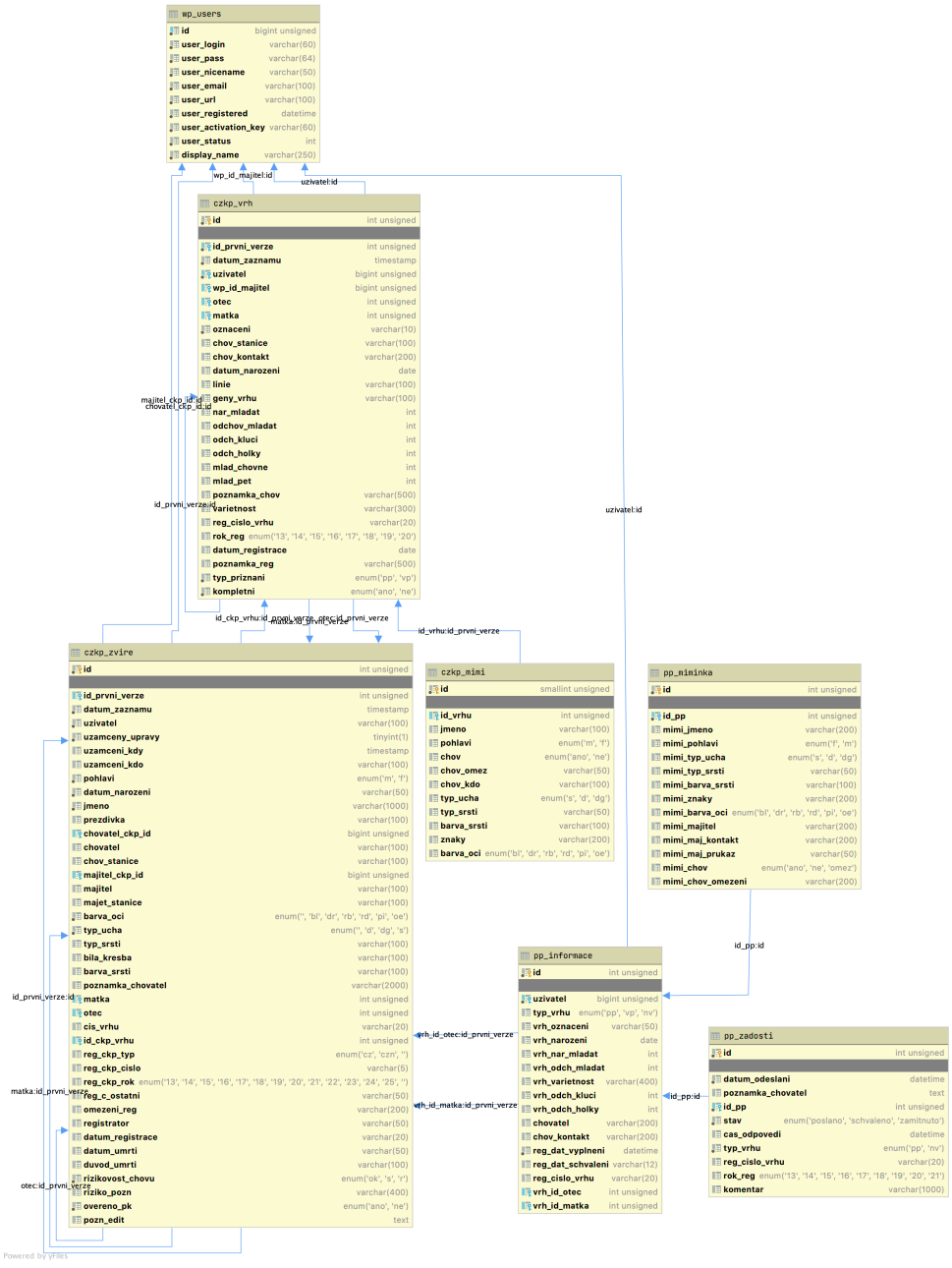
Posledná tabuľka, ktorá sa nachádza v databáze súčasnej aplikácie, je tabuľka s názvom `pp_zadosti`. Táto tabuľka má za úlohu zhromažďovať dáta o poslaných, resp. (ne)schválených žiadostiach o schválenie vrhu. Narozdiel od predchádzajúcich tabuliek, v tejto tabuľke sa nachádzajú stĺpce obsahujúce informácie iba k daným žiadostiam.

Táto tabuľka obsahuje stĺpec `id_pp`, ktorého obsahom je identifikátor vrhu z tabuľky `pp_informace`, ktorého sa týka daná žiadosť o schválenie vrhu. Tabuľka následne obsahuje stĺpec `datum_odoslani`, ktorý je automaticky nastavený na dátum a čas odoslania žiadosti a `cas_odpovedi`, ktorý indikuje dátum a čas odpovedi registrátora na danú žiadosť.

Okrem týchto informácií sa v tabuľke nachádza informácia o type schvaľovaného vrhu (`typ_vrhu`), poznámka žiadateľa o schválenie vrhu (`poznamka_chovatel`) a komentár registrátora ku žiadosti (`komentar`). V prípade, že je žiadosť schválená, `reg_cislo_vrhu` bude obsahovať registračné číslo vrhu a `rok_reg` rok registrácie daného vrhu.

1.1.2.2 Databázová schéma

V tejto podsekcii je možné nájsť logický model databázy v grafickej podobe, ktorý obsahuje schému tabuliek so vzťahmi medzi jednotlivými tabuľkami.



Obr. 1.2: Logický model súčasnej databázy

1.2 Analýza požiadaviek

V tejto sekcii sa nachádza popis všetkých požiadaviek kladených na vznikajúcu webovú aplikáciu. Tieto požiadavky delíme na funkčné a nefunkčné.

1.2.1 Funkčné požiadavky

Funkčné požiadavky vo všeobecnosti vymedzujú hranice aplikácie v kontexte jej funkcionality, ktorú používateľ od aplikácie očakáva. Ich úlohou je taktiež spresniť odhad pracnosti na vznikajúcej aplikácii [3].

1.2.1.1 Evidencia zvierat

Aplikácia musí umožniť jednoduchú správu zvierat, od vytvorenia nového zvieraťa, cez zobrazenie jeho detailov, až po následnú editáciu, prípadne jeho zmazanie.

V evidencii zvierat je pre každé zviera potrebné ukladať nasledujúce údaje:

- meno a prezývku
- dátum narodenia
- majiteľa a chovateľa zvieraťa
- vrh, v ktorom sa zviera narodilo
- matku a otca zvieraťa
- pohlavie
- farbu očí
- typ uší
- farbu a typ srsti
- znaky
- dátum a dôvod úmrtia

Pre uľahčenie vytvorenia a editovania zvieraťa budú textové polia v maximálnej možnej miere interaktívne.

To znamená, že polia majiteľa a chovateľa zvieraťa vrátia na základe vstupu zoznam majiteľov, resp. chovateľov, ktorí už sú evidovaní v aplikácii. Následne

si z tohto zoznamu používateľ zvolí požadovaného človeka. V prípade, že aplikácia požadovaného človeka nevráti, používateľovi bude ponúknutá možnosť ho dodatočne vytvoriť.

Na podobnom princípe bude fungovať aj textové pole pre vrh, s tým rozdielom, že podľa zvoleného vrhu aplikácia predvyplní matku a otca zvieraťa.

1.2.1.2 Evidencia registrácií zvieraťa

Je potrebné, aby aplikácia ponúkala možnosť zaregistrovať evidované zviera pod záujmový chov. Pri takejto registrácií sa budú zbierať nasledovné údaje:

- klub[1], pod ktorým bude zviera zaregistrované
- typ registrácie[2]
- registračné číslo
- dátum registrácie
- informácia, či je nasledovný chov povolený
- informácia o obmedzení chovu

1.2.1.3 Evidencia vrhov

Medzi nevyhnutnú funkcionalitu aplikácie sa radí aj evidencia vrhov. Tak ako pri evidencii zvierat, tak aj v tomto prípade používateľské rozhranie umožní vytvoriť nový vrh, zobraziť ho, respektíve ho editovať alebo vymazať.

Pre účel evidencie vrhov sa budú v aplikácii ukladať nasledujúce údaje:

- typ a označenie vrhu
- majiteľ vrhu
- meno a kontakt na chovateľa
- dátum narodenia
- matka a otec vrhu
- línia

[1] Na výber z možností: ČKP, SOCHP alebo Ostatné
[2] Vyžadované iba v prípade registrácie zvieraťa pod klubom ČKP

- genetické informácie
- počet narodených a odchovaných mláďat
- počet odchovaných samcov a samíc
- počet mláďat určených pre maznanie a pre následný chov

Rovnako ako pri evidencii zvierat, tak aj tu umožní aplikácia interaktívne zvoliť matku, otca a majiteľa vrhu zobrazením zoznamu uložených zvierat/ľudí.

1.2.1.4 Správa žiadostí o schválenie vrhov

Pre organizáciu je žiaduce, aby aplikácia obsahovala správu žiadostí o schválenie vrhu.

V rámci žiadosti o schválenie vrhu rozlišujeme dva typy osôb – žiadateľa o schválenie vrhu[3] a registrátora vrhu.

Aplikácia žiadateľovi umožní poslať žiadosť o schválenie vrhu s možnosťou zanechania poznámky pre registrátora. Po odoslaní a úspešnom spracovaní tejto žiadosti serverom sa odošle e-mail všetkým registrátorom s informáciou o vytvorení novej žiadosti o schválenie vrhu. Na tento e-mail bude môcť zareagovať akýkoľvek registrátor, ktorý sa na základe dostupných informácií o vrhu rozhodne, či danú žiadosť schváli alebo zamietne. O zmene stavu žiadosti bude žiadateľ informovaný e-mailom.

1.2.1.5 Zobrazenie rodokmeňu zvieraťa a vrhu

Pre jednoduchšiu vizualizáciu predkov konkrétneho zvieraťa aplikácia ponúkne zobrazenie rodokmeňu zvieraťa vo forme jednoduchej tabuľky.

V tejto tabuľke budú okrem mien zvierat zobrazené aj dodatočné informácie o zvierati, definované v sekcii 1.2.1.1. Rodokmeň bude zobrazený v rámci jednotlivých zvierat a vrhov.

V prípade vrhu bude rodokmeň zobrazovať predkov matky a otca daného vrhu.

1.2.1.6 Tvorba poznámok pre zviera a vrh

V rámci aplikácie bude potrebné implementovať poznámky, ktoré budú môcť byť priradené jednotlivým zvieratám a vrhom. Takáto poznámka by mala mať nastaviteľnú viditelnosť[4] a typ[5]. Taktiež musí poskytnúť informáciu, kedy bola

[3] Zväčša majiteľ daného vrhu

[4] Poznámka môže byť buď verejná alebo súkromná

[5] Typ poznámky je jeden z nasledujúcich: všeobecná, upozornenie alebo výstraha

vytvorená, respektíve editovaná.

Nakoľko sa jedná o poznámky pre zviera a vrh, budú zobrazené u príslušných zvieratách, resp. vrhoch, ku ktorým sa vzťahujú.

1.2.1.7 Zobrazenie histórie zmien zvierat a vrhov

Medzi želanú funkcionalitu novej webovej aplikácie patrí sledovanie a následné zobrazenie histórie zmien pre všetky zvieratá a vrhy

Aplikácia bude zaznamenávať nasledujúce zmeny v systéme:

- vytvorenie zvieraťa/vrhu
- úprava údajov zvieraťa/vrhu
- zmazanie zvieraťa/vrhu
- obnova[6] zmazaného zvieraťa/vrhu

Pri každej zmene popísanej vyššie je taktiež nutné ukladať, kto a kedy danú zmenu vykonal. V prípade úpravy údajov budú navyše zaznamenané tie údaje, ktoré boli zmenené používateľom. Túto históriu zmien bude možné vidieť vo forme tabuľky u každého zvieraťa, resp. vrhu.

1.2.1.8 Generovanie preukazov

Pre potreby organizácie je nevyhnutné implementovať generovanie preukazov (osvedčení) vo forme PDF súboru. V tomto súbore bude prvá strana vyplnená informáciami o danom zvierati 1.2.1.1, vrátane jeho registrácie zo sekcie 1.2.1.2. V niektorých prípadoch bude taktiež zobrazená registrácia vrhu v ktorom sa zviera narodilo, ktorej obsah je definovaný v 1.2.1.4. Druhá strana súboru bude vyplnená rodokmeňom zvieraťa (1.2.1.5).

Preukaz bude možné vygenerovať tlačidlom na stránke konkrétneho zvieraťa.

1.2.1.9 Filtrovanie a radenie

Pre vylepšenie používateľskej skúsenosti s aplikáciou bude potrebné implementovať filtrovanie a radenie zvierat ako aj vrhov na príslušných stránkach so zoznamom zvierat, resp. vrhov. Implementácia tejto funkcionality umožní jednoduchšiu prácu s aplikáciou a rýchlejšie nájdenie potrebných informácií.

[6] Obnoviť zviera/vrh bude môcť iba administrátor aplikácie

1.2.1.10 Správa používatelov a rolí

Pre administrátorov aplikácie je nutné implementovať možnosť správy jednotlivých používateľov aplikácie, priradovať im príslušné role, alebo im ich naopak odoberať. Na základe tejto skutočnosti je žiaduce vytvoriť pohľad so zoznamom používateľov a ich rolami, s následnou možnosťou im danú rolu zmeniť, prípadne daných používateľov odobrať.

1.2.1.11 Lokalizácia aplikácie

Nakoľko budú aplikáciu používať nie len českí ale aj zahraniční používatelia, je nutné, aby aplikácia poskytovala obsah lokalizovaný do anglického jazyka. Jazyk bude možné jednoducho zmeniť v menu na paneli webovej aplikácie.

1.2.2 Nefunkčné požiadavky

Na rozdiel od funkčných požiadaviek, nefunkčné požiadavky umožňujú určiť obmedzenia kladené na aplikáciu. V neposlednom rade majú taktiež zásadný dopad na návrh architektúry webovej aplikácie [3].

1.2.2.1 Webová aplikácia

Nakoľko je požadovaný systém navrhnutý ako webová aplikácia, bude potrebné, aby bola prístupná z internetu pomocou moderných webových prehliadačov.

1.2.2.2 Používateľské rozhranie

Webová aplikácia bude musieť obsahovať používateľské rozhranie, s ktorým budú môcť používatelia interagovať. Prostredie bude naviac responzívne, čo uľahčí prípadný prístup do systému z mobilného prehliadača.

1.2.2.3 Technológie

Po konzultácii s vedúcim práce boli vymedzené nasledovné technológie, ktoré budú použité na strane servera.

Ako programovací jazyk bude použitý jazyk PHP vo verzii 7.3, hoci momentálne existuje novšia verzia jazyku – 7.4 [4]. Dôvod výberu nižšej verzie jazyka je daný PHP podporou webhostingu[7], na ktorom bude aplikácia nasadená.

Pre ukladanie dát potrebných pre funkčnosť aplikácie a ich následné spracovávanie bude použitý rovnaký databázový systém, ako v súčasnej aplikácii – MySQL. Tento databázový systém bude použitý vo verzii 5.6.

[7]Server poskytujúci webovú aplikáciu používateľom na internete, v tomto prípade sa jedná o webhosting Endora – www.endora.cz

Táto verzia taktiež nie je najnovšou verziou (v skutočnosti bola prvýkrát vydaná v roku 2013, avšak je stále oficiálne podporovaná [5]), opäť z dôvodu neexistujúcej podpory novšieho databázového systému webhostingom.

1.3 Používateľské role

Novovznikajúca webová aplikácia bude prístupná iba zaregistrovaným a prihláseným používateľom. Navyše, niektoré akcie budú obmedzené iba pre určitý okruh používateľov.

K tomu, aby sme umožnili prístup k niektorým akciám iba vybraným používateľom, bude potrebné do aplikácie implementovať používateľské role a práva. Následne bude aplikácia riadiť prístup používateľa k jednotlivým akciám na základe príslušnosti k vybranej roli.

V nasledujúcich podsekciách budú priblížené jednotlivé role a k nim príslušné práva, ktoré sa budú vyskytovať v aplikácii.

1.3.1 Bežný používateľ

Túto rolu bude mať každý používateľ automaticky po registrácii do webovej aplikácie. Bežný používateľ bude môcť v aplikácii:

- Vytvoriť a zobraziť svoje zvieratá
- Upraviť svoje zvieratá v prípade, že nie sú zaregistrované pod klubom ČKP
- Vytvoriť a upraviť registrácie svojich vlastných zvierat, ktoré nespadajú pod klub ČKP
- Zobraziť zoznam všetkých cudzích zvierat spolu s ich detailmi
- Vytvoriť a zobraziť vrhy, u ktorých je používateľ ich majiteľom
- Upraviť vrhy, ktorých je majiteľom, pokiaľ neboli tieto vrhy schválené žiadosťou
- Zobraziť všetky vrhy typu VP alebo schválené vrhy typu PP a NV spolu s ich detailmi
- Pridať poznámku k zvieratám a vrhom, ktoré vlastní

1.3.2 Registrátor zvierat

Registrátor zvierat je v hierarchii rolí postavený nad bežným používateľom. Tým pádom má všetky práva bežného používateľa a navyše nasledujúce práva:

- Možnosť pridať poznámku k akýmkoľvek zvieratám
- Možnosť pridať, editovať a vymazať akúkoľvek registráciu u každého zvieraťa

1.3.3 Schvalovateľ vrhov

Schvalovateľ vrhov je taktiež postavený v hierarchii rolí nad bežným používateľom podobne ako registrátor zvierat, s tým rozdielom, že schvalovateľ vrhu môže v aplikácii:

- Vidieť a odpovedať na žiadosti o schválenie vrhov
- Editovať akékoľvek zviera a vrh
- Pridať poznámky k akémukoľvek vrhu
- Vygenerovať preukaz zvieraťa

1.3.4 Administrátor

Ako obvykle, pre administrátora neplatia žiadne reštrikcie, čo znamená, že bude mať prístup ku všetkej funkcionalite definovanej vo funkčných požiadavkách v sekcii 1.2.1.

1.4 Prípady použitia

Prípady použitia sú špecifikácie rôznych činností, ktoré môžu používatelia s aplikáciou vykonávať [6]. Tieto prípady použitia budú zachytené vo forme scenáru a budú vychádzať nielen zo známych funkčných požiadaviek popísaných v sekcii 1.2.1, ale aj z jednotlivých používateľských rolí, ktoré boli definované v sekcii 1.3.

V tejto práci som sa rozhodol venovať iba takým prípadom použitia, ktoré sú z môjho pohľadu pre čitateľa prínosnejšie. Tie som následne rozdelil podľa príslušnosti k jednotlivým prvkom aplikácie. Všetky nasledujúce prípady použitia predpokladajú, že používateľ bude v systéme prihlásený.

Poznámka: Prvé štyri scenáre použitia sú aplikovateľné aj na vrhy, avšak z dôvodu neuvádzania duplicitných informácií nebudú ďalej rozvinuté v príslušnej podsekcii. Zároveň pri následnej aplikácii prvých štyroch scenárov na vrhy, sa výskyt slova „zviera“ automaticky nahrádza slovom „vrh“ v príslušnej podobe.

1.4.1 Prípady použitia zvierat

V tejto podsekcii budú popísané jednotlivé prípady použitia, ktoré sa týkajú zvierat.

UC1 – Vyhľadanie zvieraťa

Vyhľadanie zvieraťa je jeden zo základných prípadov použitia, kedy používateľ chce vyhľadať dané zviera.

Scenár:

1. Používateľ zvolí možnosť „Zvieratá“ z navigačného menu aplikácie.

2. Aplikácia následne zobrazí tabuľku so zvieratami a filtrom, na základe ktorého si používateľ môže nastaviť dodatočné kritéria filtrovania zvierat.

3. Používateľ nastaví filter podľa kritérií hľadaného zvieraťa a klikne na tlačidlo „Filtrovať“.

4. Aplikácia zobrazí všetky zvieratá vyhovujúce zadaným kritériám.

Alternatívny scenár:

3. Aplikácia zobrazí hľadané zviera už po načítaní tabuľky so zvieratami. V tomto prípade scenár vyhľadávania zvieraťa končí.

Tento prípad použitia realizuje viaceré funkčné požiadavky na aplikáciu, konkrétne evidenciu a filtrovanie a radenie zvierat.

UC2 – Zobrazenie detailu zvieraťa

Tento prípad použitia popisuje zobrazenie detailu zvieraťa a zahŕňa prípad UC1 – Vyhľadanie zvieraťa.

Scenár:

1. UC1 – Vyhľadanie zvieraťa

2. Používateľ klikne na meno zvieraťa, ktorého detail si želá vidieť.

3. Aplikácia zobrazí detail zvieraťa.

Popísaný prípad taktiež realizuje funkčnú požiadavku na aplikáciu, a to evidenciu zvierat.

UC3 – Vytvorenie zvieraťa

Medzi ďalší základný prípad použitia patrí pridanie zvieraťa používateľom do systému. Pri vytváraní zvieraťa sa aplikuje dodatočné obmedzenie pre bežného používateľa, ktoré spočíva v nemožnosti zvolenia majiteľa iného ako je on sám (1.3.1).

Scenár:

1. Používateľ zvolí možnosť „Zvieratá“ z navigačného menu aplikácie, ktorá následne načíta stránku so zoznamom uložených zvierat.
2. Následne používateľ klikne na tlačidlo „Pridať zviera“.
3. Aplikácia načíta novú stránku s formulárom pre vytvorenie nového zvieraťa.
4. Používateľ vyplní formulár.
5. Po vyplnení všetkých potrebných údajov pre vytvorenie zvieraťa aplikácia umožní odoslať formulár kliknutím na tlačidlo „Uložiť“.
6. Používateľ klikne na tlačidlo „Uložiť“.
7. Po úspešnom vytvorení zvieraťa na základe vložených údajov systém presmeruje používateľa na stránku so zoznamom uložených zvierat.

Tento prípad použitia realizuje funkčnú požiadavku evidencie zvierat.

UC4 – Úprava zvieraťa

V tomto prípade sa používateľ aplikácie snaží upraviť existujúce zviera v systéme. Tento prípad použitia taktiež zahŕňa prípady UC1 – Vyhľadanie zvieraťa alebo UC2 – Zobrazenie detailu zvieraťa, v závislosti od spôsobu úpravy zvieraťa, ktorý si používateľ zvolí.

Scenár:

1. UC1 – Vyhľadanie zvieraťa
2. Používateľ klikne na rozbalovacie menu u zvieraťa, ktoré chce editovať.
3. Po tomto kroku aplikácia ponúkne možnosť editácie alebo zmazania zvieraťa.
4. Používateľ zvolí možnosť editácie.

5. Aplikácia následne presmeruje používateľa na stránku s editáciou zvieraťa, ktorá bude obsahovať formulár s predvyplnenými údajmi zvieraťa.
6. Používateľ upraví údaje zvieraťa.
7. Kliknutím na tlačidlo „Odoslať“ sa odošle vyplnený formulár na server.
8. Po úspešnom spracovaní formuláru je používateľ presmerovaný na stránku s detailmi upravovaného zvieraťa.

Alternatívny scenár:

1. UC2 – Zobrazenie detailu zvieraťa
2. Používateľ klikne na tlačidlo „Upraviť“ na pravom postrannom paneli.
3. Následne scenár pokračuje bodom 5 v pôvodnom scenári.

Popísaný prípad použitia realizuje funkčnú požiadavku evidencie zvieraťa.

UC5 – Vytvorenie registrácie zvieraťa

Tento prípad použitia popisuje vytvorenie registrácie zvieraťa. Pri samotnom vytváraní novej registrácie sa aplikujú dodatočné obmedzenia pre bežného používateľa, tak ako je popísane v 1.3.1.

Tento prípad použitia zahŕňa prípad zobrazenia detailov zvieraťa.

Scenár:

1. UC2 – Zobrazenie detailu zvieraťa
2. Kliknutím na tlačidlo „Pridať novú registráciu“ aplikácia zobrazí registračný formulár v modálnom okne[8].
3. Používateľ vyplní potrebné polia.
4. Po stlačení tlačidla „Uložiť“ sa vyplnený formulár odošle na server.
5. Po úspešnom spracovaní formuláru serverom sa modálne okno zavrie.

Tento prípad použitia realizuje funkčnú požiadavku kladenú na aplikáciu, konkrétne evidenciu registrácií zvieraťa.

[8] Menšie okno aplikácie prekrývajúce jej hlavný obsah

1.4.2 Prípady použitia vrhov

Táto podsekcia popisuje jednotlivé prípady použitie, ktoré sa týkajú vrhov.

UC6 – Vytvorenie žiadosti o schválenie vrhu

Tento prípad použitia reálne zahŕňa prípad použitia „zobrazenia detailov vrhu“, avšak pre neuvádzanie duplicitných informácii môže čitateľ vychádzať z prípadu použitia zobrazenia detailov zvieraťa. Scenáre sa v oboch prípadoch obsahovo nelíšia, tak ako bolo uvedené poznámke v sekcii 1.4.

Na tento prípad sa taktiež vzťahujú dodatočné obmedzenia pre bežného používateľa vyplývajúce z 1.3.1.

Scenár:

1. UC2 – Zobrazenie detailu zvieraťa
2. Používateľ klikne na tlačidlo „Vytvoriť žiadosť“.
3. Aplikácia následne zobrazí modálne okno s formulárom obsahujúcim textové pole slúžiace pre poznámku registrátorovi.
4. Po stlačení tlačidla „Uložiť“ sa formulár odošle na server.
5. Po úspešnom spracovaní formuláru serverom sa vytvorí nová žiadosť o schválenie vrhu.
6. Modálne okno sa automaticky zavrie.

Popísaný prípad použitia realizuje funkčnú požiadavku správy žiadostí o schválenie vrhu.

UC7 – Odpoveď na žiadosť o schválenie vrhu

Tak ako v predchádzajúcom prípade, tento prípad použitia bude zahŕňať „zobrazenie detailov vrhu“.

Pre tento prípad použitia sa vzťahujú dodatočné obmedzenia – na žiadosť o schválenie vrhu bude môcť odpovedať používateľ majúci rolu schvalovateľa žiadostí vrhov alebo administrátora.

Scenár:

1. UC2 – Zobrazenie detailu zvieraťa
2. Používateľ stlačí tlačidlo „Odpovedať“ pri príslušnej žiadosti o schválenie vrhu.
3. Aplikácia zobrazí modálne okno s formulárom pre schválenie vrhu.
4. Používateľ vyplní formulár a kliknutím na tlačidlo „Uložiť“ sa odošle na server.
5. Server spracuje odoslaný formulár a ihneď odošle e-mail žiadatelovi o stave jeho žiadosti.

Alternatívny scenár:

1. Používateľ klikne na link v e-maili, ktorý mu prišiel po odoslaní novej žiadosti o schválenie vrhu žiadateľom.
2. Ako reakcia na kliknutie na link, operačný systém otvorí predvolený webový prehliadač s adresou vedúcou na detail vrhu.
3. Následne scenár pokračuje bodom 2 v pôvodnom scenári.

Tak ako v predchádzajúcom prípade použitia, aj v tomto prípade použitia je realizovaná funkčná požiadavka správy žiadostí o schválenie vrhu.

Kapitola 2

Návrh

Táto kapitola sa venuje návrhu architektúry budúcej aplikácie, s priblížením dvoch hlavných architektúr používaných pri tvorbe webových aplikácií. Následne sú obe porovnané a podľa výsledkov porovnania je vybraná taká, ktorá bude použitá pri vývoji aplikácie. V ďalšej sekcii sú vymedzené použité technológie budúcou aplikáciou. V neposlednom rade sa kapitola zaoberá návrhom databázy s popisom normalizácie tabuliek a ich zobrazením v schéme v grafickej podobe.

2.1 Návrh architektúry

Dôležitý aspekt pri návrhu aplikácie spočíva vo výbere architektúry, na ktorej bude aplikácia postavená. Na základe jej výberu sa odvíjajú technológie, ktorými bude daná aplikácia disponovať.

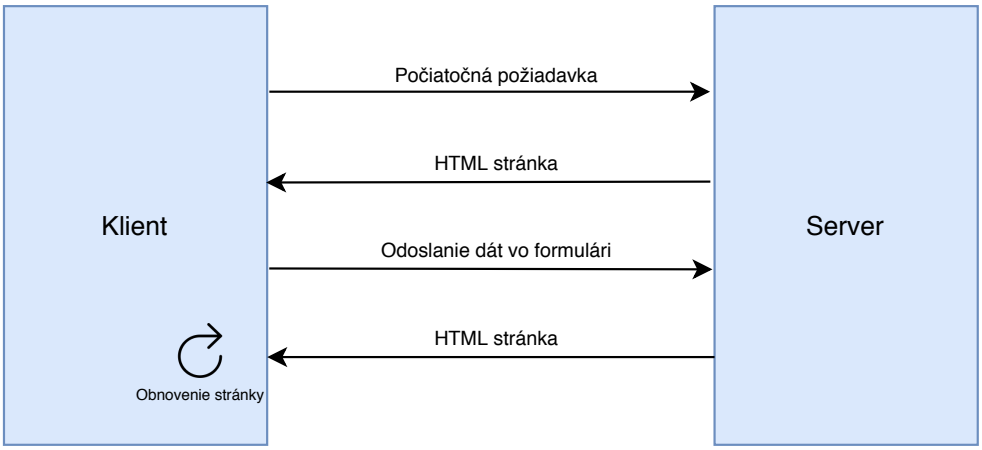
V nasledujúcich sekciách sú zdokumentované dva typy architektúr, z ktorých bude na základe porovnania a vlastných skúseností s tvorbou webových aplikácií vybraná jedna, ktorá bude definovať podobu budúcej webovej aplikácie.

2.1.1 Architektúra MPA aplikácie

MPA je skratka pre viac-stránkovú webovú aplikáciu. Takáto aplikácia vždy načíta celú stránku a zobrazí novú v prípade interakcie s aplikáciou – zvyčajne po odoslaní formuláru používateľom alebo presmerovaním na inú stránku aplikácie [7].

Priebeh komunikácie medzi serverom a klientom

Celkový priebeh komunikácie medzi klientom a serverom zachytáva nasledujúci diagram s jeho popisom.



Obr. 2.1: Diagram znázorňujúci architektúru MPA aplikácií

1. Klient požiada server o stránku
2. Server vráti klientovi požadovanú stránku
3. Klient vrátenú stránku serverom vykreslí
4. Klient vyplní formulár, ktorý sa následne odošle na server
5. Server prijme údaje od klienta, ktoré následne spracuje
6. Po spracovaní údajov klientovi vráti späť stránku s aktualizovanými údajmi
7. Klient túto stránku načíta, čím príde k obnoveniu stránky

Technológia AJAX čiastočne rieši problém znovunačítavania celej stránky dynamickým aktualizovaním tých častí aplikácie, ktoré boli používateľom zmenené. Avšak zakomponovanie tejto technológie do MPA sťažuje a komplikuje celý vývojový proces aplikácie [8].

Výhody použitia MPA architektúry

- Jednoduchá optimalizácia stránok pre webové vyhľadávače [7]
- Umožňuje jednoduchú správu informácií na jednotlivých stránkach [9]
- Je potrebný menší rozsah nástrojov a znalostí narozdiel od vývoja SPA aplikácie [10]
- Veľa dostupných riešení pre vývoj MPA aplikácií [10]

Nevýhody použitia MPA architektúry

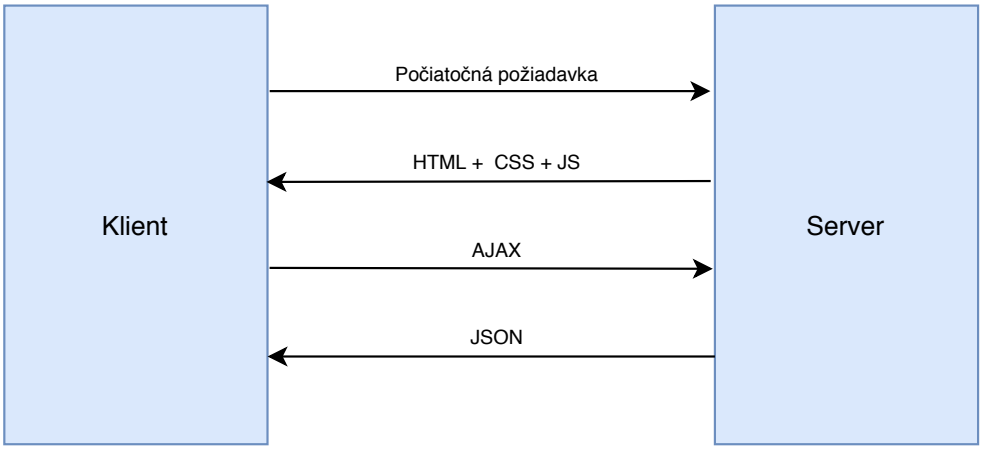
- Zvýšený čas načítavania stránok kvôli neustálemu obnovovaniu stránok (neplatí pri použití technológie AJAX) [7]
- Znížený výkon aplikácie, kvôli neustálemu načítavaniu väčšieho množstva informácií naraz pred následným poslaním stránky používateľovi [7]
- Úzka previazanosť vývoja aplikácie na strane servera a klienta, čo znemožňuje prípadné neskoršie nasadenie rôznych technológií [10]

2.1.2 Architektúra SPA aplikácie

SPA je skratka pre single-page aplikáciu. Single-page aplikácia je aplikácia, ktorá nepotrebuje znovunačítanie stránky počas jej používania. Na tomto type architektúry sú postavené aplikácie ako Gmail[9], Google Mapy[10], Facebook[11] či GitHub[12]. V tomto prípade sa o aktualizáciu obsahu na stránke nestará server, ale klient, zväčša pomocou frameworku[13] bežiacom v prostredí webového prehliadača [9].

Priebeh komunikácie medzi serverom a klientom

Celkový priebeh komunikácie medzi klientom a serverom zachytáva nasledujúci diagram s jeho popisom.



Obr. 2.2: Diagram znázorňujúci architektúru SPA aplikácií

[9]https://gmail.com
[10]https://maps.google.com
[11]https://facebook.com
[12]https://github.com
[13]Sada podporných programov uľahčujúca vývoj aplikácie

1. Klient požiada server o stránku
2. Server vráti klientovi požadovanú HTML stránku so všetkým obsahom aplikácie
3. Klient vrátenú stránku serverom vykreslí
4. Klient vyplní formulár, ktorý sa následne odošle na server pomocou AJAXu
5. Server prijme údaje od klienta, ktoré následne spracuje
6. Po spracovaní údajov server vráti späť klientovi aktualizované údaje (v JSON[14] formáte)
7. Klient zahodí staré údaje na stránke a nahradí ich novými, čím príde k obnoveniu údajov ale nie stránky

Výhody použitia SPA architektúry

- Rýchlosť a responzívnosť aplikácie založenej na aktualizovaní iba tej časti aplikácie, ktorá sa zmenila [7]
- Zjednodušení proces vývoja, nakoľko sa o vykreslenie obsahu nestará server ale klientský framework [9]
- Minimálna previazanosť kódu aplikácie na strane servera a klienta, čo umožňuje jednoduchú správu používaných knižníc bez ovplyvnenia ďalších častí aplikácie [10]

Nevýhody použitia SPA architektúry

- Ťažká optimalizácia aplikácie pre webové vyhľadávače, nakoľko sa o vykreslenie obsahu stará Javascript[15], ktorý webové vyhľadávače neinterpretujú [7]
- Prvé načítanie aplikácie trvá dlhší čas v porovnaní s MPA architektúrou, pretože sa musí všetok obsah aplikácie stiahnuť do zariadenia [7]
- Nemožnosť zobrazenia aplikácie pri vypnutom Javascripte vo webovom prehliadači [10]

[14] Formát na výmenu dát medzi klientom a serverom [11]
[15] Programovací jazyk bežiaci vo webovom prehliadači

2.1.3 Voľba architektúry

Pre budúcu aplikáciu som sa rozhodol vybrať SPA architektúru, nie len z dôvodu rýchlejšieho a jednoduchšieho procesu vývoja, ale aj vďaka vlastným skúsenostiam z oblasti vývoja SPA aplikácií. Nakoľko aplikácia bude dostupná iba vopred vybraním používateľom, nebude potrebné ju optimalizovať pre webové vyhľadávače.

2.2 Voľba technológií

V tejto sekcii sa presunieme od voľby architektúry, na ktorej bude aplikácia postavená, ku voľbe technológií, ktoré budu použité vo výslednej aplikácii.

2.2.1 PHP

Medzi zvolené technológie bude jednoznačne patriť programovací jazyk PHP, nakoľko jeho voľba je jedným z nefunkčných požiadaviek kladených na budúcu aplikáciu.

Framework

Vývoj softvéru je komplexný proces, ktorý zahŕňa nie len výslednú implementáciu konkrétneho softvéru, ale aj analýzu, návrh, a v neposlednom rade aj testovanie softvéru. Softvérové frameworky uľahčujú prácu vývojárom tým, že im umožňujú prevziať kontrolu nad celým procesom vývoja softvéru alebo jeho väčšiny [12].

Medzi niektoré výhody použitia frameworku patrí:

- Pomoc pri zavádzaní lepších programovacích postupov a vhodnom využívaní návrhových vzorov
- Väčšia bezpečnosť výsledného kódu
- Možnosť vyhnúť sa duplicite kódu
- Skrátenie času vývoju výslednej aplikácie

Pre vývoj webových aplikácií v PHP existujú viaceré frameworky, ako napríklad CakePHP[16], Nette[17], Symfony[18], Laravel[19] a iné. Spomedzi menovaných bol vybraný na základe doterajších skúseností posledný z nich – Laravel.

[16] https://cakephp.org
[17] https://nette.org
[18] https://symfony.com
[19] https://laravel.com

2.2.2 HTML 5

HTML – HyperText Markup Language – je značkovací jazyk používaný pre definovanie významu a štruktúry dokumentov. Ako už z názvu vyplýva, HTML používa značky pre definovanie elementov v dokumente s rozdielnymi vlastnosťami, ako napríklad nadpis, odstavec, tabuľky a iné. Tieto značky sú interpretované webovým prehliadačom, ktorý ich následne vykreslí na stránku [13].

Najnovšou verziou tohto jazyka je verzia HTML 5.

2.2.3 CSS 3

Cascading Style Sheets (CSS) je jazyk, ktorý sa používa na ilustráciu vzhľadu, štýlu a formátu dokumentu a jeho elementov napísaného v akomkoľvek značkovacom jazyku. Využíva sa najmä v spojitosti s webovými stránkami. CSS je tvorený pravidlami aplikovanými na jednotlivé elementy dokumentu. Tieto pravidlá sú definované selektorom elementu, ktorý určuje polohu elementu v dokumente, a CSS pravidlami rôzneho typu, ktoré definujú konkrétny štýl daného elementu [14].

CSS3 je momentálne najnovšia verzia jazyka CSS.

2.2.4 JavaScript

JavaScript je jednoduchý programovací jazyk, pomocou ktorého môžu vývojári webových stránok vytvárať interaktívne a dynamické webové stránky. Bol vyvinutý spoločnosťou Netscape, ale v súčasnosti ho používa väčšina webových prehliadačov. Je to programovací jazyk s otvoreným zdrojovým kódom, ktorý môže ktokoľvek používať [15].

Framework

Nakoľko novnovznikajúca aplikácia bude postavená na SPA architektúre, bude potrebné, aby sa stránky vykresľovali na strane klienta. To sa dá samozrejme dosiahnuť bez použitia frameworku, ktorý by sa postaral o aktualizáciu komponentov na stránke, avšak z dôvodu rýchlejšieho procesu vývoja som sa rozhodol použiť framework Vue.js[20].

Vue.js je progresívny framework pre vytváranie používateľských rozhraní. Tento framework bol navrhnutý od základov tak, aby bol postupne prispôsobiteľný. Nakoľko je zameraný iba na vrstvu zobrazenia, je ľahké ho integrovať s inými knižnicami a existujúcimi projektmi [16].

[20] https://vuejs.org

Pre vytváranie používateľského rozhrania existujú aj iné frameworky, ako napríklad React[21] a Angular[22], avšak z dôvodu doterajších skúseností bol zvolený Vue.js framework.

2.3 Návrh databázy

V predchádzajúcej kapitole v podsekcii 1.1.2 bol zanalyzovaný databázový model súčasnej aplikácie.

Táto sekcia popíše normalizáciu súčasnej databázy a na základe poznatkov vyplývajúcich z analýzy o súčasnom databázovom modeli bude navrhnutý databázový model pre vznikajúcu aplikáciu, ktorý bude brať do úvahy funkčné požiadavky kladené na aplikáciu.

2.3.1 Normalizácia databázy

Normalizácia databázy je proces efektívneho usporiadania údajov v databáze. Tento proces má dva ciele: odstránenie nadbytočných údajov (napríklad neukladanie tých istých údajov vo viac ako jednej tabuľke) a zabezpečenie ukladania iba súvisiacich údajov do tabuľky. Normalizovaná databáza znižuje množstvo miesta potrebného na ukladanie dát a navyše zabezpečí, že dáta sú logicky usporiadané [17].

Pri návrhu databázy je žiaduce dosiahnuť tretiu normálovú formu (3NF) pre každú tabuľku v databáze, pretože až takáto databáza sa považuje za normalizovanú [18].

Proces normalizácie tabuľky zahŕňa jej konverziu do prvej normálovej formy (ak v nej nie je), následne do druhej normálovej formy (ak ju nespĺňa) a na koniec do tretej normálovej formy.

Tabuľka je v prvej normálovej forme (1NF), ak spĺňa nasledujúce vlastnosti [19]:

- obsahuje iba atomické hodnoty v každom stĺpci
- hodnoty uložené v stĺpci sa viažu iba na daný stĺpec
- všetky stĺpce v tabuľke majú unikátny názov
- nezáleží na poradí ukladania dát do stĺpcov

[21] https://reactjs.org
[22] https://angular.io

Tabuľka je v druhej normálovej forme (2NF), ak má nasledujúce vlastnosti [19]:

- tabuľka spĺňa 1NF
- nemôže obsahovať čiastočnú závislosť

Čiastočná závislosť v tabuľke existuje pokiaľ atribút v tabuľke závisí od časti primárneho kľúča a nie od celého kľúča [20].

Na to aby tabuľka bola v tretej normálovej forme (3NF), musí navyše mať nasledujúce vlastnosti [19]:

- tabuľka spĺňa 2NF
- nemôže obsahovať tranzitívnu závislosť

Tranzitívna závislosť predstavuje tri alebo viac atribútov, ktoré majú funkčnú závislosť medzi sebou, čo znamená, že stĺpec A závisí na stĺpci B cez stĺpec C [21].

Tabuľka czkp_vrh

Na základe analýzy a splnenia požiadavky 3NF bude potrebné, aby údaje o chovateľovi sa nenachádzali priamo v tabuľke vrhov, ale cez cudzí kľúč smerujúci do tabuľky určenej pre chovateľov/majiteľov. Taktiež bude potrebné presunúť údaje o registrácii vrhu a poznámok do tabuliek pre ne určených.

Tabuľka czkp_zvire

Z vykonanej analýzy tejto tabuľky vyplýva, že tabuľka okrem údajov o samotnom zvierati obsahuje navyše informácie o entitách ako majiteľ/chovateľ, registrácia zvieraťa, prípadne poznámky viažuce sa k danému zvieraťu. Pretože výsledná tabuľka má byť normalizovaná, nemala by obsahovať tieto údaje priamo, ale cez cudzí kľúč.

Pre každú horeuvedenú entitu by mala vzniknúť jedna tabuľka — zvlášť pre chovateľov a majiteľov, registrácie zvierat a poznámky. Z pohľadu databázy netreba rozoznávať ľudí, či sú daní ľudia chovateľmi alebo majiteľmi, preto môžu byť uložení v jednej tabuľke.

Tabuľka czkp_mimi

Analýza tejto tabuľky ukázala, že by tabuľka spĺňala 3NF v prípade, ak by stĺpec chovateľa obsahoval cudzí kľúč namiesto textu. Pri porovnaní tejto ta-

buľky a tabuľky `czkp_zvire` je možné vidieť, že stĺpce definované v tejto tabuľke sa štruktúrou rovnajú stĺpcom v tabuľke `czkp_zvire`, z čoho vyplýva, že táto tabuľka môže byť spojená s tabuľkou `czkp_zvire`.

Tabuľka pp_informace

Táto tabuľka je obdobou tabuľky `czkp_vrh`, čo znamená, že tieto dve tabuľky môžu byť zlúčené do jednej, po oddelení údajov o registácii vrhu a presunutí údajov o chovateľovi vrhu do samostatnej tabuľky.

Tabuľka pp_miminka

Tak ako tabuľka `czkp_mimi`, tak aj táto obsahuje mláďatá narodené vo vrhu (ktorý je uložený v `pp_informace`). Pozitívum oproti tabuľke `czkp_mimi` je jej kompaktnosť (obsahuje menej informácií, ktoré sa viažu nepriamo k tejto tabuľke), avšak stále nespĺňa 3NF vďaka ukladaniu informácií o majiteľovi priamo v tejto tabuľke. Po presunutí dát o majiteľoch do samostatnej tabuľky, môže byť táto tabuľka zlúčená s tabuľkou `czkp_mimi` (ktorá bude zlúčená s tabuľkou `czkp_zvire`).

Tabuľka pp_zadosti

Štruktúra tejto tabuľky porušuje 3NF tým, že si ukladá typ vrhu, ku ktorému sa daná žiadosť viaže, pretože typ vrhu je uložený už pri samotnom vrhu a nie je potrebné ho ukladať aj pri žiadosti o schválenie vrhu.

2.3.2 Návrh databázového modelu

Na základe informácií o štruktúre databázy a funkčných požiadaviek sa táto podsekcia zaoberá výsledným návrhom databázového modelu, ktorý zahŕňa popis navrhnutých tabuliek a zjednodušenú schému logického modelu v grafickej podobe, ktorého základom tvoria navrhnuté tabuľky.

2.3.2.1 Popis tabuliek

Všetky navrhnuté tabuľky používaju stĺpec `id` ako jednoznačný identifikátor daného záznamu. Tento identifikátor je unikátny a pri každom vložení nového záznamu do databázy sa tento identifikátor inkrementuje o 1. Prvý záznam v danej tabuľke má `id` nastavený na hodnotu 1.

Tabuľka animals

Táto tabuľka bude zoskupovať všetky zvieratá, ktoré organizácia sleduje bez ohľadu na to, či sa jedná o mláďa z vrhu alebo o dospelé zviera.

Jej štruktúra sa skladá zo stĺpcov `creator_id`, čo je identifikátor používateľa, ktorý dané zviera vytvoril. Následne štruktúra tabuľky obsahuje identifikátory `breeder_id` a `owner_id`. V prvom prípade sa jedná o identifikátor človeka, ktorý je chovateľom zvieraťa, kdežto v druhom sa jedná identifikátor majiteľa zvieraťa. Pre identifikáciu matky a otca zvieraťa sa používajú stĺpce `mother_id` a `father_id`. Obsahom stĺpcu `litter_id` je identifikátor vrhu, z ktorého zviera pochádza.

Meno zvieraťa, jeho prezývka, pohlavie, dátum narodenia, farba očí, typ uší, farba srsti, typ srsti a jeho znaky budú ukladajú do príslušných stĺpcov, a to `name`, `nickname`, `sex`, `birthdate`, `eyes_color`, `ear_type`, `fur_color`, `fur_type` a `markings`.

Údaje o úmrtí zvieraťa ako dátum úmrtia a dôvod úmrtia, budú uložené v stĺpcov `death_data`, respektíve `death_reason`.

Informácia, či je povolený chov, alebo je nejakým spôsobom limitovaný, bude možné nájsť v stĺpcoch `breeding_available`, respektíve `breeding_limitation`.

Aplikácia bude obsahovať podporu pre generovanie PDF preukazov. V danom preukaze sa budú nachádzať údaje z tejto tabuľky a z tabuľky `litters`. Jedným z týchto údajov je aj meno majiteľa zvieraťa, jeho kontakt a číslo členského preukazu, ktorý vydáva organizácia. Tieto údaje by normálne mohli byť uložené v tabuľke určenej pre chovateľov a majiteľov. Avšak v prípade, že by sa tieto údaje zmenili priamo v tabuľke, v ktorej sa nachádza daný majiteľ zvieraťa, nastala by situácia, že by sa pri následnom generovaní rovnakého preukazu líšili údaje o majitelovi zvieraťa medzi týmto preukazom a preukazom vygenerovaným pred zmenou údajov. Preto sa tieto údaje budú taktiež ukladať v tabuľke zvierat – konkrétne v stĺpcoch `owner_name`, `owner_contact` a `owner_member_card_number`.

Tabuľka animal_registrations

Nakoľko zvieratá môžu byť zaregistrované pod rôznymi klubmi, je potrebné, aby sa tieto údaje ukladali do tabuľky na to určenej.

Štruktúra tejto tabuľky sa skladá z `animal_id`, čo je identifikátor zvieraťa, ktorého sa daná registrácia zvieraťa týka. Následne sa ukladá identifikátor registrátora, ktorý dané zviera zaregistroval (`registrator_id`). Medzi ďalšie údaje, ktoré sa ukladajú do tejto tabuľky, patrí klub, v ktorom je zviera zaregistrované (`club`), typ registrácie (`type`), jeho registračné číslo (`registration_number`) a rok registrácie (`year`). Každý klub má možnosť obmedziť následný chov zvieraťa a špecifikovať dodatočné informácie, prečo je následný chov zvieraťa obmedzený.

Pre tieto účely budú slúžiť stĺpce `breeding_available`/`breeding_limitation`.

Tabuľka litters

Táto tabuľka bude zhromažďovať všetky vrhy sledované organizáciou.

Medzi údaje, ktoré je potrebné ukladať o každom vrhu, patria identifikátory oboch rodičov vrhu – v prípade matky sa jedná o stĺpec `mother_id` a v prípade otca o stĺpec `father_id`. Okrem iného sa bude ukladať aj identifikátor majiteľa vrhu do stĺpca `owner_id` a identifkátor používateľa, ktorý daný záznam vytvoril (`creator_id`).

Následne sa do tabuľky budú ukladať údaje ako typ vrhu (`type`), jeho označenie (`label`), dátum narodenia vrhu (`birthdate`), línia vrhu (`line`) a genetické informácie (`genetic_information`).

Vo vrhu je nutné sledovať počet narodených a odchovaných mláďat (v stĺpcoch `babies_born` a `babies_reared`), odchovaných samcov a samíc (`reared_boys` a `reared_girls`) a počet mláďat určených pre chov a pre maznanie — tieto údaje budú uložené v stĺpcoch `for_breeding` a `for_petting`.

Nakoľko aplikácia bude podporovať generovanie preukazov vo formáte PDF, budú v tejto tabuľke uložené údaje o mene chovateľa a jeho kontaktných údajov z rovnakých dôvodov, ktoré boli uvedené v 2.3.2.1.
Meno chovateľa a jeho kontaktné údaje budú uložené v stĺpcoch `breeder_name`, resp. `breeder_contact`.

Tabuľka litter_approval_requests

Účelom tejto tabuľky je spracovávať žiadosti o schválenie vrhov. Úlohou schvalovateľa je posúdiť jednotlivé žiadosti a na základe poskytnutých údajov o vrhu rozhodnúť, či daný vrh schváli alebo nie.

Pre tie účely bude potrebné, aby tabuľka obsahovala identifikátor vrhu, ku ktorému sa daná žiadosť viaže.
Tento identifikátor bude uložený v stĺpci `litter_id`. Okrem tohto identifikátora sa bude taktiež ukladať identifikátor žiadateľa, ktorý danú žiadosť vytvoril — tento údaj sa automaticky uloží pri vytvorení žiadosti do stĺpca `creator_id`. Po posúdení žiadosti a rozhodnutí, či bude daný vrh schválený alebo nie, sa bude ukladať identifikátor schvalovateľa, ktorý danú žiadosť schválil alebo zamietol, a to do stĺpca `registrator_id`.

Žiadosti môžu mať tri stavy, a to stav „poslaná“, „schválená“ a „zamietnutá“. Stav jednotlivých žiadostí sa bude ukladať do stĺpca `state`.

Žiadateľ o schválenie vrhu bude mať možnosť pridať poznámku schvalovateľovi – táto poznámka sa bude ukladať do stĺpca `creator_note`.

Pre schválenú žiadosť je potrebné ukladať jej registračné číslo, dátum registrácie, prípadne poznámku schvalovateľa pre žiadateľa (tá môže byť uložená i v prípade odmietnutia schválenia vrhu). Tieto údaje sa budú ukladať do stĺpcov `registration_number`, `registration_date`, resp. `registrator_note`.

Tabuľka stations

Chovné stanice, ktorých súčasťou môže byť majiteľ resp. chovateľ zvieraťa/vrhu, budú uložené v tejto tabuľke. Štruktúra tabuľky je jednoduchá — obsahuje stĺpec `name`, v ktorom je uložené meno chovnej stanice, a stĺpec `creator_id`, ktorý obsahuje identifikátor osoby, ktorý danú stanicu vytvoril.

Tabuľka people

Táto tabuľka je určená pre zhromažďovanie údajov o chovateľoch a majiteľoch zvierat a vrhov.

O týchto ľuďoch je potrebné evidovať osobné údaje ako ich meno, email, telefónne číslo a číslo členského preukazu.
Tieto informácie sa budú ukladať do stĺpcov `name`, `email`, prípadne do stĺpcov `telephone_number` a `member_card_number`.

Okrem týchto základných dát sa budú ukladať dodatočné informácie, ako identifikátor chovateľskej stanice, ktorej je daný človek súčasťou (`station_id`), identifikátor používateľa, ktorý daný záznam vytvoril (`creator_id`) a identifikátor používateľa (`user_id`), ak sa používateľ nachádza v systémovej tabuľke `users`. Ak je identifikátor `user_id` vyplnený, znamená to, že daný človek je systémovým používteľom aplikácie, ale aj zároveň chovateľom, resp. majiteľom nejakého zveiraťa/vrhu.

Tabuľka users

Systémová tabuľka `users` bude obsahovať dáta o používateľoch systému. O týchto používateľoch je potrebné ukladať ich meno (`name`) a email (`email`), podľa ktorého sa budú daní používatelia identifikovať pri prihlásení do aplikácie. Taktiež bude potrebné ukladať ich heslo v zahešovanej podobe do stĺpca `password`.

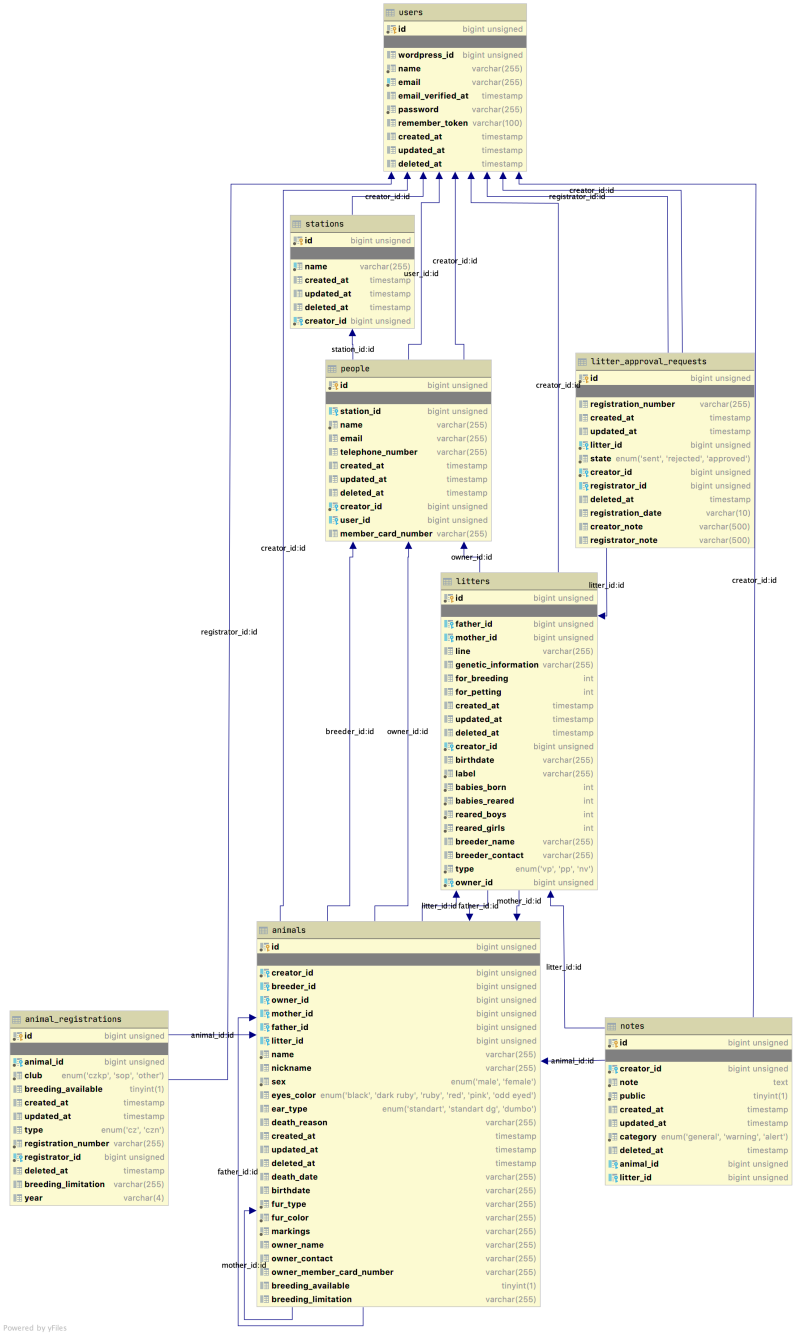
Pre účely budúceho rozšírenia aplikácie — konkrétne prihlasovania sa pomocou systému WordPress do novej aplikácie — sa bude ukladať do stĺpca `wordpress_id` identifikátor používateľa vo WordPress systéme.

Tabuľka notes

Nakoľko je žiaduce ukladať poznámky k jednotlivým zvieratám a vrhom, bude vytvorená tabuľka `notes`, do ktorej sa jednotlivé poznámky budú ukladať. Táto tabuľka bude obsahovať stĺpec `creator_id`, do ktorého sa bude ukladať identifikát používateľa, ktorý daný záznam vytvoril. Ak sa bude jednať o poznámku ku zvierati, jeho identifikátor sa uloží do stĺpca `animal_id`. V opačnom prípade – pri ukladaní poznámky viažucej sa k vrhu – sa identifikátor daného vrhu uloží do stĺpca `litter_id`. Ďalej pre účel poznámok bude potrebné ukladať ich kategóriu, viditeľnosť a obsah. Tieto informácie sa budú ukladať do stĺpcov `category`, `public` a `note`.

2.3.2.2 Databázová schéma

Pre lepšiu vizuálizáciu navrhnutých tabuliek a ich vzťahov medzi nimi sa v tejto podsekcii nachádza zjednodušený logický model databázy v grafickej podobe.



Obr. 2.3: Logický model navrhnutej databázy

Kapitola 3

Implementácia

Cieľom tejto kapitoly je na základe predošlej analýzy súčasnej aplikácie a návrhu databázy priblížiť implementačnú časť novej aplikácie. Najprv sa kapitola venuje implementácii aplikácie na serverovej časti s priblížením architektonického vzoru MVC, ktorý sa využíva pri tvorbe webových aplikácií. Následujúca sekcia je vymedzená implementácii klientskej časti aplikácie. Záver je venovaný migrácii dát z pôvodnej do novej aplikácie.

3.1 Serverová časť

Implementácia serverovej časti webovej aplikácie sprevádzala využitie zvoleného PHP frameworku, ktorým bol Laravel.

3.1.1 Štruktúra projektu

Pre inicializáciu nového Laravel projektu som použil príkaz
`laravel new pedigree-book`. Tento príkaz vytvoril nasledujúcu adresárovú štruktúru.

- `app` ..... Adresár určený pre implementáciu aplikácie
- `bootstrap` ..... Adresár určený pre aplikačnú cache
- `config` ..... Adresár obsahujúci konfiguračné súbory aplikácie
- `database` ..... Adresár určený pre databázové migrácie, továrne a seeds
- `public` ..... Adresár určený pre všetky verejne dostupné súbory
- `resources` ..... Adresár určený pre klientské skripty a Vue.js elementy
- `routes` ..... Adresár obsahujúci router
- `storage` ..... Adresár určený pre súbory generované frameworkom
- `tests` ..... Adresár určený pre aplikačné testy
- `vendor` ..... Adresár obsahujúci nainštalované serverové balíčky

Po spustení príkazu `php artisan serve` sa spustil server
na adrese `localhost:8000`. Po navštívení tejto adresy vo webovom prehliadači bolo možné vidieť predvolenú úvodnú stránku Laravelu.

3.1.2 MVC architektúra

Laravel je implementovaný na základe architektonického vzoru MVC, ktorý je akronymom pre Model-View-Controller. Použitie tohto vzoru rozdeľuje aplikáciu do troch hlavných skupín — na modely, views a controllery. Nasledujúce podsekcie vysvetlia význam jednotlivých skupín.

Model

Model (modelová trieda) obsahuje logiku aplikácie a všetko, čo do tejto logiky spadá. Môžu to byť rôzne výpočty, databázové dotazy, validácia prichádzajúcich dát a podobne. Funkcia modelu spočíva v prijatí parametrov, ktoré spracuje s možným využítím dát z databázy. Následne takto spracované dáta vráti späť metóde, ktorá danú funkciu modelu volala [22].

Platí, že pre každú tabuľku v databáze by mala byť vytvorená príslušná modelová trieda. Takáto modelová trieda je následne úzko spätá s danou tabuľkou, nakoľko poskytuje rôzne metódy pre prácu s dátami uloženými v tejto tabuľke [23].

Laravel pristupuje k týmto dátam pomocou frameworku Eloquent, ktorý je určený pre objektovo-relačné mapovanie. Objektovo-relačné mapovanie (ORM) je technika prístupu k dátam uloženým v databáze. Táto technika spočíva v mapovaní riadkov príslušnej tabuľky v relačnej databáze do objektov, ktoré sú instancie jednotlivých modelov. Táto technika funguje aj opačným smerom.

```
1  /**
2   * Create new animal
3   * @param $data
4   * @return Animal
5   * @throws Throwable
6   */
7  public static function createAnimal($data) {
8      $animal = new Animal($data);
9      $animal->creator_id = Auth::id();
10     $animal->saveOrFail();
11     return $animal->refresh();
12 }
```

Obr. 3.1: Ukážka modelovej triedy Animal.php

Na obrázku 3.1 je možné vidieť metódu `createAnimal`, ktorá je zodpovedná za vytvorenie nového zvieraťa. Táto metóda sa nachádza v modelovej triede `Animal`, ktorá je modelovou triedou tabuľky `animals`. V prvom riadku tela metódy prebieha vytvorenie novej instancie modelu, ktorá je naplnená dátami nachádzajúcimi sa v parametri `$data`. Následne je parameter `creator_id` objektu `Animal` nastavený na identifikátora používateľa, od ktorého požiadavka na vytvorenie nového zvieraťa prišla. Nie je náhoda, že názov tohto parametru je možné nájsť ako stĺpec v tabuľke `animals`. Zavolaním Eloquent metódy `saveOrFail` sa daný objekt pretransformuje do SQL príkazu určeného pre pridanie dát do tabuľky. Následnej sa tento príkaz na pozadí vykoná. V prípade, že uloženie objektu do databázy z rozličných dôvodov zlyhá, táto metóda vráti výnimku. Nakoniec sa zavolaním metódy `refresh` aktualizuje daná instancia modelu aktuálnymi dátami prislúchajúcimi k danej instancii z databázy (o identifikátor zvieraťa a systémové stĺpce) a vráti.

View

View je časť aplikácie, ktorá zodpovedá za vykreslenie obsahu používateľom. Jej obsahom je šablóna, obsahujúca HTML stránku a tagy nejakého značkovacieho jazyka, ktorý umožňuje do šablóny vkladať premenné, prípadne vykonávať iterácie a podmienky. View nie je len šablóna, ale zobrazovač výstupu obsahujúca minimálne množstvo logiky potrebnej pre výpis dát. Podobne ako model, view nevie, odkiaľ mu dáta prišli, iba sa stará o ich finálne zobrazenie používateľovi [22].

V prípade Laravelu sa používa v šablónach značkovací jazyk Blade. Blade umožňuje nie len vytvárať jednotlivé šablóny, ale aj komponenty, ktoré môžu byť použité viackrát v rôznych šablónach. Tieto šablóny sa nachádzajú v adresári `resources/views`.

```
1  <section id="breeding-info" class="bold">
2      <span class="breeding_available">
3          @isset($isAnimalAvailableForBreeding)
4              {{ $isAnimalAvailableForBreeding ?
5                 'Chov povolen' : 'Chov není povolen'
6              }}
7          @endisset
8      </span>
9      <p class="breeding_limitation">
10         @isset($animalBreedingLimitation)
11         {{ !empty($animalBreedingLimitation()) ?
12            ('Poznámka: ' . $animalBreedingLimitation) : ''
13         }}
14         @endisset
15     </p>
16 </section>
```

Obr. 3.2: Ukážka šablóny animal_overview.blade.php

Na obrázku 3.2 sa nachádza ukážka časti šablóny, ktorá je zodpovedná za vygenerovanie preukazu obsahujúci detaily zvieraťa. Ako si je možné všimnúť, obsah tejto šablóny sa skladá z použitia HTML jazyka pre definovanie štruktúry preukazu, zo špeciálnych Blade príkazov ako `isset` a `endisset`, a z funkcie s premennými v syntaxe jazyka PHP.

Controller

Controller je poslednou časťou tejto skladačky, ktorá ozrejmí funkčnosť celého MVC vzoru. Jedná se o prostredníka, s ktorým komunikuje používateľ, model i view. Tieto komponenty drží pohromade a zabezpečuje komunikáciu medzi nimi [22]. Samotné controllery aplikácie sa nachádzajú v adresári `app/Http/Controllers`.

```
1  /**
2   * Show view with animal's template for .pdf export
3   *
4   * @param int $id
5   * @return Factory|View
6   * @throws AuthorizationException
7   */
8  public function export(int $id) {
9      $animal = Animal::getAnimalForPdf($id);
10
11     // Check, whether user is authorized to view this page
12     $this->authorize('canExportAnimal', $animal);
13
14     $litter = $genealogy = null;
15
16     // Is animal part of some litter?
17     if (isset($animal->litter_id)) {
18         $litter = Litter::getLitter($animal->litter_id);
19         $genealogy = Animal::getGenealogy($litter, 0);
20     }
21
22     (...)
23
24     return view('generated_docs.pdf', [
25         'animal' => $animal,
26         'litter' => $litter,
27         'genealogy' => $genealogy,
28         'orientation' => $orientation
29     ]);
30 }
```

Obr. 3.3: Ukážka controlleru AnimalController.php

Na obrázku 3.3 sa nachádza ukážka metódy `export`. Z obrázku je možné vidieť volanie metód modelov `Animal` a `Litter`. Po tom čo controller dostane dáta

z jednotlivých metód modelov, sú tieto dáta predané view, ktorý následne ich doplní do šablóny `pdf`. Nakoniec controller šablónu s dátami vráti, a framework sa postará o jej odoslanie do klientského prehliadača, ktorý danú šablónu zobrazí používateľovi.

Avšak, na akom základe sa spustí táto metóda? Odpoveďou na túto otázku je časť aplikácie zvaná „Router“. Router je zodpovedný za spustenie metód controllerov na základe prijatého HTTP požiadavku (buď iniciovaného používateľom navštívením danej adresy alebo skriptom, ktorý pošle požiadavku na danú adresu). Na základe URL adresy požiadavku router rozhodne, ktorú metódu daného controlleru spustí.

Router aplikácie sa nachádza v priečinku `routes`.

```
1 Route::middleware('auth:sanctum')->group(function() {
2     Route::get(
3     '/animals/{id}/export',
4     'AnimalsController@export'
5     )->name('animals.id.export.get');
6
7     (...)
8 });
```

Obr. 3.4: Ukážka routeru web.php

Na obrázku 3.4 je zobrazená ukážka routeru. V prípade, ak HTTP požiadavka prichádzajúca na server bude mať adresu v tvare „https://domena.sk/animals/<id>/export“[23], router spustí metódu `export` v controlleri `AnimalsController`.

3.2 Klientská časť

Klientská časť aplikácie sa skladá prevažne z implementácie jednotlivých stránok, ktoré sa skladajú z viacerých nezávislých komponent, ktoré spolu komunikujú. Tieto stránky sú zobrazované klientskym frameworkom Vue.js, ktorý sa stará o správne zobrazenie jednotlivých stránok na základe adresy, na ktorej sa používateľ nachádza.

Spustením príkazu `npm install vue` sa nainštaloval framework Vue.js spolu s jej závislosťami do adresára `node_modules`, ktoré sú potrebné pre beh samotného frameworku.

[23] <id> sa nahradí ľubovoľným číslom

Komponenta, ktorá zobrazí stránku zvieraťa, vyzerá nasledovne:

```
<template>
    <div id="animal-page" class="columns section">
        <left-panel
            :animal="animal"
            :user="user">
        </left-panel>
        <animal-information
            :animal-id="animalId"
            :animal="animal"
            :is-loading="isLoading"
            :user="user">
        </animal-information>
        <right-panel
            :animal-id="animalId"
            :is-loading="isLoading"
            :animal="animal"
            :user="user"
            :key="animalId">
        </right-panel>
    </div>
</template>
```

Obr. 3.5: Ukážka šablóny komponentu AnimalPage.vue

Komponenty vo Vue.js sa definujú pomocou štandardných HTML značiek. Okrem iného jednotlivým komponentám môžu byť predané premenné rôzneho typu. Vue.js taktiež podporuje vkladanie existujúcich komponent do výslednej komponenty. Týmto spôsobom sa dá dosiahnuť znovupoužiteľnosť jednotlivých komponent naprieč veľkým množstvom stránok.

V horeuvedenom prípade komponent `AnimalPage` obsahuje tri komponenty:

- LeftPanel
- AnimalInformation
- RightPanel

Samotná komponenta obsahuje nie len jej deklaráciu v rámci HTML jazyka, ale aj biznis logiku v jazyku JavaScript. Väčšinou sa jedná o metódy, ktoré buď načítavajú dáta zo servera, alebo reagujú na používateľský vstup.

V prípade `AnimalPage` komponenty sa jedná o nasledujúcu logiku:

```
<script>
    export default {
        name: "AnimalPage",
        components: {RightPanel, LeftPanel, AnimalInformation},
        data() {
            return {
                animal: null,
                isLoading: false,
            }
        },
        computed: {
            animalId() {
                return Number(this.$route.params.animal);
            }
        },
        methods: {
            async loadData() {
                this.isLoading = true;
                const url = `/api/animals/${this.animalId}`;
                try {
                    const request = await axios.get(url);
                    this.animal = request.data;
                } catch (e) {
                    this.$buefy.toast.open({
                        message:
                        this.$t('animal.index.page_load_fail'),
                        type: 'is-danger'
                     });
                    throw e;
                } finally {
                    this.isLoading = false;
                }
            }
        },
        async created() {
            await this.loadData();
        }
    }
</script>
```

Obr. 3.6: Ukážka biznis logiky AnimalPage.vue komponentu

Hodnota atribútu `name` určuje meno danej komponenty naprieč celou klientskou aplikáciou. Obsah objektu `components` deklaruje použité komponenty v rámci `AnimalPage`. Metóda `data` vracia objekt s reaktívnymi atribútami — pri zmene jednotlivých atribútov Vue.js notifikuje komponentu o zmene atribútu. V prípade, že sa daný atribút používa v komponente, Vue.js zaistí jeho znovuvykreslenie. Dané atribúty sa taktiež používajú na ukladanie stavu jednotlivých komponent. Objekt `methods` obsahuje biznis metódy, ktoré môžu byť voláne inými metódami. V tomto prípade sa používateľská metóda `loadData` spustí po tom, čo sa spustí exekúcia metódy `created`. Metóda `created` je ale systémová – konkrétne sa exekuuje po vytvorení danej komponenty na stránke.

Zjednodušený životný cyklus tejto komponenty bude nasledovný[24]:

1. Vue.js vytvorí a zobrazí komponentu `AnimalPage`
2. Po vytvorení tejto komponenty bude automaticky spustená metóda `created`, ktorá vzápätí zavolá metódu `loadData`
3. Daná metóda sa postará o načítanie informácii o zvierati s identifikátorom `animalId`
4. V prípade zlyhania načítavania dát sa zobrazí chybová hláška, v opačnom prípade sa informácie o zvierati uložia do premennej `animal`
5. Obsah premennej `animal` sa spropaguje do komponentov, ktoré prijímajú objekt `animal`
6. Ak daná komponenta závisí pri jej zobrazení od obsahu premennej `animal`, bude znovu vykreslená

Po zmene jednotlivých komponentov je potrebné spustiť príkaz `npm run prod`, ktorý jednotlivé komponenty skompiluje do formy zrozumiteľnej prehliadaču. Pri vývoji aplikácie je však komfortnejšie použiť príkaz `npm run watch`, ktorý na pozadí zaistí automatickú kompiláciu komponentov po každej ich zmene. Po následnom znovunačítaní stránky prehliadača (`localhost:8000`) sa zobrazia najnovšie zmeny aplikácie.

3.3 Migrácia dát

Po implementácii novej aplikácie bolo potrebné zmigrovať existujúce dáta do novej aplikácie. Proces migrácie spočíval vo vytvorení starých tabuliek

[24]Informácie o kompletnom životnom cykle komponentov je možné nájsť na https://vuejs.org/v2/guide/instance.html

s dátami v databáze novej aplikácie. Následne boli pre tieto tabuľky vytvorené modelové triedy, ktoré obsahujú logiku pre transformáciu dát zo starých do nových tabuliek. Tieto modelové triedy je možné nájsť v priečinku `app/ImportModels`.

Každá modelová trieda obsahuje metódu `import`. Tieto metódy sú volané v jednotlivých metódach controllera `ImportController`, ktoré sú zaregistrované v routri `web.php` 3.7.

```
1  Route::get(
2      '/import/contacts',
3      'ImportController@importContacts'
4  )->name('import.contacts.get');
5  Route::get(
6      '/import/czkp_animals',
7      'ImportController@importCZKPAnimals'
8  )->name('import.czkp_animals.get');
9  Route::get(
10     '/import/czkp_litters',
11     'ImportController@importCZKPLitters'
12 )->name('import.czkp_litters.get');
13 Route::get(
14     '/import/czkp_animals/litters',
15     'ImportController@connectAnimalWithLitter'
16 )->name('import.czkp_animals.litters.get');
17 Route::get(
18     '/import/czkp_babies',
19     'ImportController@importCZKPBabies'
20 )->name('import.czkp_babies.get');
21 Route::get(
22     '/import/pp_information',
23     'ImportController@importPPInformation'
24 )->name('import.pp_information.get');
```

Obr. 3.7: Ukážka volania import metód v routri web.php

Pred importom dát muselo byť veľké množstvo záznamov manuálne očistených, z dôvodu nekonzistentnosti mien majiteľov/chovateľov, ich kontaktných údajov a príslušnosti k jednotlivým chovným staniciam medzi súčasnými tabuľkami. Táto nekonzistentnosť dát bola spôsobená ich ukladaním do nenormalizovaných tabuliek. Čistenie dát zahŕňalo manuálnu kontrolu a dodatočnú úpravu vyše 3500 záznamov v súčasnej databáze.

Ak by sa záznamy neočistili, nebolo by možné nie len napárovať jednotlivé zvieratá a vrhy k ich majiteľom a chovateľom, ale ani importovať týchto ľudí do tabuľky `people` a napárovať ich s jednotlivými chovnými stanicami z tabuľky `stations`.

Kapitola 4

Testovanie

Po implementácii aplikácie je nevyhnutné aplikáciu otestovať, najmä ak sa jedná o komplexnejšiu aplikáciu.Testovanie aplikácie je nevyhnutnou súčasťou pre potvrdenie, že implementovaná funkcionalita funguje podľa špecifikácie

4.1 Jednotkové testy

Jednotkové testy sú testy, ktoré testujú jednotlivé komponenty softvéru. Účelom týchto testov je overiť, či každá komponenta softvéru funguje tak, ako je navrhnutá. Jednotka je najmenšia testovateľná súčasť akéhokoľvek softvéru. Zvyčajne má jeden alebo niekoľko vstupov a zvyčajne jeden výstup [24].

Nakoľko biznis logika tejto aplikácie nie je rozsiahla, boli namiesto jednotkových testov vytvorené integračné testy.

4.2 Integračné testy

Integračné testovanie je úroveň testovania aplikácie, pri ktorej sa jednotlivé komponenty kombinujú a testujú ako skupina. Účelom tejto úrovne testovania je odhaliť poruchy v interakcii medzi integrovanými komponentami [25].

Samotné integračné testy sa zameriavajú na testovanie komunikácie medzi klientom a serverom, pri ktorých sa overuje aktuálna a očakávaná odpoveď servera. Počas behu jednotlivých testov dochádza k overeniu autentifikácie používateľa, prístupu do databázy, respektíve overeniu práv používateľa k danej akcii.

Príkazom `php artisan test` sa spustia všetky implementované testy. Tento príkaz je potrebné spustiť z hlavného adresára webovej aplikácie.

```
1  public function testUserDelete() {
2      // Create new user
3      $user = factory(User::class)->create();
4
5      // Deleting unauthenticated should result in 401
6      $this->deleteJson(route('users.destroy', ['user' => $user->id]))
7          ->assertStatus(401);
8
9      // Login as user
10     $this->loginAsUser();
11
12     // Deleting as user should result in 403
13     $this->deleteJson(route('users.destroy', ['user' => $user->id]))
14         ->assertStatus(403);
15
16     // Login as admin
17     $this->loginAsAdmin();
18
19     // Now the deletion should be alright
20     $this->deleteJson(route('users.destroy', ['user' => $user->id]))
21         ->assertStatus(204);
22 }
```

Obr. 4.1: Ukážka integračného testu zmazania používateľa

Na ukážke testu 4.1 je zobrazený priebeh testu zmazania používateľa. V tomto teste je využitá kontrola prihlásenia a následná kontrola práv. Ak je používateľ odhlásený, respektíve nemá dostatočné práva pre zmazanie používateľa, server by mal vrátiť odpoveď s HTTP status kódom 401, resp. 403. Po prihlásení sa ako administrátor by malo byť možné zmazať vytvoreného používateľa poslaním DELETE požiadavky na server.

Na podobnom princípe sú postavené ostatné integračné testy. Integračné testy boli vytvorené pre všetky entity, ktoré sú dostupné cez API servera. Tieto testy sa nachádzajú v adresári `tests/Feature`.

Pri jednotlivých testoch sa nevyužíva MySQL databáza ako v prípade samotnej webovej aplikácie, ale databáza SQLite, ktorá je nakonfigurovaná tak, aby fungovala v pamäti počítača. Jej podrobnú konfiguráciu spolu s konfiguráciou testov je možné nájsť v súbore `phpunit.xml` nachádzajúci sa v hlavnom adresári webovej aplikácie.

Literatúra

[1] Pros and Cons of PHP Programming Language. [webstránka][cit. 4. 5. 2020] (vlastný preklad). Dostupné z: https://www.pros-cons.net/pros-and-cons-of-php-programming-language/

[2] What is MySQL? [webstránka][cit. 4. 5. 2020] (vlastný preklad). Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[3] Mlejnek, J.: *Analýza a sběr požadavků*. [online prezentácia][cit. 3. 5. 2020]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/161920/mod_resource/content/3/03.prednaska.pdf

[4] PHP: Releases. [webstránka][cit. 4. 5. 2020]. Dostupné z: https://www.php.net/releases/index.php

[5] Mysql :: mysql 5.6 release notes 2020. [webstránka][cit. 4. 5. 2020]. Dostupné z: https://dev.mysql.com/doc/relnotes/mysql/5.6/en/

[6] Arlow, J.; Neustadt, I.; Kiszka, B.: *UML 2 a unifikovaný proces vývoje aplikací*. Computer Press, 2007, 95 s., [webstránka][cit. 4. 5. 2020].

[7] Adomavicius, S.: Which and when to use: SPA vs MPA - JAXenter. 2020, [webstránka][cit. 5. 5. 2020](vlastný preklad). Dostupné z: https://jaxenter.com/spa-vs-mpa-160963.html

[8] Quanit, M.: Single Page Application vs. Multi-page Application. 2020, [webstránka][cit. 5. 5. 2020](vlastný preklad). Dostupné z: https://dev.to/mquanit/single-page-application-vs-multi-page-application-p2m

[9] Kukhnavets, P.: SPA vs MPA: Key Difference Between Single-Page and Multi-Page Apps. 2020, [webstránka][cit. 5. 5. 2020](vlastný

preklad). Dostupné z: https://cuspy.io/blog/spa-vs-mpa-what-is-the-difference/

[10] Melnik, I.: Single Page Application (SPA) vs Multi Page Application (MPA): Pros and Cons. 2020, [webstránka][cit. 5. 5. 2020](vlastný preklad). Dostupné z: https://merehead.com/blog/single-page-application-vs-multi-page-application/

[11] JSON. 2020, [webstránka][cit. 5. 5. 2020](vlastný preklad). Dostupné z: https://www.json.org/json-en.html

[12] Singh, V.: What is Frameworks? [Definition] Types of Frameworks. 2020, [webstránka][cit. 6. 5. 2020](vlastný preklad). Dostupné z: https://hackr.io/blog/what-is-frameworks

[13] HTML: Hypertext Markup Language. 2020, [webstránka][cit. 6. 5. 2020](vlastný preklad). Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[14] What is CSS3? 2020. Dostupné z: https://www.educba.com/what-is-css3/

[15] Belyh, A.: JavaScript. 2020. Dostupné z: https://www.cleverism.com/lexicon/javascript/

[16] 2020. Dostupné z: https://vuejs.org/v2/guide/

[17] Chapple, M.: The Basics of Normalizing a Database. 2020. Dostupné z: https://www.lifewire.com/database-normalization-basics-1019735

[18] Foster, E.; Godbole, S.: *Database Systems: A Pragmatic Approach*. Apress, druhé vydání, 2016, ISBN 9781484211922;1484211928;.

[19] 2020. Dostupné z: https://www.studytonight.com/dbms/database-normalization.php

[20] 2020. Dostupné z: https://www.studytonight.com/dbms/second-normal-form.php

[21] 2020. Dostupné z: https://www.studytonight.com/dbms/third-normal-form.php

[22] Čápka, D.: MVC architektura. 2020. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor

[23] Eloquent: Getting Started. 2020. Dostupné z: https://laravel.com/docs/7.x/eloquent

[24] Unit Testing. 2020. Dostupné z: http://softwaretestingfundamentals.com/unit-testing/

[25] Integration Testing. 2020. Dostupné z: http://softwaretestingfundamentals.com/integration-testing/

Dodatok A

Používateľské rozhranie

V tejto prílohe sa nachádzajú snímky obrazovky finálnej webovej aplikácie s ich popisom. Dáta, ktoré je možné vidieť na jednotlivých snímkoch, boli vytvorené PHP knižnicou `Faker`, ktorá umožňuje generovanie falošných dát so špecifikáciou ich formy.