TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace pro zvířecí útulky

Diplomová práce

Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie

Autor práce: **Bc. Radovan Čapek**
Vedoucí práce: Ing. Igor Kopetschke
Ústav nových technologií a aplikované informatiky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Aplikace pro zvířecí útulky

Jméno a příjmení: **Bc. Radovan Čapek**
Osobní číslo: M19000185
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Zpracujte rešerši aktuálních možností online prodeje zvířat z útulků a zdůvodněte potřebu vašeho řešení.
2. Navrhněte funční model vašeho řešení s důrazem na oddělené rozhraní pro útulky včetně jejich ověřování a na rozhraní pro zájemce o koupi.
3. Implementujte webové rozhraní pro útulky. Toto rozhraní umožní ověřenou registraci útulku, vkládání a editaci nabízených zvířat dle kategorií a včetně popisných metadat pro vyhledávání. Umožněte vložení čísla čipu včetně (pokud to bude možné) ověření v registrech zvířat.
4. Implementujte klientskou část jako webové rozhraní a aplikaci pro Android. Zákazníkům bude umožněno nabízená zvířata filtrovat a vyhledávat dle zadaných metadat.
5. Ve spolupráci s vybranými útulky aplikaci otestujte a získejte zpětnou vazbu od útulků i zákazníků. V závěru práce též navrhněte další možná vylepšení a rozšíření.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah pracovní zprávy: 40 – 50 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. Big Data a NoSQL databáze. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.
[2] LOPEZ, Lionel, 2017. React: QuickStart Step-By-Step Guide to Learning React JavaScript Library. 1. Scotts Valley (California, USA): Createspace Independent Publishing Platform. ISBN 1976210232.
[3] LACKO, Ľuboslav, 2017. Mistrovství – Android. Přeložil Martin HERODEK. Brno: Computer Press. Mistrovství. ISBN 978-80-251-4875-4.

Vedoucí práce: Ing. Igor Kopetschke
Ústav nových technologií a aplikované informatiky

Datum zadání práce: 19. října 2020
Předpokládaný termín odevzdání: 17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2020

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

17. května 2021 Bc. Radovan Čapek

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce panu Ing. Igoru Kopetschke za čas a rady, které mi věnoval. Dále bych chtěl poděkovat všem zástupcům útulků, kteří se podíleli na testování aplikace.

Abstrakt

Cílem práce je inzertní aplikace, která bude podporovat adopci zvířat ze zvířecích útulků. Práce začíná rešerší aktuálních možností online inzerce zvířat. Následuje návrh vlastního řešení v podobě webové aplikace a mobilní aplikace pro platformu Android. Dále je popsána datová struktura aplikace a použité technologie. Hlavní část práce je věnovaná implementaci řešení webové a mobilní aplikace. Závěrečná část analyzuje zpětnou vazbu od uživatelů. Některé náměty byly aplikovány již při tvorbě aplikací a několik jich zůstává navrženo pro budoucí vylepšení.

Klíčová slova: Android, Java, ReactJS, zvířecí útulek, Firebase, mobilní aplikace, webová aplikace

Abstract

The objective of this project is an application which will support animal shelters with advertising their animals. The thesis begins with analysis of current online animals advertising possibilities. It is followed by design of the solution which contains web application and mobile application for Android platform. Further on in this thesis presents description of data structure and used technologies. The main part of the thesis is about implementation of both applications. At the end there is a summary of user feedback and possible improvements.

Keywords: Android, Java, ReactJS, animal shelter, Firebase, mobile application, web application

Obsah

1. Úvod

Cílem této práce je podpořit adopci zvířat ze zvířecích útulků. Rozhodl jsem se vytvořit pro potřeby těchto útulků mobilní aplikaci určenou zájemcům o domácího mazlíčka. A také webovou aplikaci se stejnými funkcemi, rozšířenými o administrační rozhraní pro zvířecí útulky. Mobilní aplikace se stávají nedílnou součástí našeho života. Podle Českého statistického [1] úřadu používalo v roce 2020 chytrý telefon 72.6% obyvatel České republiky starších 16 let, což je ještě o 9,5% víc než v roce 2018.

Motivací pro zahájení tohoto projektu byla vlastní zkušenost při hledání domácích mazlíčků pro sebe a rodinu. Zjistil jsem, jak náročné to může být a kolik lidí zneužívá zvířata pro svůj finanční prospěch. Na toto zjištění navazuje první část této práce - rešerše aktuálních možností online inzerce zvířat. Rešerše ukázala, že důležitými faktory pro aplikaci jsou přehlednost a jednoduchost prohlížení zvířat z pohledu zájemce. Dále potvrdila že administrační rozhraní by mělo být také intuitivní a mělo by poskytovat možnost ukládat informace strukturovaně, aby je bylo možné využít k filtrování.

První částí řešení je webová aplikace. K návrhu jsem použil poznatky z rešerše, včetně připomínek zvířecích útulků. Aplikace umožňuje registraci a přihlašování. Registrace pro zvířecí útulky zahrnuje ověření přes registrační číslo útulku ze Státní veterinární správy [2]. Registrace pro zájemce není nutná, ale neregistrovaný uživatel má omezené funkce aplikace. Útulek má možnost vyplnit svůj profil včetně otevírací doby, obrázku a kontaktních údajů.

K přidávání zvířete do databáze slouží přehledný formulář. Útulek si může zobrazit všechna svá přidaná zvířata a jednotlivě je editovat nebo mazat. Zájemci mohou prohlížet zvířata ze všech útulků a filtrovat podle zvolených kritérií. Ve druhém způsobu prohlížení se zvířata uživateli zobrazují po jednom a uživatel je může hodnotit. V případě zájmu o mazlíčka může zájemce kontaktovat útulek pomocí zpráv přímo v aplikaci. Útulky tak mají přehled o komunikaci se všemi zájemci.

Druhou částí projektu je mobilní aplikace pro operační systém Android. Ta je určená především zájemcům o domácího mazlíčka. Zájemci mají stejné možnosti jako při použití webové aplikace. Mobilní aplikace také nabízí přehledné prohlížení mazlíčků a navíc může

uživatel nabízená zvířata řadit podle vzdálenosti. Hodnocení zvířat je v mobilní aplikaci inspirováno seznamovacími aplikacemi a mělo by přilákat i mladší generaci uživatelů.

V závěrečné části práce je popsáno testování aplikace a zpětná vazba vybraných zvířecích útulků.

2. Rešerše

V rámci rešerše jsem hledal aktuální možnosti online inzerce zvířat. Výsledkem byly čtyři způsoby, kterým jsou věnovány následující podkapitoly.

Dále jsem byl v kontaktu se 45 útulky z České republiky. Pomocí zaslané prezentace jsem jim představil grafický návrh aplikace včetně funkcí, které by měla nabízet. Od útulků jsem obdržel zpětnou vazbu včetně nápadů a připomínek, které jsem později využil při svém řešení.

2.1. Webové stránky jednotlivých útulků

Jednou z možností inzerce zvířat jsou webové stránky útulků. Některé útulky je ale vůbec nemají nebo je mají zastaralé.

Nevýhoda inzerce na vlastních stránkách je nižší počet návštěvníků. Zájemci často musí vyhledat konkrétní útulek, aby se na k informacím dostali. V případě, že zájemci nezáleží na vzdálenosti a je ochotný si pro zvíře dojet kamkoliv, je pro něj vyhledávání komplikované.

Pokud vyhledá například „útulek česká republika“, zobrazí se sice spoustu výsledků, ale je možné, že v tom množství zájemce přehlédne zrovna to, co hledá. Navíc s hledáním stráví zbytečně mnoho času.

Pokud bude vyhledávat po blízkých městech, tak se zase může stát, že opomene nějaký útulek v obci, kde ani nevěděl, že by mohl být. Z toho vyplývá, že možnost využívat weby konkrétních útulků je vhodná především pro ty, kteří zkouší své štěstí na konkrétním místě.

Určité nevýhody existují také pro útulky. Správa vlastních webů je finančně i časově náročná. Z tohoto důvodu spoustu útulků svůj web neprovozuje nebo neaktualizuje nabídku zvířat. Některé útulky proto raději inzerují na sociálních sítích.

2.2. Inzerce na sociálních sítích

Inzerce na sociálních sítích má své výhody. Jednou je jednoduchost a rychlost přidávání inzerátů a další, že je to pro obě strany zdarma (bez propagace). Útulky ale nemají celkový přehled o svých inzerátech a zájemcích. Přehled nemají ani zájemci, protože informace

o zvířatech nejsou moc přehledné a nemohou si inzeráty nijak filtrovat. Někdy není jasné, jestli je zvíře stále dostupné a koho v případě zájmu kontaktovat.

Další nevýhodou je, že zájemce musí mít účet na sociální síti. Problém je i s internetovými vyhledávači, které obvykle účty na sociálních sítích nenabízí mezi prvními výsledky, pokud uživatel nezná jejich název.

2.3. Inzertní servery

Inzertní servery jsou často využívaným nástrojem pro inzerování zvířat. Výhodou je inzerování zdarma. Vkládání inzerátů na těchto serverech je jednoduché a přehledné. Jsou ale určeny k inzerování věcí velkého množství kategorií a na zvířata nejsou specializované. Není možné podrobnější informace o zvířeti nijak strukturovat. Z toho plyne nevýhoda pro zájemce, kteří si nemohou inzeráty filtrovat. Je ale možné prohlížet zvířata z různých krajů na jednom místě.

Tyto servery jsou ale plné podvodných inzerátů. Objevují se inzeráty lidí, kteří se vydávají za majitele psů nebo za útulky a jen sbírají osobní údaje zájemců, nebo se snaží získat peníze za nic. Mnoho inzerátů je od tzv. množíren. Množírny se snaží vypadat jako chovatelé nebo útulky a chtějí oklamat důvěřivé zájemce. Bývá těžké mezi těmito inzeráty najít ty od skutečných útulků.

2.4. PESWEB

PESWEB je portál s informacemi ze světa psů. Obsahuje články o tématech spojených většinou se psy, ale i se zvířaty obecně. Dále poskytuje informace o různých akcích jako jsou výstavy, výcvikové akce, semináře a další. Na portálu mají také inzerci psů a koček k adopci.

Jelikož je portál zaměřen na psy, nabízí zájemcům podrobné filtrování psů podle jejich vlastností. Další výhodou je, že portál sjednocuje inzeráty z různých útulků.

Nevýhodou inzerce na tomto portálu je nepřehlednost. Portál je velmi obsáhlý a na každé stránce je tolik různých informací a bannerů, že je obtížné se v něm zorientovat.

2.5. Shrnutí

Během rešerše jsem prozkoumal různé způsoby online inzerce zvířat. Našel jsem výhody a nevýhody jednotlivých řešení. Tyto poznatky jsem využil při návrhu vlastní aplikace.

Vybral jsem vlastnosti, které by měly být součástí tohoto řešení. Důležitá je jednoduchost a přehlednost jak pro útulky, tak pro zájemce. Útulky musí mít k dispozici přehledné rozhraní pro přidávání a editaci inzerovaných zvířat. Vlastnosti zvířat musí být uchovávány strukturovaně a tomu musí být rozhraní přizpůsobeno. Díky tomu bude možné zvířata filtrovat. Stejně tak by mělo být uzpůsobeno rozhraní pro zájemce s možností snadného procházení a vyhledávání pomocí filtrů.

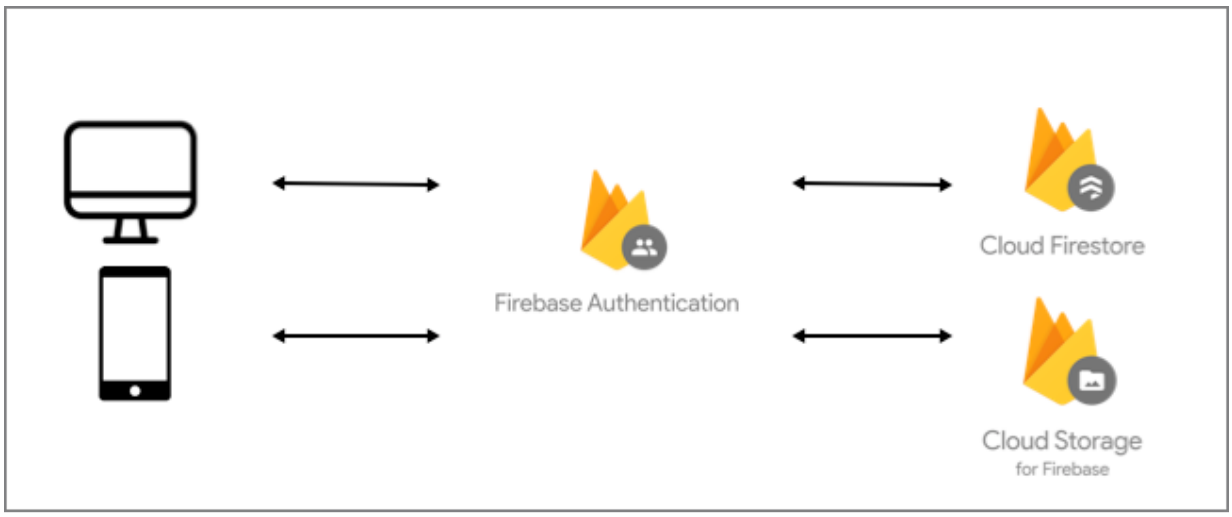
Velkým problémem některých řešení je velký počet podvodných inzerátů. Tomu bude bránit ověřená registrace útulků do systému. Aplikace by měla také bránit prodeji zvířat do nesprávných rukou nebo ho alespoň omezit. Proto bude pro adopci zvířete a některé funkce vyžadována registrace zájemce.

Aplikace bude zdarma včetně všech funkcí jak pro zájemce, tak pro útulky.

3. Struktura aplikace

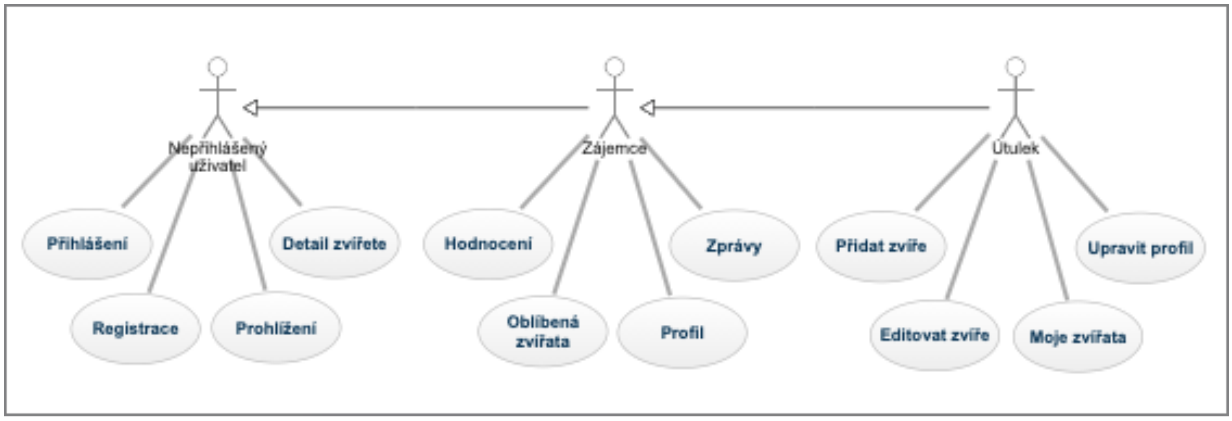
Aplikace je složená ze dvou hlavních částí - webová a mobilní. Webová aplikace obsahuje všechny funkcionality systému. Mobilní aplikace je pro platformu Android a některé funkce jsou zde omezené. Pro vkládání a editaci dat je primárně určena aplikace webová.

Aplikace využívá k ověřování uživatelů službu Firebase Authentication. K ukládání dat Cloud Firestore a obrázků Cloud Storage. Na obrázku 1 je zobrazeno schéma propojení aplikace a uvedených technologií.



Obrázek 1 - Schéma komunikace s Firebase

Na obrázku 2 je diagram užití aplikace. Jsou zde znázorněny tři aktéři (Nepřihlášený uživatel, Zájemce, Útulek) a jejich možné akce.



Obrázek 2 - Diagram případů užití

3.1. Datová struktura

V aplikaci jsou data rozdělena do čtyř entit (Zvíře, Uživatel, Konverzace, Zpráva, Recenze), které mají vlastní kolekce a pomocné struktury (Location), které jsou popsány v následujících kapitolách.

3.1.1. Animal (Zvíře)

Animal je hlavní entita aplikace, která obsahuje všechny informace o zvířeti. Každé zvíře má svůj unikátní identifikátor. Dále obsahuje identifikátor uživatele (útulku), ke kterému patří a další vlastnosti (jméno, věk, rasa, velikost, …). Jedna z vlastností je poloha zvířete, která je samostatná struktura popsaná níže. Atribut `images` obsahuje pole řetězců s cestami k souborům uloženým v úložišti Cloud Storage. `behaviorMap` obsahuje mapu, kde určitému chování zvířete je přiřazena hodnota `true` nebo `false`.

Animal	
ID	String
user	String
adopted	boolean
type	String
breed	String
name	String
size	String
weight	Integer
gender	String
age	Integer
desc	String
image	String
images	Array<String>
location	Location
behaviorMap	Map
created	Timestamp

Tabulka 1 - Animal

3.1.2. User (Uživatel)

User je entita definující jednoho uživatele aplikace. Obsahuje unikátní identifikátor, typ (zájemce, útulek) a další informace o uživateli. Některé vlastnosti má podle toho, jaký je typ. V tabulce jsou písmeny označeny vlastnosti, které se týkají buď pouze běžného uživatele (U), nebo pouze útulku (S). Conversations je pole identifikátorů konverzací.

User		
	ID	String
	type	String
	email	String
	name	String
U	surname	String
S	registrationNumber	String
S	phone	String
S	location	Location
	image	String
S	openingHours	Map
S	showOpenHours	boolean
	conversations	Array<String>
U	likedAnimals	Array<String>
U	skipedAnimals	Array<String>
S	reviews	Collection

Tabulka 2 - User

3.1.3. Conversation (Konverzace)

Conversation slouží k uchovávání konverzací mezi uživateli. Obsahuje vlastní ID a tři další atributy - ID dvou účastníků konverzace a kolekci zpráv, kde zpráva má vlastní entitu.

Conversation	
ID	String
user1	String
user2	String
messages	Collection

Tabulka 3 - Conversation

3.1.4. Message (Zpráva), Review (Recenze)

Message má své ID a uchovává odesílatele a příjemce zprávy, čas odeslání zprávy a samotný text zprávy. Review

Message	
ID	String
from	String
to	String
sent	Timestamp

Tabulka 4 - Message

Review	
ID	String
text	String
name	String
rating	integer
created	Timestamp

Tabulka 5 - Review

3.1.5. Location (Poloha)

Location je objekt platformy HERE, který obsahuje všechny dostupné informace o poloze zadané uživatelem. Důležité atributy pro aplikaci jsou `address, position`. a `title`. `address` obsahuje například ulici, město nebo PSČ. `position` obsahuje přesnou polohu ve formě souřadnic v desetinných stupních a `title` je souhrn adresy v jenom řetězci.

Location	
access	Array
address	Map
houseNumberType	String
id	String
mapView	Map
position	Map
resultType	String
scoring	Map
title	String

Tabulka 6 - Location

4. Použité technologie

4.1. Firebase

Firebase je platforma společnosti Google určená pro tvorbu mobilních a webových aplikací. Vznikla v roce 2011 a byla samostatná až do roku 2014, kdy ji koupila právě společnost Google [3]. První produkt platformy byla Firebase Realtime Database, který slouží k synchronizaci aplikačních dat Android, iOS a webových aplikací. Do roku 2014 vznikly ještě produkty Firebase Hosting a Firebase Authentication.

V následujících letech se vývoj zrychlil a nyní Firebase nabízí celkem 18 produktů [4]. Aplikace z nabízených produktů přímo využívá Cloud Firestore, Authentication, Cloud Storage a Hosting. K vývoji řešení a správě produktů byl využit nástroj Firebase Console.

4.1.1. Cloud Firestore

Cloud Firestore je NoSQL databáze a patří mezi hlavní produkty platformy Firebase. Je velmi rychlá a jednoduše přístupná v různých jazycích na různých platformách [5]. V mnoha směrech je Cloud Firestore nástupcem Realtime Database. Oproti zmiňované databázi je Firestore více a lépe strukturovaná [6].

Její struktura je tvořená dokumenty seskupenými do kolekcí. Dokument tvoří páry klíč - hodnota, kde klíč je řetězec. Hodnotami v dokumentů mohou být různé typy od integerů až po mapy. Dokument může také obsahovat kolekce obsahující další dokumenty s dalšími kolekcemi. Není třeba se ale bát dotazů na dokumenty, které obsahují hodně úrovní dalších dokumentů a kolekcí, jelikož je možné načíst dokument bez nutnosti načíst všechny kolekce v něm obsažené. Firestore tím dává možnost strukturovat data tak, jak je logicky vhodnější a bez obav ze stahování velkého množství nepotřebných dat.

Další výhodou je jednoduchost a přímočarost dotazování přes několik atributů včetně řazení či limitování pro stránkování.

Na data je možné se ptát jednorázově, kdy uživatel potřebuje ale i čekat na změny v databázi a aktualizovat data ihned po změně v databázi.

4.1.2. Authentication

Authentication je služba platformy Firebase, která poskytuje možnosti zabezpečení aplikace formou ověření uživatele. Nabízí spoustu metod ověření (E-mail - heslo, E-mail s odkazem, SMS) včetně ověření pomocí některých z hlavních poskytovatelů identity (Google, Facebook, Twitter, Apple, GitHub, …) [7]. Je možné využít funkci anonymního přihlášení. V této aplikaci jsem využil ověření pomocí e-mailu a hesla.

Authentication nabízí i předpřipravené uživatelské rozhraní pro přihlašování a registraci. Já jsem ale tuto možnost nevyužil, protože by rozhraní graficky nezapadalo do aplikace.

4.1.3. Cloud Storage

Cloud Storage je úložiště poskytované platformou Firebase. Slouží k uchovávání dat z aplikací [3]. Cloud Storage využívá klasický souborový systém a k datům lze přistupovat přímo z aplikace pomocí referencí, které ukazují, kde přesně je soubor v úložišti umístěn.

Nahrávání souborů do úložiště je také snadné. Vytvoří se reference na umístění souboru a soubor se nahraje například ve webové aplikaci standardně pomocí
`<input type="file" … />`.

4.1.4. Firebase Console

Firebase Console je nástroj, který usnadňuje vývojářům práci s Firebase. Lze v něm spravovat projekty a všechny služby Firebase, které jsou na projekty vázány. Pro využití služeb Firebase je nutné si v tomto nástroji vytvořit projekt a přiřadit k němu aplikace vybraných platforem.

Produkty Firebase se pak konkrétněji nastavují. Například ve Firebase Authentication se zde aktivují metody přihlašování uživatelů do aplikace. U Cloud Firestore a Cloud Storage může vývojář libovolně manipulovat s daty včetně jejich filtrování a nastavit pravidla pro přístup k databázi.

Je možné zde sledovat aktivitu aplikací a přístupy do databází a nakonec případně změnit finanční plán.

4.1.5. Zabezpečení

Firebase používá vlastní zabezpečení databáze Security Rules. Jedná se o pravidla, podle kterých Firebase umožňuje uživatelům přistupovat k databázi [8]. K zabezpečení se používá kombinace těchto pravidel a Firebase Authentication. Na základě nastavení pravidel a autentizace Firebase uděluje přístup do databáze.

Pravidla se nastavují ve Firebase Console, kde se zapisují ve formátu podobnému JSON. Je možné určit pravidla pro celou databázi nebo úložiště, ale i zvlášť pro kolekce nebo konkrétní dokumenty a soubory tak, že se nastaví pravidlo pro konkrétní cestu. Pravidlo může omezovat přístup pro uživatele nebo skupiny uživatelů. Na obrázku 3 je nastaven přístup ke čtení a zápisu pro všechny přihlášené uživatele aplikace.

```
1  rules_version = '2';
2  service cloud.firestore {
3    match /databases/{database}/documents {
4      match /{document=**} {
5        allow read, write: if request.auth != null;
6      }
7    }
8  }
```

Obrázek 3 - Pravidla zabezpečení Firebase

4.2. React

React je JS open-source knihovna od společnosti Facebook pro tvorbu uživatelského rozhraní (UI). Na rozdíl od různých kompletních frameworků (např. Angular) se tedy soustředí na jednu specifickou oblast a pokud se na ní podíváme z hlediska klasické MVC architektury, tvoří právě a pouze view, tedy vrstvu pohledu, která prezentuje data uživateli [9]. Knihovna je vhodná pro tvorbu single-page i mobilních aplikací.

Kód React aplikací je členěný na komponenty, které je možné vykreslit v nějakém HTML prvku. Komponenty mohou obsahovat jednu nebo více dalších komponent a tvořit stromovou strukturu. Při vytváření komponenty jí můžeme předat parametry označené jako `props`. Komponenty obsahují stavové proměnné, jejichž hodnota když se změní, tak se celá

komponenta vykreslí znovu. Stavové proměnné se inicializují v konstruktoru komponenty a dále se mohou měnit jako reakce na podnět od uživatele.

Komponenty jsou většinou psány syntaxí JSX, což je rozšířená syntaxe JavaScriptu připomínající formát HTML. JSX syntaxi není nutné využívat, ale většina vývojářů tak dělá. Umožňuje nám spojit vykreslování s logikou aplikace, které pak není potřeba rozdělovat do více souborů. Každá komponenta pak obsahuje metodu `render()`, ve které se vykresluje její obsah.

React komponenta má svůj životní cyklus (viz Obrázek 4) a k němu jsou vázány i další metody kromě metody `render()`. Zde jsou metody využívané v tomto řešení.

`constructor()` - Volá se ještě před vykreslením komponenty. V konstruktoru se inicializuje lokální stav, tak že do `this.state` přiřadíme objekt s výchozími stavy.

`componentDidUpdate()` - Metoda, která se volá automaticky hned po vykreslení komponenty, po metodě render(). Využívá se obvykle pro volání API, načítání dat z databáze.

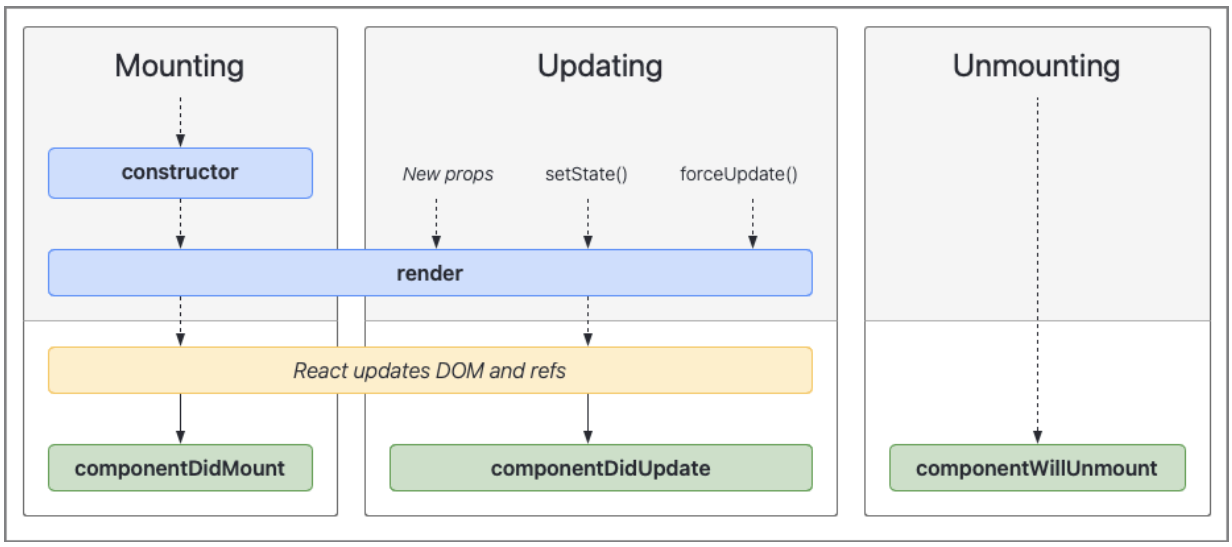
`componentWillUnmount()` - Tato metoda se volá na konci životního cyklu komponenty. Ukončují se v ní časovače, observery a všechny akce které by se samy neukončily při zničení komponenty.

`componentDidUpdate()` - Metoda zavolaná pokaždé, když se komponenta nějak změní (změní se stavová proměnná, komponenta se vykreslí). Je vhodná pro sledování hodnot v `props`, uvnitř je možné porovnat aktuální hodnoty s předchozími. V případě změny například volat API, bez změny naopak nedělat nic. Měnit stavové proměnné v této metodě se musí pouze pod nějakou podmínkou, jinak by vznikla nekonečná smyčka.

4.3. Figma

Figma je cloudový designový nástroj. Je to aktuálně nejpoužívanější designový nástroj na světě a to i kvůli tomu, že vznikl jako webová aplikace a je proto nezávislý na platformě. Figma nabízí i desktopové aplikace a možnost pracovat offline, avšak je na internetovém připojení dost závislá a práce offline přináší problémy.

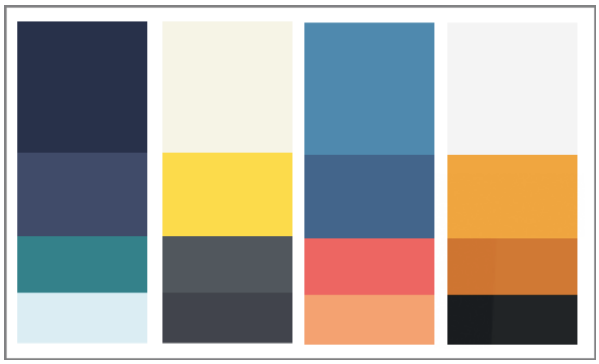
Jednou z výhod, která ale nebyla v této práci využita, je možnost týmové spolupráce, která má zde spoustu vylepšení a zjednodušení oproti ostatním nástrojům. Například podobně jako v dokumentech Google může více lidí pracovat na stejném souboru v reálném čase [10].



Obrázek 4 - Metody životního cyklu

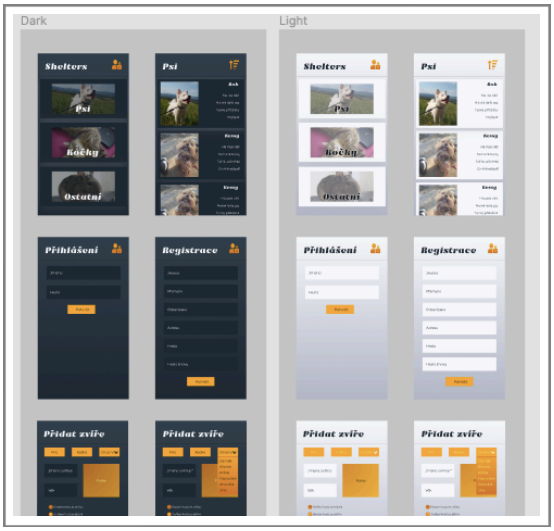
5. Grafický návrh

Při plánování vzhledu aplikace jsem začal volbou barevné palety aplikace. Nejprve jsem se snažil barevnou paletu vytvořit sám. Jelikož ale nemám dostatečné grafické cítění, nedařilo se mi barvy k sobě sladit. K výběru jsem proto využil kolekci barev ze stránky ColorHunt.co. Cílem bylo najít paletu, kde bude většina barev neutrální a jedna výrazná pro upoutání pozornosti. Na obrázku 5 jsou vidět palety, které jsem postupně zvažoval. Poslední z nich je vítězná a je využitá pro světlý i tmavý mód.



Obrázek 5 - Barevné palety

K návrhu grafiky jednotlivých stránek jsem využil nástroj Figma (viz Obrázek 6). Výhodou nástroje je intuitivní a snadné ovládání. Stránky jsou přehledně roztříděné do vrstev. K jednotlivým grafickým prvkům lze přistupovat přímo v náhledu nebo ve stromové struktuře vrstev. Vrstvy se dají snadno kopírovat a měnit. Tím je vytváření návrhu rychlé a prvky jsou jednotné.



Obrázek 6 - Ukázka návrhu v nástroji Figma

6. Webová aplikace

V této kapitole bude popsána implementace řešení webové aplikace, která je jednou ze dvou částí této práce.

6.1. Struktura aplikace

Webová aplikace v tomto projektu je tzv. single-page aplikace. Je složená z komponent, které tvoří jednotlivé stránky aplikace. O zobrazování stránek se stará komponenta `BrowserRouter` z knihovny `react-router-dom`. Slouží jako navigace. Ta se nachází na vrchu stromové struktury komponent. O úroveň níž je komponenta `App`, hlavní komponenta aplikace. Následující kapitoly budou detailně rozebírat jednotlivé komponenty.

6.1.1. Komponenta App

Obsah této komponenty je obalen komponentou `Layout`, která kromě hlavního obsahu stránek zajišťuje zobrazení vrchního menu. Zároveň udržuje jednotný styl pro všechny stránky aplikace. Uvnitř se nechází komponenta `Switch`, která se stará o zobrazování správných komponent podle aktuální URL adresy. Jednotlivé cesty jsou reprezentovány buď pomocí výchozí komponenty `Route`, nebo vlastní komponenty `PrivateRoute`. `PrivateRoute` jsem použil v případě, kdy je zobrazení požadované komponenty podmíněno přihlášením.

6.1.2. Komponenta PrivateRoute

Tato komponenta rozšiřuje komponentu `Route`. Kontroluje zda je uživatel přihlášen funkcí `onAuthStateChanged()` služby Firebase Authentication, která se volá v metodě `componentDidMout()`. Stav přihlášení uživatele se zjišťuje asynchronně, a proto je nutné hlídat, jestli už přišla aktuální informace a nedošlo k přesměrování uživatele moc brzo. Kromě stavové proměnné, která drží stav přihlášení je použita ještě stavová proměnná, která nabude hodnoty `true` až po vrácení hodnoty z asynchronní operace.

Když je uživatel přihlášen, zobrazí se požadovaná komponenta, která byla předána v `props`. Pokud uživatel není přihlášen, zobrazí se komponenta `Login`. Komponentě `Login` je ještě předána informace, kam nebyl uživatel bez přihlášení vpuštěn.

6.1.3. Komponenta Home

Komponenta Home reprezentuje domovskou stránku aplikace. Na domovské strace jsou zobrazeny komponenty pro psy, kočky a ostatní zvířata (viz Obrázek 32). Kliknutím na jednu z nich se uživatel může rychle dostat na prohlížení mazlíčků zvoleného druhu.

6.1.4. Komponenta Login

Na přihlašovací stránce vidí uživatel klasický přihlašovací formulář (viz Obrázek 33). K přihlášení je vyžadován e-mail a heslo. V případě zapomenutého hesla má uživatel možnost heslo resetovat.

Po kliknutí na potvrzovací tlačítko se zavolá funkce `login()`. Na začátku funkce proběhnou základní kontroly vstupů. O většinu validace se stará sama přihlašovací funkce služby Authentication - `signInWithEmailAndPassword(email, password)` (viz Obrázek 7). V případě úspěchu je uživatel přesměrován. Cíl přesměrování je určen podle hodnoty v proměnné `this.props.location.state.from`. Ta je naplněna v komponentě `PrivateRoute`. Pokud je prázdná, uživatel je přesměrován na komponentu `Home`. V případě neúspěchu validace, vrací přihlašovací funkce chybu. Podle kódu chyby se uživateli zobrazí příslušná chybová hláška. Například při nesprávném heslu přijde chyba s kódem `"auth/wrong-password"`.

```
signInWithEmailAndPassword(email, password) Promise<UserCredential>
then((userCredential :UserCredential ) => {
    let from;
    if (this.props.location.state) {
        from = this.props.location.state.from || {from: {pathname:
    } else {
        from = {pathname: '/'}
    }
    this.props.history.push(from);
) Promise<UserCredential>
```

Obrázek 7 - Přihlašovací funkce Firebase Authentication

Po kliknutí na odkaz „Zapomenuté heslo“ se objeví vyskakovací okno s jedním textovým vstupem pro zadání e-mailu, na který má přijít odkaz pro obnovení hesla (viz Obrázek 34). Pro resetování hesla se používá funkce služby Authentication `sendPasswordResetEmail(email)`. Funkce zajistí validaci e-mailu, kontrolu existence uživatele a odeslání e-mailu s odkazem pro resetování hesla. V případě chyby je zobrazena příslušná chybová hláška. V opačném případě se zobrazí pozitivní hláška a zavře dialogové okno.

6.1.5. Komponenta Registration

Vzhled této komponenty je podobný komponentě `Login`. Navíc má uživatel možnost zvolit, jestli se registruje jako běžný uživatel (zájemce), nebo útulek (viz Obrázek 58). Podle jeho volby se zobrazují pole ve formuláři.

Z funkčního hlediska se v komponentě `Registration` řeší především validace vstupních dat (viz Obrázek 59). K tomu slouží funkce `validate()`. Kontroluje se, jestli jsou vyplněná všechna povinná pole, jestli jsou zadaná hesla shodná. V případě útulku se navíc kontroluje registrační číslo, které musí být v seznamu registrovaných útulků pro zvířata na stránkách státní veterinární správy. Zbylé kontroly (podmínky a správnost hesla, platnost e-mailu, již existující uživatel) kontroluje registrační funkce služby Firebase Authentication `createUserWithEmailAndPassword(email, password)`. Pokud funkce obdrží platnou e-mailovou adresu, která není obsazena a heslo splňující podmínky, vytvoří objekt uživatele a přiřadí ho k projektu ve Firebase. Tento objekt uživatele slouží k přihlašování do aplikace. Z dat zadaných uživatelem do formuláře obsahuje objekt jen e-mail a heslo. Po jeho vytvoření se proto pod jeho `uid` vytvoří záznam v databázi Cloud Firestore, kde jsou uložena všechna zbylá data.

Do databáze se uživatel přidá ve funkci `addUserToFirestore()`, kde se komunikuje s databází.

6.1.6. Komponenta Animals

Komponenta `Animals` je jedna z hlavních komponent aplikace. Komponenta je využita na třech stránkách aplikace. Na stránce „Prohlížení“ (viz Obrázek 35) zobrazuje všechna dosud

neadoptovaná zvířata. Na stránce „Oblíbená zvířata“ zobrazuje zvířata, která si přihlášený uživatel přidal mezi oblíbená. Na stránce „Naše zvířata“ (viz Obrázek 49) jsou zvířata přidaná přihlášeným uživatelem (útulkem). Zvířata jsou zobrazena ve formě kartiček (komponenta `AnimalCard`) uspořádaných do mřížky. Uživatel má možnost data filtrovat a vyhledávat podle čísla čipu.

V metodě `componentDidMount()` se volá funkce pro načtení výchozích dat z databáze. Funkce volá databázi s parametry podle toho jestli je uživatel na stránce prohlížení zvířat, oblíbených zvířat, nebo vlastních nabízených zvířat (viz Obrázek 8). Záznamy jsou řazeny podle data přidání do databáze a je zohledněn je `adopted`, říkající jestli bylo zvíře již adoptováno.

```
loadData(likedAnimals) {
    if (this.props.myAnimals) {
        const task = db.collection( collectionPath: "animals") ...
            .where( fieldPath: "user",  opStr: "==", auth.currentUser.uid) Query<DocumentData>
            .orderBy( fieldPath: "adopted",  directionStr: "asc").orderBy( fieldPath: "created",  directionStr: "desc");
        this.sendTask(task);
    } else if (this.props.likedAnimals) {
        if(likedAnimals.length > 0) {
            const task = db.collection( collectionPath: "animals").
            where(fieldPath.documentId(),  opStr: "in", likedAnimals);
            this.sendTask(task);
        }
    } else {
        const task = db.collection( collectionPath: "animals") ...
            .where( fieldPath: "adopted",  opStr: "!=",  value: true) Query<DocumentData>
            .orderBy( fieldPath: "adopted",  directionStr: "asc").orderBy( fieldPath: "created",  directionStr: "desc");
        this.sendTask(task);
    }
}
```

Obrázek 8 - Vykreslení komponent AnimalCard

Data se uloží do dvou seznamů. Jeden seznam se udržuje nezměněný a obsah druhého se mění při filtrování a zobrazuje se. S filtrováním pomáhá komponenta `Filter`. Ta filtr zobrazí a potvrzením filtru zavolá funkci v komponentě `Animals` a předá filtrovací parametry zvolené uživatelem. Ve funkci `filter(filterData)` se pomocí vybraných filtrů vyfiltrují data z neměnného seznamu zvířat a uloží do zobrazovaného seznamu.

Po kliknutí na kartu zvířete se zobrazí jeho detail - komponenta `AnimalDetail`.

6.1.7. Komponenta Swipe

`Swipe` je druhá komponenta, která zobrazuje zvířata. Tady jsou ale představována zábavnější formou. Zobrazují se po jednom a uživatel má možnost přidat si zvíře do oblíbených nebo ho přeskočit (viz Obrázek 42). Po přidání nebo přeskočení se zobrazí další náhodně vybrané zvíře.

Zvíře zobrazuje komponenta `AnimalCard` s parametry pro užití v komponentě `Swipe`. Po kliknutí na kartu se zobrazí komponenta `AnimalDetail`. Zvířata se filtrují stejně jako v komponentě `Animals` s použitím komponenty `Filter`.

Načítání výchozích dat zohledňuje seznamy zvířat, která přihlášený uživatel již hodnotil. Zvíře, které už má uživatel mezi oblíbenými, se ve výběru už neobjeví. Zvíře, které přeskočil se znovu objeví až když už nezbyla žádná neohodnocená zvířata.

Kliknutím na hodnocení zvířete se zavolá funkce `swipe(list)`. Jako parametr se funkci předá název listu (likedAnimals/skipedAnimals). Funkce přidá ID ohodnoceného zvířete do databáze (viz Obrázek 9). Každý uživatel má v databázi dvě pole, jedno pro oblíbená a jedno pro přeskočená zvířata.

```
const userRef = db.collection( collectionPath: "users").doc(this.state.currentUser.uid);
userRef.update( data: {
    [list]: firebase.firestore.FieldValue.arrayUnion(this.state.animal.id)
}).then(() => {
    this.setState( state: {animalDocuments: array, animal: animal});
}).catch(e => {
    console.log("error " + e.message);
})
```

Obrázek 9 - Přidání položky do pole v databázi

6.1.8. Komponenta AddAnimal

Komponenta určená pro útulky. Slouží k přidávání zvířat do databáze a jejich editaci.

Komponentu tvoří formulář (viz Obrázek 44), který obsahuje pole pokrývající všechny vlastnosti zvířete. První informací je typ zvířete (Pes, Kočka, Ostatní). Podle zvoleného typu se mění obsah skupiny zaškrtávacích polí s vlastnostmi zvířete. Tyto vlastnosti se týkají jeho chování. Skupiny vlastností pro každý typ zvířete jsou uloženy v jednotlivých polích

v souboru Const.js. Pole jsou sjednocena do mapy, kde jako klíč je typ zvířete a hodnotou je pole vlastností. Z této mapy je v komponentě vytvořena mapa `<String, boolean>` jako stavová proměnná s vlastnostmi a hodnotami příslušných zaškrtávacích polí.

Jméno, věk, hmotnost, rasa a adresa jsou textová pole `<input type="text" … />`, popis `<textarea …/>`. Pohlaví se přepíná komponentou `Switch` z knihovny `material-ui`.

Po kliknutí na pole pro výběr rasy se pod ním objeví nabídka ras načtená z JSON souboru. Psaním do pole se nabídka filtruje a uživatel může jednoduše rasu najít a vybrat.

V poli s adresou je výchozí hodnota adresa přihlášeného útulku. Pro zadávání adresy jsem využil HERE Geocoding and Search API. Toto API jsem vybral především kvůli finančnímu plánu, protože tato služba je do určitého počtu požadavků zdarma. Při každé změně v textovém poli pro adresu se volá funkce `handleSearchChange(e)`, ve které se do stavové proměnné ukládá jeho hodnota. Když je délka zadaného řetězce vyšší než jeden znak, volá se funkce `search()`. Kromě délky řetězce se ještě časovačem kontroluje, jestli uživatel stále píše (500ms), aby se funkce a následně API nevolalo zbytečně několikrát během jednoho slova (viz Obrázek 10).

```
handleSearchChange = (e) => {
    clearTimeout(this.timer);
    this.setState( state: {address: e.target.value});
    if (e.target.value.length >= 2) {
        this.timer = setTimeout(this.search, WAIT_INTERVAL);
    }
}
```

Obrázek 10 - Časovač volání API

Ve funkci `search()` se už pracuje přímo s API (viz Obrázek 11). Funkci API `geocode()` se jako parametr předává objekt s hledaným řetězcem a kódy zemí, kde má hledat. Pro účely aplikace jsem nastavil hledání na Českou Republiku a sousední státy. Výsledek hledání jsou objekty s informacemi o vyhledané adrese. Ty se uloží do stavové proměnné jako pole a dynamicky se vykreslují jako seznam pod vyhledávací pole. Kliknutím na položku se vyhledávací pole vyplní názvem adresy a celý objekt se uloží do stavové proměnné.

```
search = () => {
    searchService.geocode({
        q: this.state.address,
        in: "countryCode:CZE,SVK,DEU,POL,AUT"
    }, (result) => {
        console.log(result.items);
        this.setState( state: {searchResults: result.items, showSearchResults: true})
    }, (error) => {
        console.log("Error", error);
    });
}
```

Obrázek 11 - Volání funkce HERE Maps API

V poslední řadě se v této komponentě přidávají obrázky do galerie zvířete. Obrázky se přidávají po jednom a hned se v komponentě zobrazují jejich miniatury. Hlavní obrázek je zobrazen ve větší velikosti. Po kliknutí na miniaturu se vybraný obrázek nastaví jako hlavní.

Obrázek se nahrává pomocí `<input type="file" … />`. Z vloženého souboru se přidá do jednoho pole samotný soubor a do druhého pole jeho URL. Přidáním URL do stavového pole se znovu načte komponenta. V metodě `render()` se adresy vloží do HTML prvků `<img src="URL" … />` i s přidanou miniaturou nového obrázku. Jako hlavní obrázek je nastaven ten na první pozici v poli. Výměna hlavního obrázku kliknutím na jiný probíhá výměnou jejich pozic.

Uložení zvířete do databáze je rozděleno na dvě části. První částí je uložení obrázků do Cloud Storage. Proces se zahájí potvrzením formuláře a zavoláním funkce `handleUpload()`. Funkce postupně prochází obrázky. Pro každý vytvoří složením jména zvířete a UUID budoucí cestu souboru v úložišti. Do úložiště se se soubory nahrávají funkcí `put(soubor)`. Ta se volá z reference `firebase.storage.Reference` a vrací objekt Promise - příslib výsledku asynchronní operace. Když se obrázek nahraje, uloží se jeho cesta. Každý obrázek má vlastní asynchronní operaci a na konci je potřeba vědět, kdy už jsou všechny operace hotové. Proto se všechny přísliby ukládají do pole a funkce `Promise.allSettled<[]>` hlídá dokončení všech operací. Po jejich dokončení začne druhá část zavoláním funkce `addAnimal()`.

Data ze všech polí formuláře jsou uložena ve stavu komponenty jako objekt. Ve funkci `addAnimal()` se do objektu přidá pole obrázků a objekt se nahraje do databáze.

Ve funkci `componentDidMount()` kontroluje komponenta existenci proměnné `this.props.location.state.animal`. Pokud jde o editaci (viz Obrázek 45), proměnná obsahuje data editovaného zvířete. Data se uloží do stavové proměnné `this.state.animal` a tím se zobrazí v polích formuláře. Speciálním případem je galerie zvířete. Pokud má zvíře nějaké obrázky, musí se načíst z úložiště. Aplikace najde obrázky v úložišti podle cest uložených v databázi. Ke každému obrázku získá jeho URL a tu přidá do pole adres, do kterého se potom přidávají i adresy nově nahraných obrázků.

Kromě uložení dat zvířete do databáze má uživatel možnost zvíře smazat nebo ho označit jako adoptované.

6.1.9. Komponenta Profile

Komponenta uživatelského profilu funguje ve dvou režimech - režim zobrazení a režim úprav (viz Obrázky 38 a 39). Výchozí stav je režim zobrazení. Vzhled komponenty se liší také podle typu přihlášeného uživatele (viz Obrázek 41). Profil útulku je rozšířený oproti profilu zájemce. V této kapitole budu popisovat pouze rozšířenou verzi. V režimu zobrazení jsou zobrazeny informace o útulku, obrázek útulku a adresa, která je i vyznačena na mapě. Jsou zde také uživatelské recenze.

Vykreslení komponenty závisí na zvoleném režimu. Do režimu úprav lze přejít tlačítkem „Upravit profil“. Vykreslení tlačítka je podmíněno rovností ID přihlášeného uživatele a ID uživatele zobrazeného profilu. Informace o uživateli jsou uloženy ve stavové proměnné `this.state.user`. V metodě `render()` jsou zobrazeny v HTML prvcích `<div …/>`. Po změně na režim úprav se vymění za vstupy `<input … />` s výchozími hodnotami.

Funkce pro načtení dat se volá v metodě `componentDidMount()`. Daty z databáze se naplní stavová proměnná `this.state.user`. Pokud má uživatel obrázek, má uloženou v databázi jeho cestu v úložišti Cloud Storage. Z cesty se následně vytvoří reference a metodou `getDownloadURL()` získá adresa obrázku. Ta je určená k zobrazení obrázku v prvku `<img … />`.

Pro vyznačení polohy útulku na mapě jsem využil mapu platformy HERE Maps. Obecně bych zvolil Google Maps, ale chtěl jsem, aby byly polohové služby v aplikaci sjednocené. Útulek má v databázi uložený objekt Location. Z toho objektu se v tomto případě využijí souřadnice, které se předají mapě. Další parametr mapy je zoom. Tím se nastavuje výchozí přiblížení mapy. Aby bylo možné se v mapě pohybovat nebo pracovat se zoomem, je nutné nastavit její reakce na události. Využil jsem základní nastavení, dostačující pro základní prohlížení mapy. Nakonec se vytvoří značka na poloze útulku a přidá do mapy. Značkou je objekt `Marker`, který má v konstruktoru jako parametr souřadnice.

V režimu úprav jsou dvě hlavní tlačítka - „Uložit“ a „Zpět“. Kliknutím na „Zpět“ se přepne režim a do stavových proměnných se načtou původní data z databáze a tím zruší všechny změny. Útulek má možnost editovat všechna data kromě e-mailové adresy. Navíc může přidat otevírací dobu. Pro zadávání otevírací doby jsem použil knihovnu `material-ui`, konkrétně prvek `<TextField type="time" />`. Otevírací doba je ve výchozím stavu zablokovaná. Pro aplikaci to znamená, že v režimu zobrazení je skrytá. V režimu úprav jsou zablokovány úpravy a pro přehlednost je zvýšena neprůhlednost - `opacity: 0.5`. Otevírací doba se aktivuje a deaktivuje zaškrtávacím polem.

Uživatel může mít jeden obrázek. Ten lze přidat stisknutím tlačítka „Nahrát obrázek“. Po výběru obrázku z uživatelova zařízení se zavolá funkce, ve které se funkcí `URL.getObjectURL()` vytvoří URL souboru a spolu se souborem se uloží do stavových proměnných (viz Obrázek 12). Pomocí URL se obrázek ihned zobrazí v prvku `<img src="URL" />`.

```
loadPhoto = (image) => {
    const ref = storage.ref();
    const imageRef = ref.child(image);
    return imageRef.getDownloadURL() Promise<any>
        .then((url) => {
            this.setState( state: {photoUrl: url});
        }) Promise<any>
        .catch(function (error) {
            console.log("Error " + error.message);
        });
}
```

Obrázek 12 - Získání URL obrázku v úložišti

Adresa se zadává do textového pole. Pomocí Geocoding and Search API platformy HERE Maps jsou uživateli nabízeny možnosti adres podle zadaného řetězce. Řešení podrobněji popsáno v kapitole 6.1.8. Po kliknutí na adresu se adresa uloží do stavové proměnné. Ukládání hodnot do stavových proměnných metodou `this.setState()` je asynchronní operace. Jako callback je zavolána funkce `moveMap()`. V této funkci se střed mapy přesune na nové souřadnice. Poté se vymažou z mapy všechny objekty, v tomto případě značka předchozí polohy. Nakonec se vytvoří nová značka na nové poloze.

Kliknutím na tlačítko „Uložit“ se zavolá funkce `handleUpload()`, která se stará o nahrání nového obrázku do úložiště Cloud Storage. K nahrání souboru slouží funkce `storage.ref(imageStoragePath).put(this.state.file)`, která vrací objekt umožňující sledovat průběh nahrávání. Objekt `firebase.storage.Storage.ref` je reference na soubor v úložišti. K jeho vytvoření je potřeba cesta souboru. Ta je vytvořena z názvu složky v úložišti, ID uživatele a UUID. Druhou část ukládání zvířete obstará funkce `save()`. Ta se zavolá po úspěšném nahrání obrázku nebo ihned, pokud není vybrán žádný obrázek.

V této funkci se do databáze data vkládají nebo se upravují. To se dělá funkcí `update({})` reference dokumentu uživatele. Jako parametr se vkládají data ze stavových proměnných jako jeden objekt. Vkládání dat je podmínkou rozděleno na dva případy, útulek a zájemce, protože každý má jiné atributy. Po vykonání operace se změní režim komponenty `addAnimal()` zpátky na režim zobrazení.

Při návštěvě profilu útulku může zájemce přidat recenzi (viz Obrázky 40 a 48). Recenze obsahuje text, hodnocení, čas přidání a jméno uživatele.

6.1.10. Messages

Komponenta Messages slouží ke komunikaci mezi útulky a zájemci. V komponentě je zobrazen seznam konverzací a okno s otevřenou konverzací (viz Obrázek 47). Uživatel si vybírá aktivní konverzaci kliknutím na jednu ze zobrazených. Konverzace i okno mají vlastní komponentu.

Konverzace zobrazuje komponenta `Conversations`, které se předá ID přihlášeného uživatele, pole jeho konverzací a funkce `onConversationClick()`. Zmíněná funkce se volá z komponenty `Conversations` a uloží se v ní ID aktivní konverzace do stavové proměnné. Vedle komponenty s konverzacemi je komponenta `Chat`, která se objeví pokud je v proměnné `this.state.activeConId` nějaká hodnota. Této komponentě se předává ID aktivní konverzace a ID přihlášeného uživatele.

V metodě `componentDidMount()` je podmínka kontrolující obsah proměnné `this.state.location.state`. Proměnná je existuje pouze v případě, pokud uživatel otevřel komponentu kliknutím na „Kontaktovat“ v detailu zvířete. Je v ní uložen objekt vybraného zvířete, který obsahuje ID jeho útulku. Pokud nastane tato situace, funkce se dotazuje databáze, zda existuje konverzace obsahující ID přihlášeného uživatele a ID útulku.

Když konverzace existuje, uloží se její ID jako ID aktivní konverzace a vloží se jako parametr komponentám `Chat` a `Conversations`. Pokud neexistuje, vytvoří se v databázi nová a přidá se do pole konverzací předaného komponentě `Conversations`. Stejně jako v předchozím případě se její ID předá komponentám jako ID aktivní konverzace.

6.1.11. Pomocné komponenty

6.1.11.1. Conversations

Tato komponenta slouží k zobrazení konverzací v komponentě `Messages`. Od rodičovské komponenty obdrží ID přihlášeného uživatele. V metodě `componentDidUpdate()` se volá funkce pro načtení konverzací z databáze. Volání funkce je podmíněné změnou `props`, konkrétně `props.conversations`. Z databáze se načtou všechny konverzace, které mají ID v poli `this.props.conversations`. Každá konverzace má v sobě uložené dva účastníky a do pole `userIds` se uloží ID toho druhého, než je přihlášený.

Po načtení všech konverzací se zavolá metoda `loadUsers(userIds)`, ve které se načtou uživatelé patřící ke konverzacím. Díky tomu je ke každé konverzaci přiřazen uživatel a může se vypsat například jeho jméno.

6.1.11.2. Chat

Tato komponenta zobrazuje aktivní konverzaci v komponentě Messages. Od rodičovské komponenty převezme ID konverzace, podle kterého ji najde v databázi. Konverzace uchovává kromě ID dvou účastníků ještě kolekci zpráv. V metodě `fetchMessages()` zavolané v metodě `componentDidMount()` se načtou všechny zprávy z této kolekce.

Cloud Firestore umožňuje získávat data v reálném čase a funkce kromě prvního načtení dat čeká na změny v kolekci po celou dobu existence komponenty. Každá změna spustí akci, ve které se prochází všechny změněné dokumenty. Změna je reprezentována objektem `DocumentChange`. Ten obsahuje dokument, kterého se změna týká a typ změny. Pro tuto komponentu je důležitý typ změny `„added“`, což znamená že dokument je nový. Když je změna tohoto typu, přidá se nový dokument (zpráva) do pole zpráv. Načtené zprávy jsou uloženy do stavové proměnné, aby se s každou změnou překreslila komponenta a změna se projevila.

Zprávy jsou získávány z databáze seřazené podle data odeslání, a v metodě `render()` vykresleny. Podle ID přihlášeného uživatele se jeho zprávy vzhledově odlišují od příchozích a zobrazují se na opačné straně okna konverzace.

Novou zprávu uživatel píše do textového pole, jehož obsah se rovnou ukládá do stavové proměnné `this.state.newMessage`. Odeslat zprávu lze kliknutím na tlačítko nebo stisknutím klávesy „Enter“. Odesláním se spustí funkce `sendMessage()`, která nejprve zkontroluje, jestli zpráva není prázdná. Následně se zpráva zapíše do databáze společně s ID odesílatele, ID příjemce a časem odeslání.

6.1.11.3. AnimalCard

Tato komponenta slouží k zobrazení jednoho zvířete. Používá se vždy v komponentách `Animals`, `Swipe`, nebo `AnimalDetail`. Pro každou z těchto komponent má variantu zobrazení.

Komponenta `AnimalCard` přijme v `props` objekt zvířete a variantu zobrazení. V metodě `componentDidMount()` se zavolá funkce pro načtení obrázku z úložiště. Pokud se jedná o variantu pro komponentu `AnimalDetail` (viz Obrázek 37) a zvíře má galerii (existuje

proměnná `this.props.gallery`), zavolá se funkce pro načtení všech obrázků z úložiště. Ve funkci se prochází cesty ke všem obrázkům, vytváří reference na objekty v úložišti a získávají se jejich URL funkcí `getDownloadURL()`. Funkce vrací příslib `Promise`, který se přidá do pole. Po uložení všech příslibů do pole se zavolá funkce `Promise.allSettled(Promise[])`, která čeká až skončí všechny asynchronní operace v poli v parametru.

Kliknutím na komponentu se zavolá funkce `opendetail()` rodičovské komponenty, která byla této komponentě předána v `props`.

Nejmenší varianta komponenty `AnimalCard` je určena pro komponentu `Animals`. Kromě obrázku obsahuje ještě jméno zvířete, věk, a polohu. Tato varianta má alternativní vzhled, pokud je zvíře již adoptováno (viz Obrázek 46).

V komponentě `Swipe` se karta liší především názvy tříd, kvůli stylování. Je větší a obsahuje navíc informaci o velikosti zvířete.

Nejobsáhlejší varianta karty je v komponentě `AnimalDetail`. V této variantě jsou zobrazeny všechny dostupné informace o zvířeti a obrázky. Pokud má zvíře více obrázků, jsou zobrazeny v galerii. Pro galerii jsem využil komponentu `ImageGallery` z knihovny `react-image-gallery`. Té se předá pole objektů reprezentujících obrázky. Každý objekt obsahuje URL obrázku a URL miniatury. Komponentě jsem ještě upravil styly, aby zapadala do vzhledu aplikace.

V této variantě a v případě že přihlášený uživatel je útulek, je navíc tlačítko „Editovat“. Po kliknutí na tlačítko se nastaví stavová proměnná `this.state.editAnimal`. V metodě `render()` se vrátí místo normálního obsahu komponenta `<Redirect />` z knihovny `react-router-dom`. Ta přesměruje uživatele na komponentu `AddAnimal` (viz Obrázek 13). Pokud je přihlášený uživatel zájemce, jsou zobrazeny tlačítka „Kontaktovat“ a „Profil útulku“. Po kliknutí na „Kontaktovat“ se stejným způsobem uživatel přesměruje na komponentu `Messages`, kde se zahájí konverzace s útulkem, který nabízí vybrané zvíře. Kliknutím na „Profil útulku“ je uživatel přesměrován na jeho profil (komponenta `Profile`). Tlačítka jsou přístupná pouze přihlášenému uživateli.To je nepřihlášenému uživateli zobrazeno po najetí kurzorem na tlačítko.

```
if (this.state.editAnimal) {
    return (
        <Redirect
            to={{
                pathname: "/edit",
                state: {animal: this.props.animal, animalId: this.props.animalId}
            }}
        />
    );
}
```

Obrázek 13 - Přesměrování uživatele komponentou Redirect

6.1.11.4. Filter

Komponenta `Filter` se využívá v komponentách `Animals` s `Swipe`. Pro každou komponentu má vlastní styl, ale zobrazuje stejná data (viz Obrázky 36 a 43). Komponenta slouží k získání filtrovacích dat od uživatele a předání nadřazené komponentě. Ve výchozím stavu je zobrazen jen prvek s textem „Filtrovat“ nebo ikonou filtru. Po kliknutí se zobrazí celý filtr s formulářem.

Typy zvířete, velikost, pohlaví a chování si uživatel vybírá zaškrtávacími poli. Zaškrtávací pole chování se generují dynamicky podle zvolených typů zvířete. Pro rasu je textové pole. Po kliknutí do pole se zobrazí nabídka ras načtená z JSON souboru. Psaním se nabídka filtruje podle napsaného textu do pole. Věk a hmotnost se filtrují určením spodní a vrchní hranice. K tomu jsem využil komponentu `Slider` z knihovny `material-ui`.

Kliknutím na tlačítko „Filtrovat“ se zavolá funkce `filter(data)`, která zajistí samotné filtrování. Je to funkce nadřazené komponenty, která je této komponentě předána v `props`. Parametrem je objekt se všemi daty z formuláře.

6.1.11.5. Toolbar

Komponenta `Toolbar` slouží k zobrazení menu. Součástí menu je položka „Domů“, která odkazuje na komponentu `Home`. Ostatní položky se mohou lišit podle toho, jestli je uživatel přihlášen.

V metodě `componentDidMount()` se kontroluje přihlášení uživatele funkcí `onAuthStateChanged()` služby Firebase Authentication (viz Obrázek 14). Ta po celou existenci komponenty naslouchá změnám stavu přihlášení do aplikace. Podle aktuálního stavu vykreslí komponentu s požadovanými položkami menu. Když uživatel není přihlášen, vykreslí se komponenta `PublicToolbar`. V opačném případě `PrivateToolbar`, ve které se položky vykreslují ještě na základě typu přihlášeného uživatele.

Každá položka menu obsahuje komponentu `NavLink` z knihovny `react-router-dom`. Té se jako parametr předá URL komponenty, která se má zobrazit.

```
componentDidMount() {
    auth.onAuthStateChanged( nextOrObserver: (user : User | null ) => {
        if (user) {
            this.loadData(user.uid);
        } else {
            this.setState( state: {toolbar: <PublicToolbar/>});
        }
    });
}
```

Obrázek 14 - Kontrola stavu přihlášení uživatele

6.1.11.1. Reviews

Pomocná komponenta, která je využitá v komponentě Profile. Slouží k vykreslení uživatelských recenzí na profilu zvířecího útulku. Součástí komponenty je i přidání nové recenze. Recenze obsahuje text, hodnocení, jméno hodnotitele a čas vytvoření. Hodnocení se zadává klikáním na hodnotící panel ve formě hvězd. Takto je hodnocení i zobrazeno. Pokud uživatel nemá vyplněné jméno, recenze je vložena jako anonymní.

6.1.12. Konfigurační soubory a konstanty

Pro větší přehlednost jsem si vytvořil konfigurační soubory pro používání služeb Firebase a HERE Maps. Pro Firebase je soubor Firebase.js, ve kterém je potřebná konfigurace k připojení k jeho službám. Jsou zde také vytvořeny a exportovány používané objekty těchto služeb (viz Obrázek 15).

Pro HERE Maps existuje soubor HERE.js, ze kterého jsou exportovány objekty pro mapu a vyhledávací službu.

Většina konstant použitých v aplikaci je definována v souboru Const.js. Jsou zde například názvy kolekcí v databázi, názvy vlastností zvířat nebo názvy akcí opakujících se tlačítek.

```
export const db = firebase.firestore();
export const storage = firebase.storage();
export const auth = firebase.auth();
export const timestamp = firebase.firestore.Timestamp;
export const fieldPath = firebase.firestore.FieldPath;
export const fieldValue = firebase.firestore.FieldValue;
```

Obrázek 15 - Export objektů Firebase

6.2. Jazykové mutace

Webová aplikace má možnost fungovat ve více jazycích. K vícejazyčnosti jsem využil JavaScriptovou knihovnu `I18next`. Aktuálně aplikace nabízí dva jazyky. Jazyk má svůj JSON soubor, kde jsou uloženy použité řetězce ve formátu klíč - hodnota (viz Obrázek 16). Výchozí jazyk je určen z prohlížeče uživatele pomocí balíčku `i18next-browser-languagedetector`. Jinak má uživatel možnost jazyky manuálně přepínat.

```
    "messages": "Zprávy",
    "myAnimals": "Moje zvířata",
    "addAnimal": "Přidat zvíře"
  },
  "animals": {
    "dogs": {
      "dogs": "Psi",
      "male": "Pes",
      "female": "Fena"
    },
    "cats": {
```

Obrázek 16 - Ukázka souboru s překlady

V jednotlivých komponentách se překlad aplikuje obalením komponenty funkcí `withTranslation()`. Tím komponenta získá přístup k souborům s překlady přes proměnnou `props.t`. Na obrázku 17 je ukázka použití překladu u jedné položky menu.

```
<li className="toolbar_items_item">
    <NavLink to="/add" className="toolbar_items_item_link">
        <p className="toolbar_items_item_text">{t('toolbar.addAnimal')}</p>
    </NavLink>
</li>
```

Obrázek 17 - Použití překladu v metodě render()

7. Mobilní aplikace

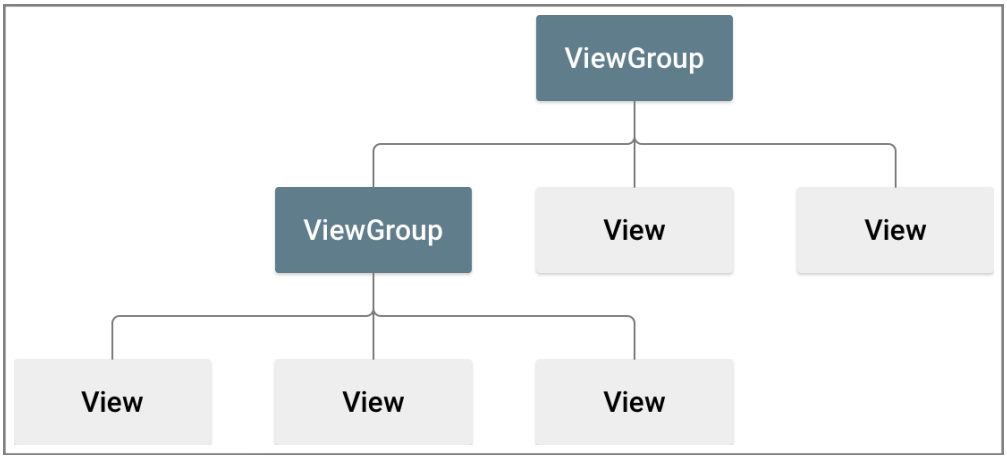
Druhou částí tohoto řešení je mobilní aplikace pro platformu Android. Aplikace je určena především pro zájemce o zvíře z útulku. Nabízí ale i omezené funkce pro útulky. Aplikaci jsem psal v jazyce Java.

7.1. Struktura aplikace

Aplikace se řídí architekturou MVVM (ModelViewViewModel). Základem aplikace je jedna aktivita. Dále je aplikace rozdělena na fragmenty, které aktivita zobrazuje na základě vstupů od uživatele. Data z databáze a komunikaci mezi fragmenty a aktivitou zprostředkovává `BaseViewmodel`.

7.1.1. Uživatelské rozhraní

Uživatelské rozhraní je definováno v jazyce XML v samostatných souborech, aby bylo oddělené od logiky aplikace. Rozhraní reprezentují tzv. layouty. Layout je poskládaný z tříd `View` a `ViewGroup` a jejich podtříd, které tvoří stromovou strukturu. Pod třídou `ViewGroup` jsou například `ConstraintLayout` nebo `LinearLayout`. Tyto třídy slouží k obalení prvků View a definují jejich rozložení. Podtřídy View jsou například Button, `TextView` nebo `ImageView`. Na obrázku 18 je schéma hierarchie objektů layoutu.



Obrázek 18 - Schéma hierarchie Android UI

Jednotlivé XML soubory jsou svázány se svými protějšky v logice aplikace. Je více možností, jak je svázat. Já jsem v této aplikaci využil funkcionalitu View binding. View

binding vygeneruje pro každý layout vlastní třídu s názvem odvozeným od názvu XML souboru. Tato třída obsahuje odkazy na všechny prvky layoutu, ke kterým je pak možné přistupovat z tříd v logice aplikace.

7.1.2. Aktivita BaseActivity

`BaseActivity` je jediná aktivita aplikace a dědí z `AppCompatActivity`. Aktivita se stará o zobrazování jednotlivých fragmentů. K tomu slouží `NavigationView` a `DrawerLayout`. Pokud uživatel klikne na ikonu menu nebo přejede prstem horizontálně od pravé strany obrazovky, pomocí `DrawerLayout` se s animací vysune menu. Menu má uživatel k dispozici ve všech fragmentech aplikace (viz Obrázek 53).

O chování při kliknutí na položku v menu se stará `NavigationView`, konkrétně metoda `onNavigationItemSelected(MenuItem)` (viz Obrázek 19). V metodě je `switch`, který při každé možnosti otevře příslušný fragment. Fragment se otevírá metodou třídy `NavController navigate(id, bundle)`. Parametry jsou ID fragmentu a objekt `Bundle`, pokud se mají fragmentu předat nějaká data.

```
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
    NavController navController = getNavigationController();
    drawerLayout.closeDrawer(GravityCompat.END);
    Bundle bundle = new Bundle();
    int id = menuItem.getItemId();
    switch (id) {
        case R.id.home:
            navController.navigate(R.id.mainPage);
            break;
        case R.id.dog_list:
            bundle.putString("animalType", Const.DOGS);
            navController.navigate(R.id.listFragment, bundle);
            break;
        case R.id.cat_list:
```

Obrázek 19 - Metoda onNavigationItemSelected()

Položky menu se liší podle stavu přihlášení uživatele a jeho typu. Jestli je uživatel přihlášen se zjistí funkcí služby Authentication `getCurrentUser()`, která se volá v metodě `onResume()`. Ta vrátí objekt uživatele nebo `null`. Pro nepřihlášeného uživatele má menu

omezený obsah položek. Dvě z nich jsou pro přihlášení a pro registraci. Jednou z položek navíc pro přihlášeného uživatele je „Odhlásit se“. Kliknutím se zavolá metoda v aktivitě pro odhlášení z Firebase. Jednou ze společných položek menu je možnost přepnutí na tmavý režim.

V aktivitě je řešeno chování aplikace po stisknutí vestavěného tlačítka „Zpět“. Při otevřeném menu se jen menu zavře. Jinak se využije `NavigationController`, který uchovává historii otevíraných fragmentů a metodou `navigateUp()` se otevře poslední fragment.

7.1.3. BaseViewModel

Třída `BaseViewModel` dědí ze třídy `ViewModel` a reprezentuje vrstvu, která získává data z databáze a poskytuje je jednotlivým fragmentům. Fragmenty k ní získají přístup v metodě `onCreate()` pomocí třídy `ViewModelProvider`, která poskytne instanci třídy `BaseViewModel`.

Třída obsahuje funkce volající databázi. Každá funkce nastaví `SnapshotListener`, který čeká na změny v databázi po celý životní cyklus aktivity. Když přijde změna, uloží nová data do proměnné typu `MutableLiveData<Object>` (viz Obrázek 20). Tento objekt má metodu `observe(LifecycleOwner, Observer)`. Ta mu přiřadí objekt `Observer`, který pozoruje obsah proměnné, dokud existuje. Ve fragmentech se nastaví `Observery` na objekty s daty, které jsou ve fragmentu potřeba.

```
FirestoreService.getAnimalsLive(Const.DOGS).addSnapshotListener((value, e) -> {
    if (e != null) {
        Log.w( tag: "Error",  msg: "Listen failed.", e);
        return;
    }
    List<Animal> animals = new ArrayList<>();
    for (QueryDocumentSnapshot doc : value) {
        if (doc.get("name") != null) {
            animals.add(doc.toObject(Animal.class));
        }
    }
    dogsLiveData.setValue(animals);
});
```

Obrázek 20 - Získávání dat z Cloud Firestore

`BaseViewModel` také uchovává instanci třídy `FirebaseAuth` a všechny přistupující fragmenty jí mohou využívat ke komunikaci se službou Firebase Authentication.

7.1.4. BaseFragment

Třída, ze které dědí všechny fragmenty aplikace. Obsahuje všechny společné metody pro dědící fragmenty. Sáma dědí z třídy `Fragment`. Jedna z metod této třídy je `getStringresourceByName(String)`, která slouží k zobrazování řetězce ze souboru strings.xml. Využívá se pokud je v souboru potřeba vyhledat řetězec podle hodnoty, která je uložena v proměnné.

7.1.5. MainPageFragment

`MainPageFragment` (viz Obrázek 51) je úvodní fragment aplikace. Uživateli zobrazí tři tlačítka s obrázky, kde každé představuje jednu skupinu zvířat. Kliknutím na jedno z tlačítek se zobrazí `ListFragment` se zvířaty zvoleného typu.

7.1.6. ListFragment

Tento fragment slouží k zobrazování seznamu zvířat na třech stránkách aplikace (viz Obrázek 54). Jedná se o všechna dostupná neadoptovaná zvířata, oblíbená zvířata uživatele nebo nabízená zvířata přihlášeného útulku. Fragmentu je při kliknutí v menu předána v parametru informace, která zvířata má zobrazit. V metodě `onActivityCreated()` se podle toho nastaví `Observer` na zvířata vybrané skupiny, případně typu (viz Obrázek 21) . Při vytvoření fragmentu a poté při každé změně v databázi se uloží data do seznamu. Před uložením se volá metoda `filter(params)`. Ta aplikuje na data aktuálně zvolené filtry, pokud má uživatel vyfiltrováno během příchodu nových dat.

Fragment zobrazuje zvířata do prvku `RecyclerView`. Ten umožňuje vykreslovat seznam položek vlastního obsahu a vzhledu. K práci s `RecyclerView` slouží třída `ListFragmentAdapter` dědící od `RecyclerView.Adapter`. V této třídě se vkládají data do jednotlivých položek seznamu. Obsahuje vlastní `RecyclerView.ViewHolder`, který ji propojí s XML souborem pomocí `ListFragmentItemBinding`.

```
    break;
case Const.OTHER:
    animalsLiveData = baseViewModel.getOtherLiveData();
    animalsLiveData.observe(getActivity(), animals -> {
        hideProgressBar();
        animalList.clear();
        animalList.addAll(animals);
        filter(typeList, genderList, sizeList, ageMin, ageMax, weightMin, weightMax, breed);
    });
    break;
default:
    if(mode.equals(OUR_ANIMALS)) {
        baseViewModel.setLoggedId(auth.getCurrentUser().getUid());
        animalsLiveData = baseViewModel.getOurAnimalsLiveData();
    } else if(mode.equals(LIKED_ANIMALS)) {
        animalsLiveData = baseViewModel.getLikedAnimalsLiveData();
    } else {
        animalsLiveData = baseViewModel.getAnimalsLiveData();
    }
    animalsLiveData.observe(getActivity(), animals -> {
        hideProgressBar();
        animalList.clear();
        animalList.addAll(animals);
        filter(typeList, genderList, sizeList, ageMin, ageMax, weightMin, weightMax, breed);
    });
    break;
```

Obrázek 21 - Nastavení chování po obdržení dat z databáze

V metodě `onBindViewHolder(ListFragmentViewHolder, int position)` se vkládají data položky na dané pozici do určených polí včetně obrázku.

Fragment implementuje rozhraní `ListFragmentAdapter.Callback`. Toto rozhraní má metodu `onItemClick(int position)`. Metoda je ve fragmentu přepsána a zajistí po kliknutí na položku přesměrování uživatele na `DetailFragment`.

Uživatel může zvířata filtrovat dle jejich vlastností. K zobrazení filtru slouží plovoucí tlačítko `FloatingActionButton`. Klikem na tlačítko se zobrazí `FilterDialogFragment`, rozšiřující třídu `AppCompatDialogFragment`.

7.1.7. SwipeListFragment

Fragment sloužící k zobrazování nabízených zvířat a jejich hodnocení (viz Obrázek 55). Zvířata se uživateli zobrazují po jednom a může si každé přidat do oblíbených nebo přeskočit. Ohodnotit může přejetím prstem vlevo nebo vpravo, případně dvěma tlačítky.

Fragment získá data z databáze pomocí třídy `BaseViewModel`. V metodě `onCreate()` získá její instanci (viz Obrázek 22) a v metodě `onActivityCreated()` nastaví `Observer` na objekt `MutableLiveData<List<Animal>>`. Objekt je naplněn daty z databáze v metodě třídy `BaseViewModel`.

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    animalSharedViewModel = new ViewModelProvider(getActivity()).get(AnimalSharedViewModel.class);
    baseViewModel = new ViewModelProvider(getActivity()).get(BaseViewModel.class);
    firebaseAuth = FirebaseAuth.getInstance();
    user = firebaseAuth.getCurrentUser();
    animalList = new ArrayList<>();
}
```

Obrázek 22 - Metoda onCreate()

Po přijetí dat z databáze se naplní objekt třídy `ViewFlipper`. `Ten` umožňuje přecházení mezi prvky `View`. Pro každé zvíře se vytvoří `SwipeItemView`. To je vlastní třída dědící od třídy `CardView`. V konstruktoru se třídě předá kontext fragmentu a následně se přiřadí objekt `Animal` s daty zvířete. `ViewFlipperu` se nastaví `SwipeTouchListener`, který reaguje na přejetí prstem po obrazovce. Má metody pro odchycení každého ze čtyř směrů a metodu `onClick()`. V metodách `onSwipeLeft()` a `onSwipeRight()` se volají `like()` a `skip()`. V obou funkcích se spustí animace přechodu na další `View`, předchozí `View` se odebere a odebere se také ohodnocené zvíře z listu zvířat. V metodě `like()` se zavolá funkce `addUserLikedAnimal(user, animal)` třídy `FirestoreService`. Ta přidá přihlášenému uživateli zvíře do listu oblíbených zvířat. Obdobně funguje i funkce v metodě `skip()`.

V metodě `onClick()` se uživateli zobrazí `DetailFragment` s aktuálním zvířetem.

7.1.8. DetailFragment

`DetailFragment` zobrazuje detailní informace o vybraném zvířeti (viz Obrázek 57). Uživatel se k němu dostane z fragmentů `ListFragment` a `SwipeListFragment` kliknutím na jednu z položek. Ve výchozím stavu je vidět pouze jméno zvířete a obrázek,

který je přes celou obrazovku. Přejetím prstem nahoru se obrázek zmenší a ukáže se zbytek informací o zvířeti.

Objekt `Animal` je fragmentu předán přes třídu `AnimalSharedViewModel`. Jeho instanci fragment získá v metodě `onCreate()` a v metodě `onActiviyCreated()` načte data. Obrázek je zobrazen pomocí třídy `GlideApp` (viz Obrázek 23). Jednou z jejích metod nutných pro zobrazení obrázku je `load(Object)`. Jako parametr je předána reference na soubor v úložišti Cloud Storage, která je vytvořena z cesty k souboru.

```
RequestOptions options = RequestOptions
        .fitCenterTransform()
        .override(binding.detailPhoto.getWidth(), binding.detailPhoto.getHeight());

if (animal.getImage() != null) {
    StorageReference storageReference = StorageService
            .storage.getReference().child(animal.getImage());
    GlideApp.with(getParentFragment()) GlideRequests
            .asBitmap() GlideRequest<Bitmap>
            .apply(options)
            .load(storageReference)
            .placeholder(circularProgressDrawable)
            .into(binding.detailPhoto);
}
```

Obrázek 23 - Vložení obrázku do ImageView

7.1.9. EditProfileFragment

Fragment určený k editaci uživatelského profilu. Obsahuje prvek `ImageView`, kde je zobrazen jeho aktuální obrázek. Dvě tlačítka u obrázku umožňují obrázek odebrat nebo nahrát nový. Pod obrázkem jsou editovatelná textová pole `EditText`. Výchozí hodnoty v polích jsou načteny z databáze. Práci s databází zajišťuje funkce `getUser(uid)` třídy `BaseViewModel`.

7.1.10. ProfileFragment

Tento fragment uživatelský profil pouze zobrazuje (viz Obrázek 56). Obrázek je zobrazen pomocí `ImageView`. Informace o uživateli jsou v prvcích `TextView`. Stejně jako ve

fragmentu `EditProfileFragment`, jsou data získávána prostřednictvím třídy `BaseViewModel`.

7.1.11. RegistrationFragment

`RegistrationFragment` zobrazuje uživateli registrační formulář (viz Obrázek 52). Pole formuláře jsou prvky `EditText`. Po stisknutí tlačítka „Registrovat“ proběhne validace vstupních dat. Pro validaci jsem využil knihovnu Saripaar. Knihovna umožňuje využívat anotace k definování podmínek pro kontrolovaná pole (viz Obrázek 24). Samotnou validaci zajišťuje objekt `Validator`. Jeho metoda `validate()` spustí validaci a proběhne jedna z funkcí `onValidationSucceeded()` a `onValidationFailed(List<Error>)`.

```
@NotEmpty
@Email
EditText emailEditText;
@Password(scheme = Password.Scheme.ALPHA_NUMERIC)
EditText passwordEditText;
@ConfirmPassword
EditText password2EditText;
```

Obrázek 24 - Anotace validace knihovny Saripaar

Při úspěchu validace se zavolá funkce služby Firebase Authentication `createUserWithEmailAndPassword(email, password)`, která po vlastní validaci vytvoří Firebase uživatele. Po vytvoření se uloží vyplněná data do databáze Firestore metodou `FirestoreService.addUserToFirestore(user)`. Na závěr je uživatel přihlášen a přesměrován na hlavní stránku aplikace `MainPageFragment`.

Při neúspěchu validace zobrazí validátor chybové hlášky v prvcích `EditText`.

7.1.12. LoginFragment

Tento fragment slouží k přihlášení uživatele do aplikace. Fragment obsahuje přihlašovací formulář se dvěma editovatelnými textovými poli `EditText` (viz Obrázek 50). Uživatel je tázán na přihlašovací e-mail a heslo. Formulář je po stisknutí tlačítka „Přihlásit“ validován nejdříve na straně aplikace pomocí knihovny Saripaar. Ta kontroluje prázdná pole a formát e-mailu. O druhou část validace se stará služba Firebase Authentication při zavolání její

funkce `signInWithEmailAndPassword(email, password)`. Ta kontroluje existenci uživatele a správnost hesla. Po přihlášení je uživatel přesměrován na hlavní stránku aplikace `MainPageFragment`.

7.1.13. MessagesFragment

Fragment sloužící k zobrazování konverzací přihlášeného uživatele. Konverzace jsou reprezentovány jménem a obrázkem uživatele. Fragment pomocí instance třídy `BaseViewModel` získává data z databáze. V databázi je v dokumentu uživatele uložen seznam jeho konverzací. Každá konverzace obsahuje ID účastníků a ze všech je vytvořen seznam uživatelů, se kterými vede přihlášený uživatel konverzaci.

Kliknutí na konverzaci se otevře `ChatFragment` a v `BaseViewModelu` se nastaví ID aktivní konverzace.

7.1.14. ChatFragment

`ChatFragment` zobrazuje otevřenou konverzaci a její zprávy. Ty jsou načteny na základě ID aktivní konverzace, které si drží `BaseViewModel`. Jednotlivé zprávy zobrazuje `ChatFragmentAdapter` do prvku `RecyclerView`. V metodě `OnViewCreated()` je nastaveno, aby se položky `RecylerView` zobrazovaly od spodu a spodní položka byla ve výchozím stavu vždy vidět (viz Obrázek 25).

```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    binding.toolbar.binding.toolbarText.setText("Zprávy");
    LinearLayoutManager linearLayoutManager = new LinearLayoutManager(getContext());
    linearLayoutManager.setReverseLayout(true);
    binding.recyclerView.setLayoutManager(linearLayoutManager);
    binding.recyclerView
        .addOnLayoutChangeListener((view1, left, top, right, bottom, oldLeft, oldTop, oldRight, oldBottom) -> {
            if (bottom < oldBottom) {
                binding.recyclerView.postDelayed(() ->
                        binding.recyclerView.smoothScrollToPosition(adapter.getItemCount()),  delayMillis: 100);
            }
        });
    binding.sendButton.setOnClickListener(v -> {
        sendMessage();
    });
}
```

Obrázek 25 - Metoda OnViewCreated()

7.1.15. Toolbar

`Toolbar` je třída reprezentující horní panel aplikace. Obsahuje textové pole a ikonu. Obsah si nastavují jednotlivé fragmenty, ve kterých je panel zobrazen.

7.1.16. Modely

Aplikace obsahuje třídy pro zvíře, uživatele, konverzaci, zprávu a polohu. Každá třída obsahuje proměnné reprezentující atributy v databázi, konstruktory s různými parametry a metody pro přístup k proměnným

7.1.17. Pomocné třídy

7.1.17.1. FirestoreService

Tato třída slouží k práci s databází. Je to abstraktní třída, která obsahuje instanci třídy `FirebaseFirestore` služby Cloud Firestore. Dále obsahuje metody pro různá volání databáze, které využívají fragmenty aplikace. Na obrázku 26 je příklad funkce získávající zprávy z konverzace mezi dvěma uživateli.

```
public static Query getMessages(String conversationId) {
    return db.collection( collectionPath: "conversations")
            .document(conversationId).collection( collectionPath: "messages")
            .orderBy( field: "sent", Query.Direction.DESCENDING);
}
```

Obrázek 26 - Získávání zpráv z databáze

7.1.17.2. StorageService

Abstraktní třída `StorageService` zprostředkovává komunikaci se službou Cloud Storage. Obsahuje instanci třídy `FirebaseStorage` a metody pro práci s úložištěm.

7.1.17.3. Const

`Const` je třída pro uchovávání konstant pro celou aplikaci. Uchovávám zde především konstanty používané ve více fragmentech.

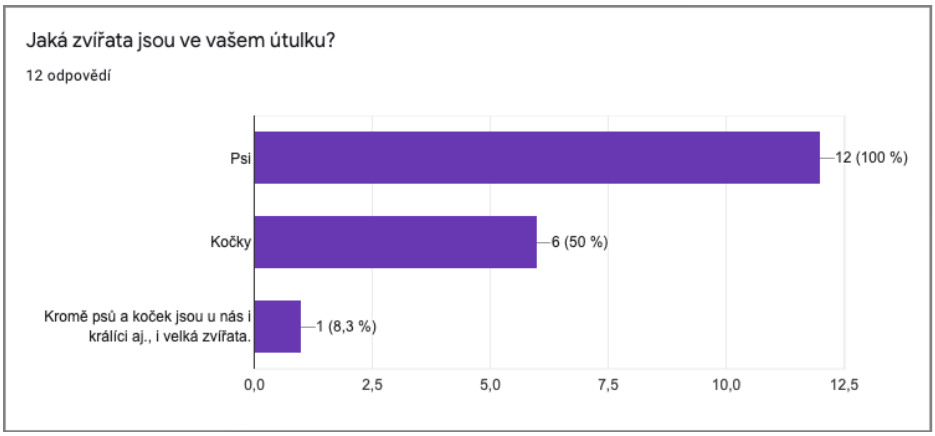
7.1.17.4. CSVFile

Tato třída načítá soubor formátu CSV. V konstruktoru převezme datový tok `InputStream`. Obsahuje jednu metodu `read()`, ve které načte vstup pomocí třídy `BufferedReader` a vrátí seznam řetězců. Třída je využitá při načítání zvířecích ras a seznamu registračních čísel zvířecích útulků.

8. Testování

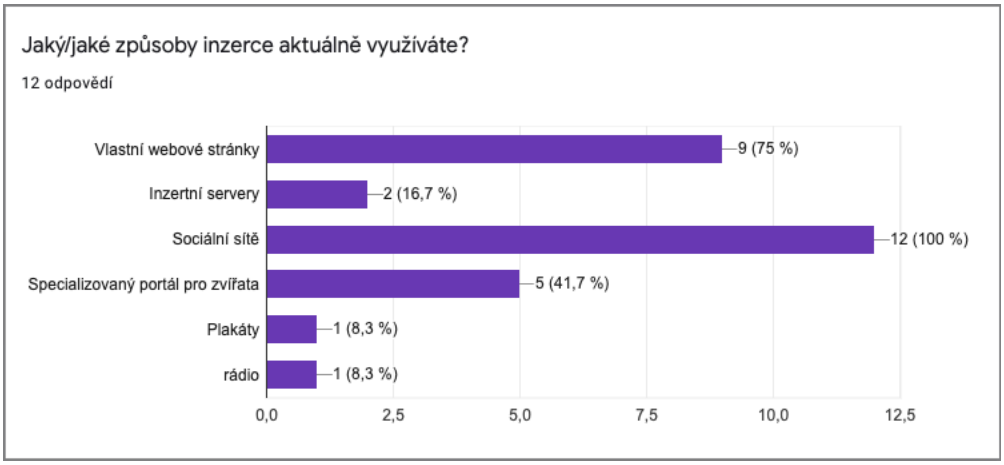
V závěrečné fázi projektu jsem spolupracoval s 12 zvířecími útulky, abych získal důležitou zpětnou vazbu k využitelnosti aplikace. Testování webové aplikace se zúčastnilo všech 12 a 1 z nich testoval i aplikaci mobilní. Odpovědi na připravené otázky jsem sbíral pomocí elektronického formuláře ve službě Formuláře Google. Ostatní připomínky a chyby z testování mi byly zasílány e-mailem. Výsledky z testování jsou zobrazeny v následujících grafech a tabulkách.

Graf 1 potvrzuje předpoklad, že většina zvířat v útulcích jsou psi a kočky. Proto má aplikace jen 3 kategorie: Psi, kočky a ostatní druhy.

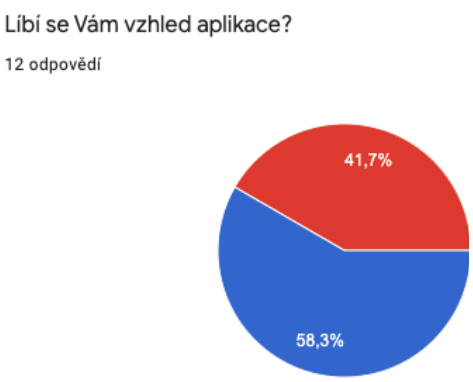


Graf 1 - Jaká zvířata jsou ve vašem útulku

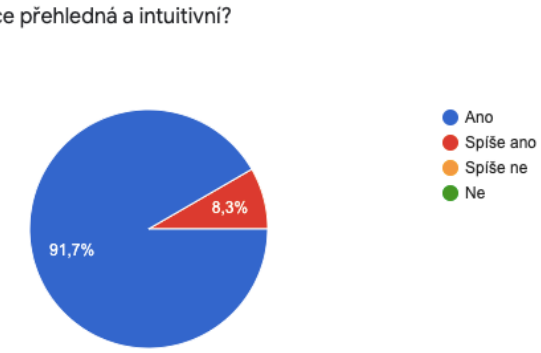
Graf 2 ukazuje, že uživatelé aktuálně inzerují především na sociálních sítích a vlastních webových stránkách. Specializované portály pro zvířata útulky aktuálně tolik nepoužívají, ale uvedly, že o tuto aplikaci mají zájem.



Graf 2 - Jaké způsoby inzerce využíváte

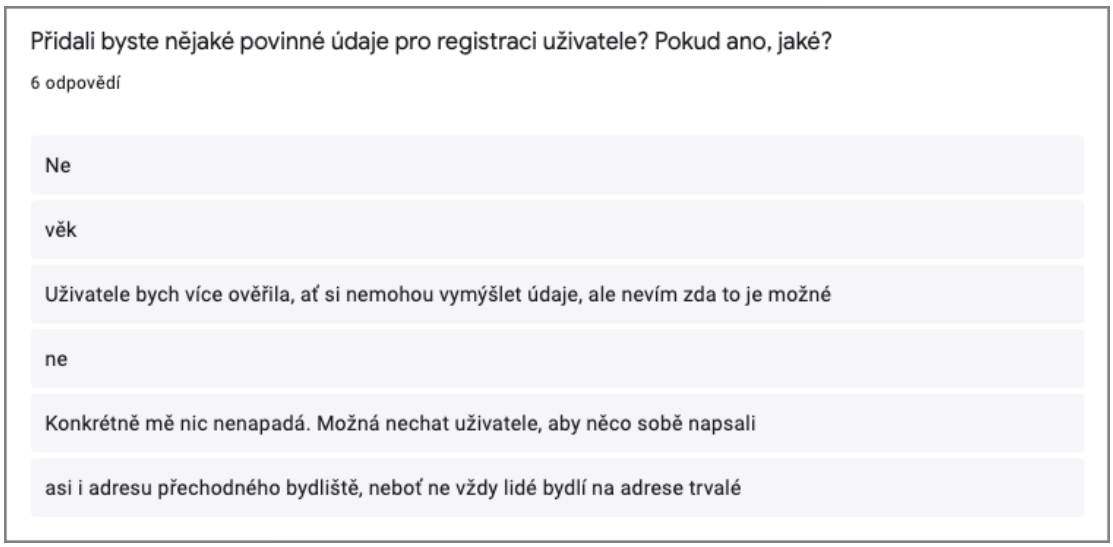


Graf 3 - Líbí se vám vzhled aplikace



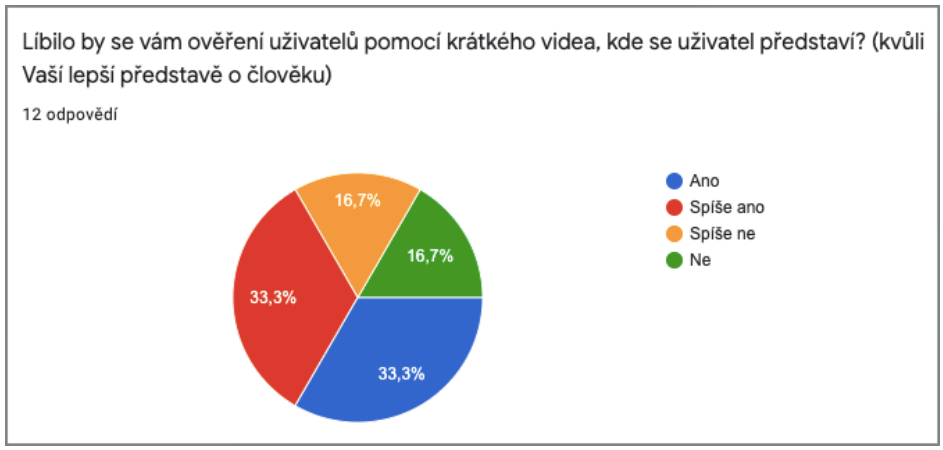
Graf 4 - Je aplikace přehledná a intuitivní

Věk uživatele bude přidán jako volitelná možnost. Větší ověření uživatele aktuálně neplánuji.



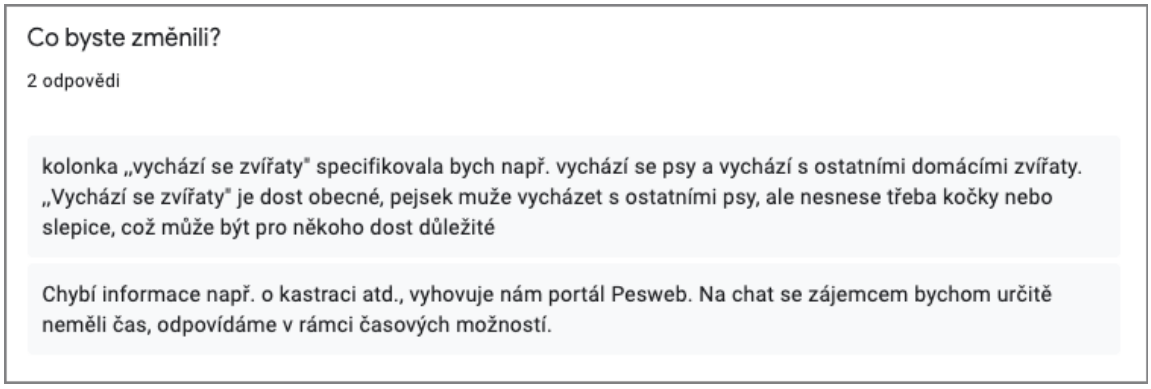
Obrázek 27 - Přidali byste nějaké povinné údaje pro registraci uživatele

Graf 5 ukazuje zájem útulků o rozšíření profilu uživatele o krátké video. Proto jsem přidal možnost nahrání videa mezi budoucí vylepšení aplikace.

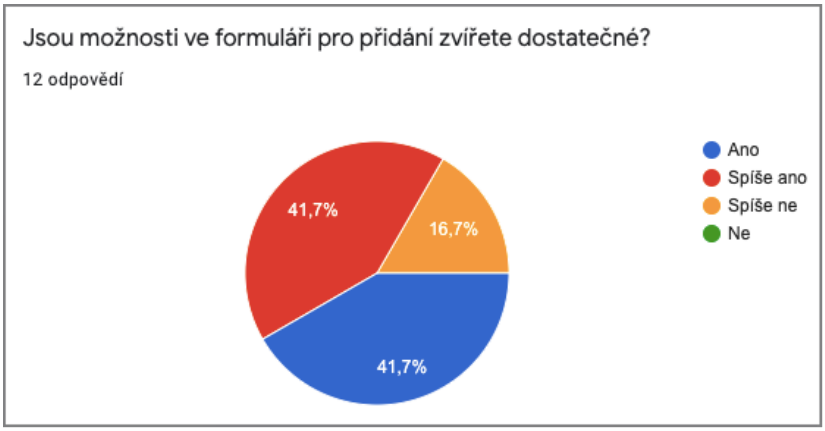


Graf 5 - Ověření uživatelů pomocí videa

Graf 6 - Je formulář pro přidání zvířete rychlý a snadný

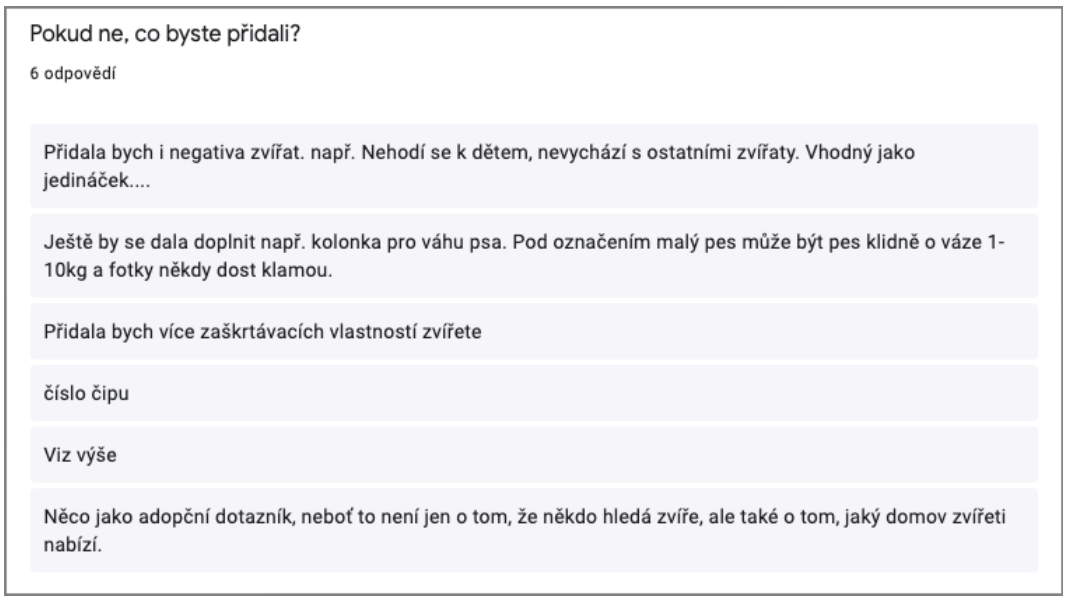


Obrázek 28 - Co byste změnili

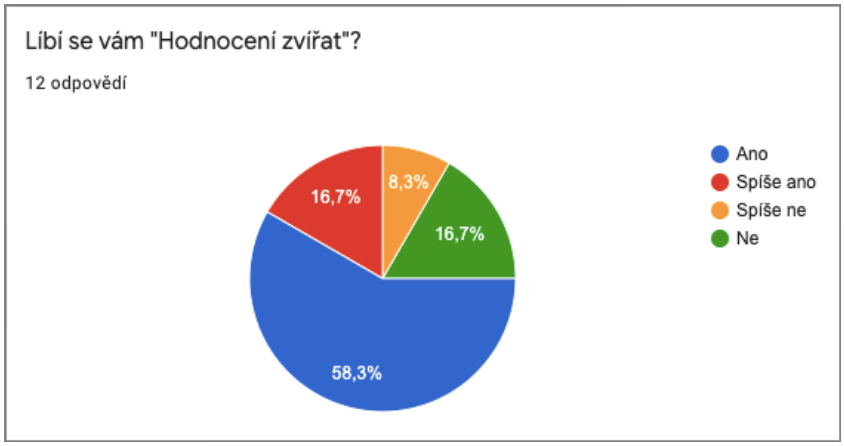


Graf 7 - Jsou možnosti ve formuláři pro přidání zvířete dostatečné

Na základě připomínek (viz Obrázky 28 a 29) jsem upravil formulář pro přidání zvířete. Přidal jsem informaci o kastraci a hmotnosti. Doplnil jsem i možnost vyplnit číslo čipu zvířete, což je i součást zadání. Místo útulkem navrhovaného dotazníku jsem přidal možnost uživateli napsat něco o sobě ve svém profilu.

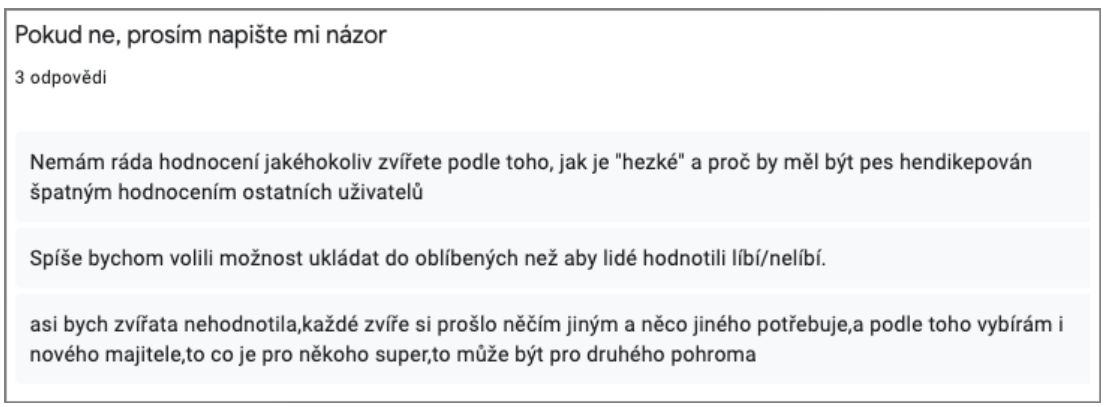


Obrázek 29 - Co byste přidali do formuláře



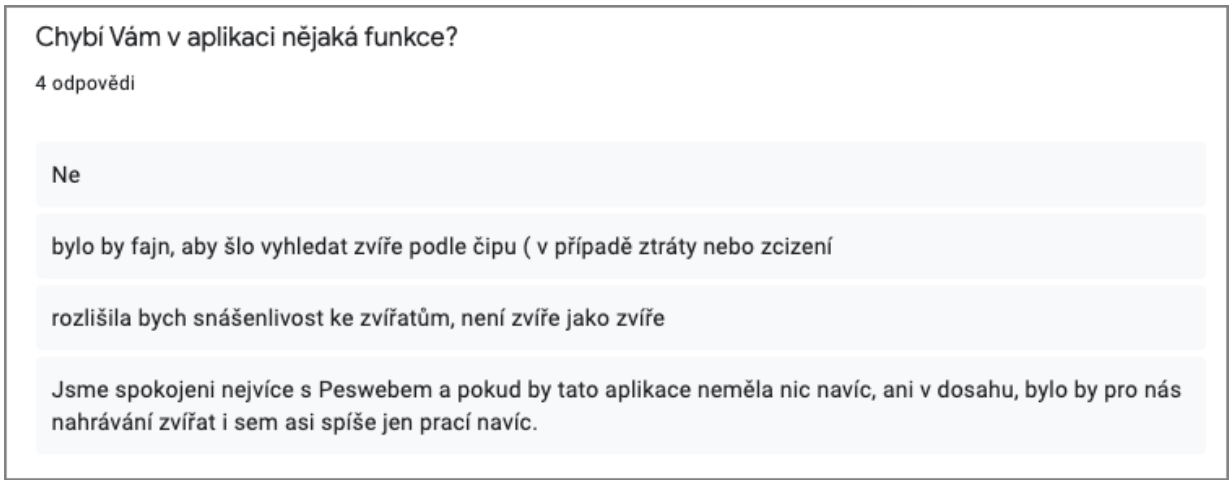
Graf 8 - Hodnocení zvířat

Hodnocení zvířat mělo nešťastně pojmenované možnosti a nemá sloužit k hodnocení vzhledu. Komponentu jsem podle toho upravil.

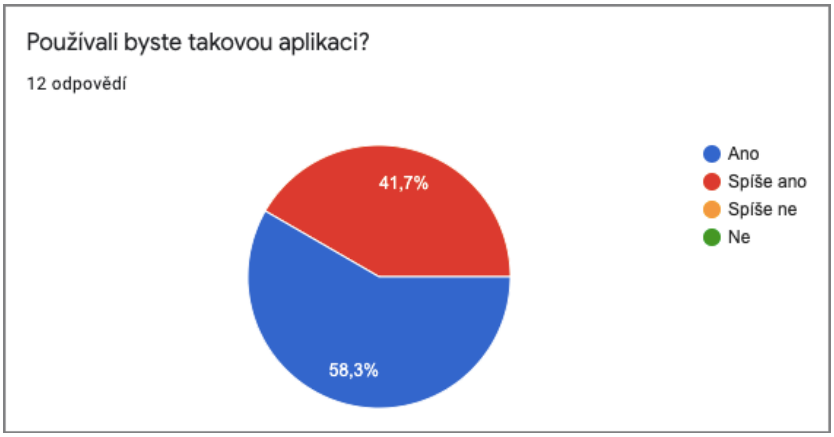


Obrázek 30 - Názor na „Hodnocení zvířat“

Rozšíření snášenlivosti zvířete k ostatním zvířatům (viz Obrázky 28 a 31) jsem přidal do možných vylepšení aplikace.



Obrázek 31 - Chybí v aplikaci nějaká funkce



Graf 9 - Používali byste takovou aplikaci

9. Možná vylepšení

9.1. Aplikace pro platformu iOS

V budoucnu bych chtěl projekt rozšířit o aplikaci pro platformu iOS. Tím by došlo k rozšíření skupiny zájemců o uživatele zařízení společnosti Apple.

9.2. Video zájemce o zvíře

Podle dotazníku pro zvířecí útulky (viz Graf 5) by mohla aplikace v budoucnu nabízet zájemcům nahrání krátkého videa. Video by umožnilo zájemci krátce se představit útulku. Útulky by díky tomu měly větší představu o jednotlivci či rodině, která má zájem o jejich zvíře. Dále také o prostředí, ve kterém zvíře bude žít.

9.3. Černá listina uživatelů

Aplikace bude obsahovat „černou listinu“ uživatelů, která bude přístupná registrovaným útulkům. Útulky budou mít možnost sdílet mezi sebou podvodné zájemce o zvířata.

9.4. Možnost zvýraznění zvířete

Pokud bude nutné určité zvíře urychleně umístit do rodiny například z důvodu plné kapacity útulku, bude možné urychlit jeho adopci umístěním mezi první zobrazovaná zvířata.

9.5. Rozšíření vlastností zvířete

Na základě připomínek v dotazníku (viz Obrázky 29 a 31) bude v budoucnu rozšířen formulář pro přidání zvířete. Bude zvýšen počet zaškrtávacích polí a umožněno specifikovat snášenlivost daného zvířete s ostatními.

10. Závěr

Práci jsem zahájil rešerší aktuálních možností online inzerce zvířat. Průzkum ukázal, že nejvíce používané jsou čtyři způsoby inzerce. Na tyto jsem se zaměřil. Jedná se o inzerci na vlastních webových stránkách, sociálních sítích, inzertních serverech a specializovaných portálech pro zvířata. Všechny možnosti mají nějaké výhody a nevýhody, které jsem na závěr rešerše shrnul a následně vyjasnil, na co se zaměřím ve svém řešení.

Další kapitola se věnuje struktuře celého systému. Stručně jsem popsal využití platformy Google Firebase. Napojení na konkrétní služby je znázorněno schématem. Ve zbytku kapitoly jsem definoval datové struktury použité v aplikaci.

Dále jsem do práce zařadil popis použitých technologií. Jedná se především o Google Firebase, u které jsem představil služby využité v aplikaci. Krátce jsem popsal knihovnu ReactJS, její základní prvky a nástroj Figma, který jsem použil při grafickém návrhu.

Na základě informací z rešerše a grafického návrhu jsem implementoval webovou aplikaci. Aplikace byla zhotovena tak, aby splňovala všechny požadavky návrhu a mohla spolehlivě sloužit zvířecím útulkům a jejich zákazníkům. Ve zprávě je představena struktura aplikace a popsána implementace jednotlivých komponent včetně práce s knihovnou ReactJS.

Druhou částí řešení je mobilní aplikace pro platformu Android. Ta je určena především pro zájemce o zvíře. Je zaměřena na přehledné a rychlé prohlížení zvířat. Oproti webové aplikaci má omezené některé funkce pro útulky. V kapitole určené mobilní aplikaci je rozebrána implementace aplikace v jazyce Java. Kapitola je rozdělena na implementace jednotlivých tříd, převážně fragmentů.

V závěru práce jsem ve spolupráci s vybranými útulky provedl testování aplikace. Výsledky testování jsem znázornil pomocí grafů a obrázků

Poslední částí jsou možné budoucí úpravy a zdokonalení aplikace. Cílem celého projektu je, aby se aplikace stala nástrojem, který se rozšíří mezi širokou veřejnost a pomůže zvířecím útulkům zajistit jejich obyvatelům nový domov v té správné rodině.

Webová aplikace je dostupná na adrese https://shelters-2932f.web.app. Tato aplikace není primárně určena pro mobilní zařízení.

Seznam použité literatury

[1] Používání Mobilního Telefonu A Internetu Na Mobilním Telefonu. *Český Statistický Úřad* [Online]. 25.11.2020 [Cit. 2021-5-5]. Dostupné Z: Https://Www.Czso.Cz/Csu/Czso/3-Pouzivani-Internetu-Jednotlivci

[2] Používání Mobilního Telefonu A Internetu Na Mobilním Telefonu. *Český Statistický Úřad* [Online]. 25.11.2020 [Cit. 2021-5-5]. Dostupné Z: Https://Www.Czso.Cz/Csu/Czso/3-Pouzivani-Internetu-Jednotlivci

[3] Registrované Útulky Pro Zvířata. *Státní Veterinární Správa* [Online]. [Cit. 2021-5-5]. Dostupné Z: Https://Www.Svscr.Cz/Registrovane-Subjekty-Svs/Registrovane-Utulky-Pro-Zvirata/

[4] Stevenson, Doug. What Is Firebase? The Complete Story, Abridged. *Medium* [Online]. 2018-08-25 [Cit. 2021-5-5]. Dostupné Z: Https://Blog.Back4App.Com/What-Is-Cloud-Firestore/

[5] Firebase. *Firebase* [Online]. [Cit. 2021-5-5]. Dostupné Z: Https://Firebase.Google.Com/

[6] Batschinski, George. What Is Cloud Firestore? Back4App [Online]. [Cit. 2021-5-5]. Dostupné Z: Https://Blog.Back4App.Com/What-Is-Cloud-Firestore/

[7] Kerpelman, Todd. Cloud Firestore Vs The Realtime Database: Which One Do I Use? *Firebase: The Firebase Blog* [Online]. October 3, 2017 [Cit. 2021-5-5]. Dostupné Z: Https://Firebase.Googleblog.Com/2017/10/Cloud-Firestore-For-Rtdb-Developers.Html

[8] Clark, Jessica. What Is Firebase Authentication? *Back4App* [Online]. [Cit. 2021-5-5]. Dostupné Z: Https://Blog.Back4App.Com/Firebase-Authentication/

[9] Get Started With Cloud Firestore Security Rules. *Firebase* [Online]. 2021-04-30 [Cit. 2021-5-5]. Dostupné Z: Https://Firebase.Google.Com/Docs/Firestore/Security/Get-Started

[10] Máca, Jindřich. Úvod Do React. *Itnetwork.Cz* [Online]. [Cit. 2021-5-5]. Dostupné Z: Https://Www.Itnetwork.Cz/Javascript/React/Zaklady/Uvod-Do-React/

[11] Kopf, Ben. The Power Of Figma As A Design Tool. *Toptal: Designers* [Online]. [Cit. 2021-5-5]. Dostupné Z: Https://Www.Toptal.Com/Designers/Ui/Figma-Design-Tool

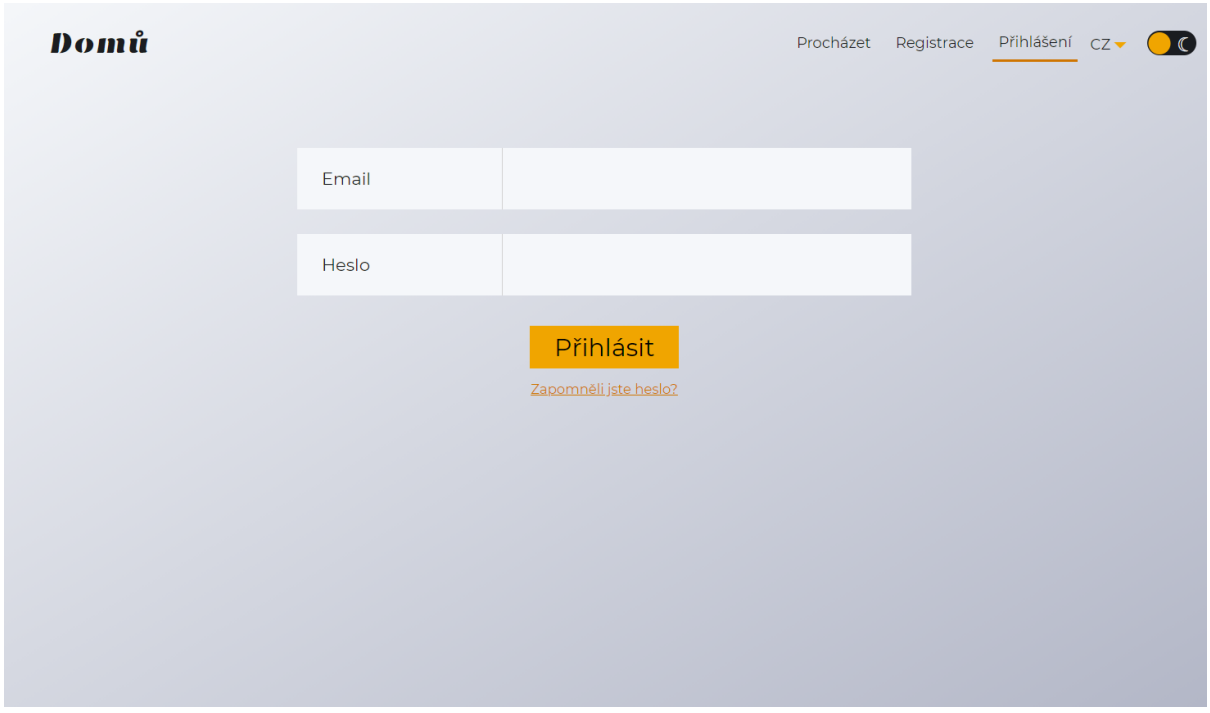
Seznam obrázků

Seznam tabulek

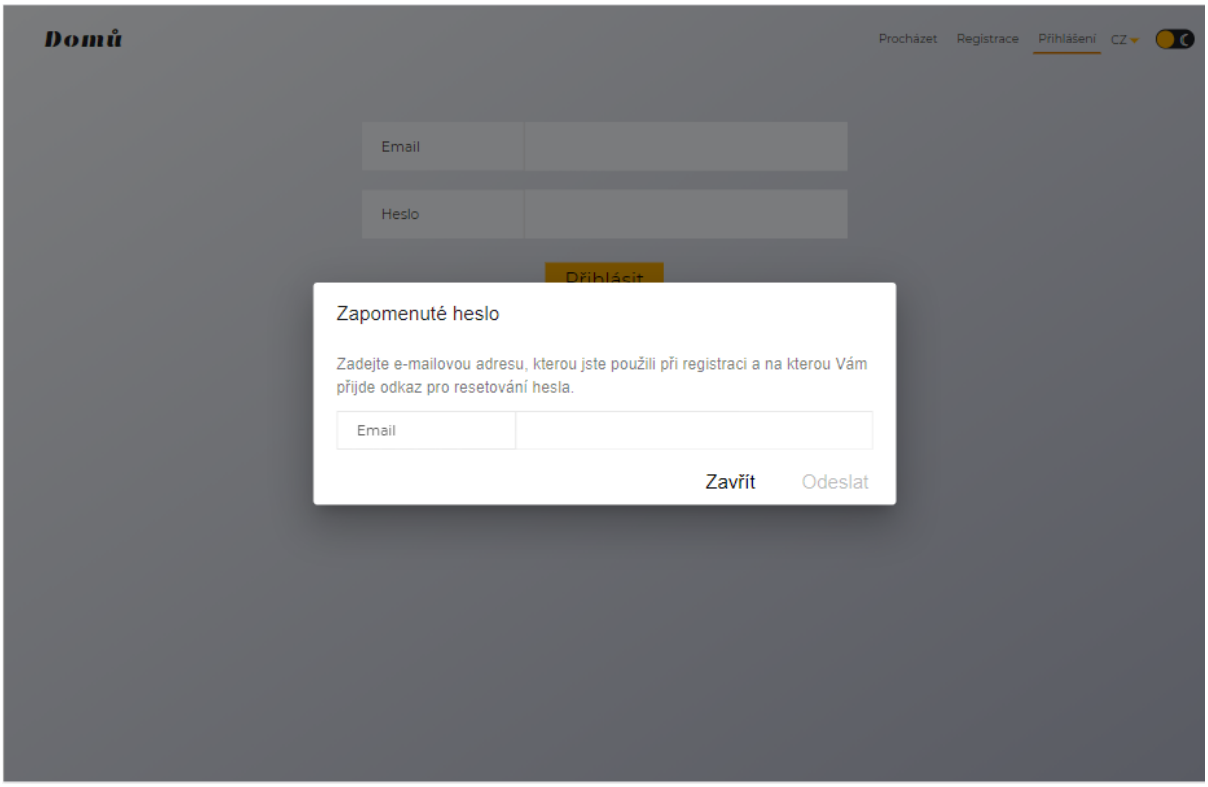
Seznam grafů

A Obrázky webové aplikace

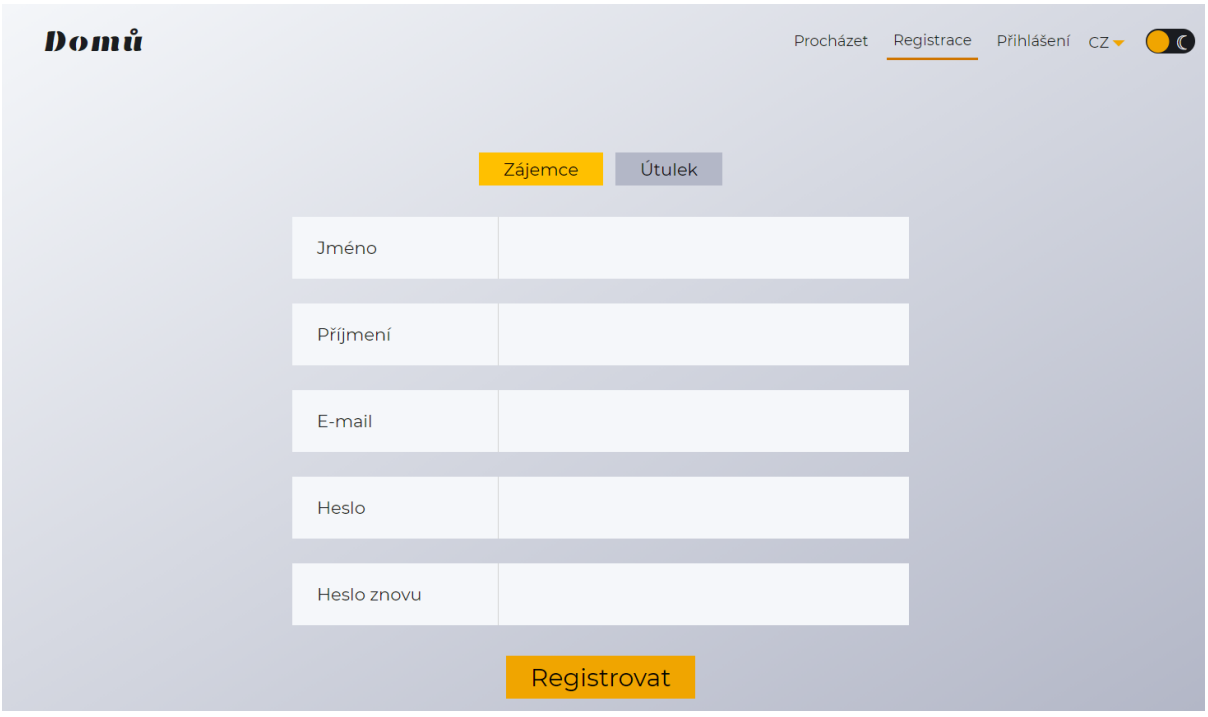
Obrázek 32 - Domovská stránka



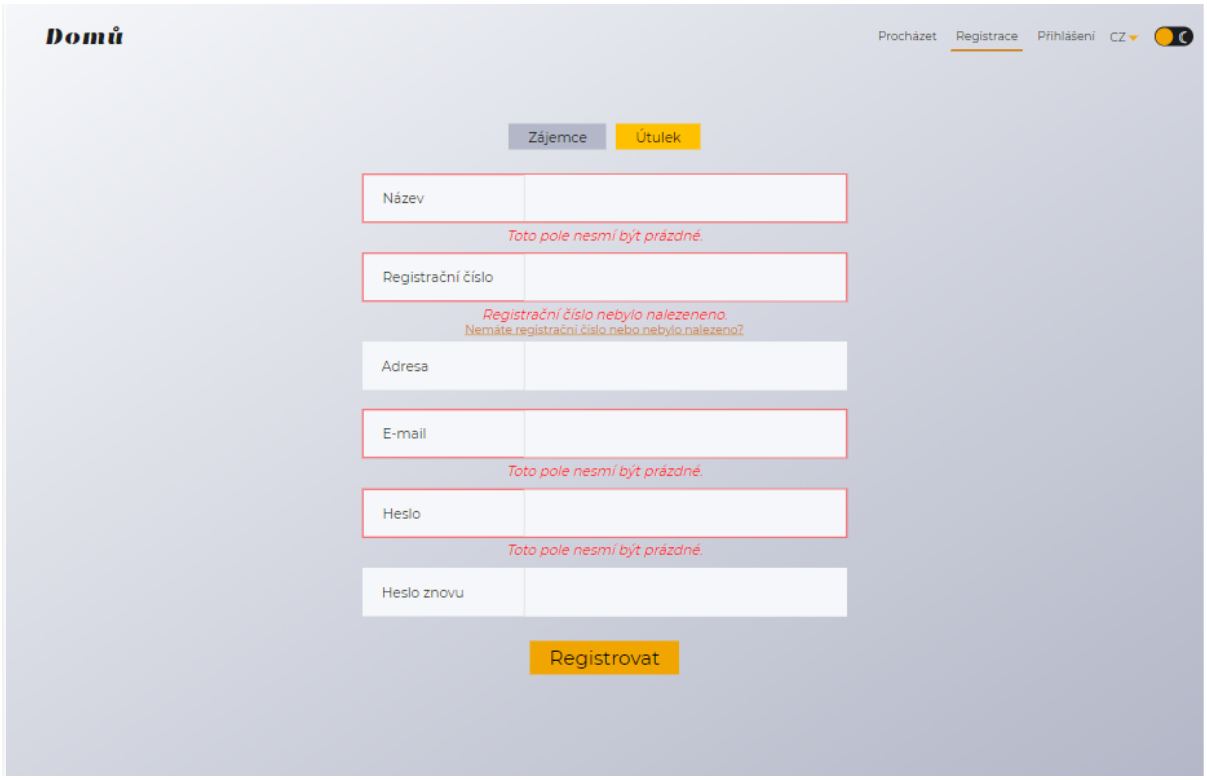
Obrázek 33 - Přihlašovací stránka



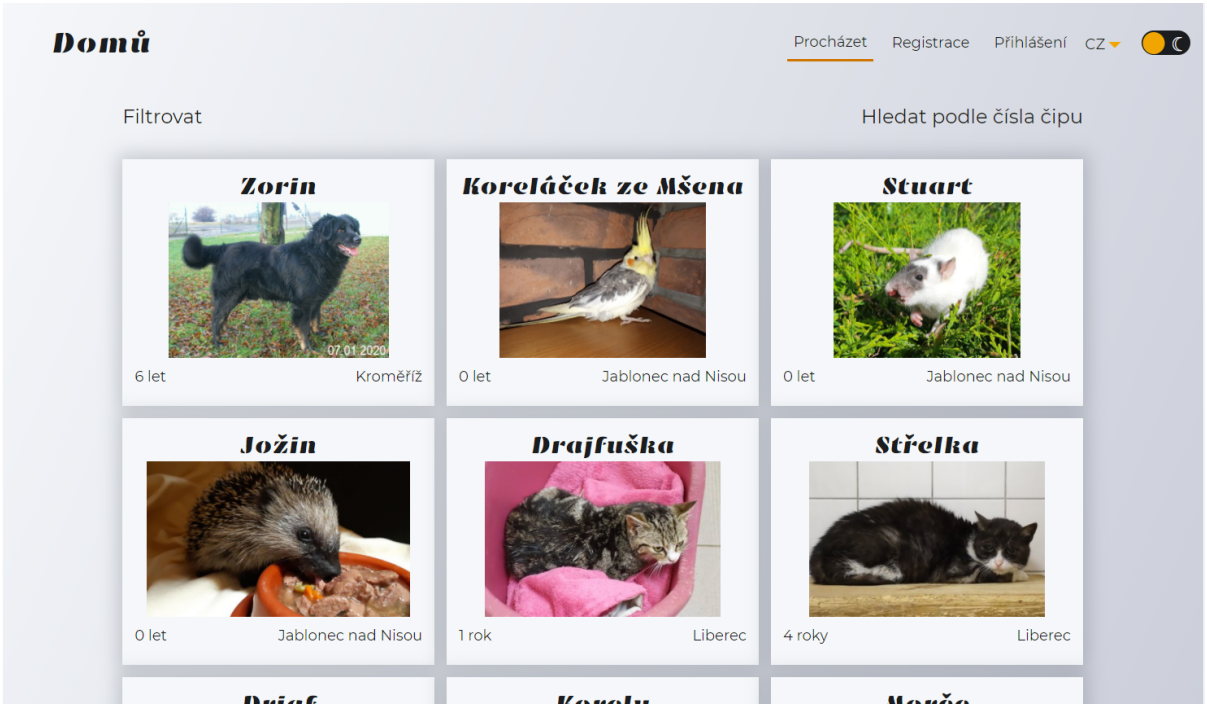
Obrázek 34 - Zapomenuté heslo



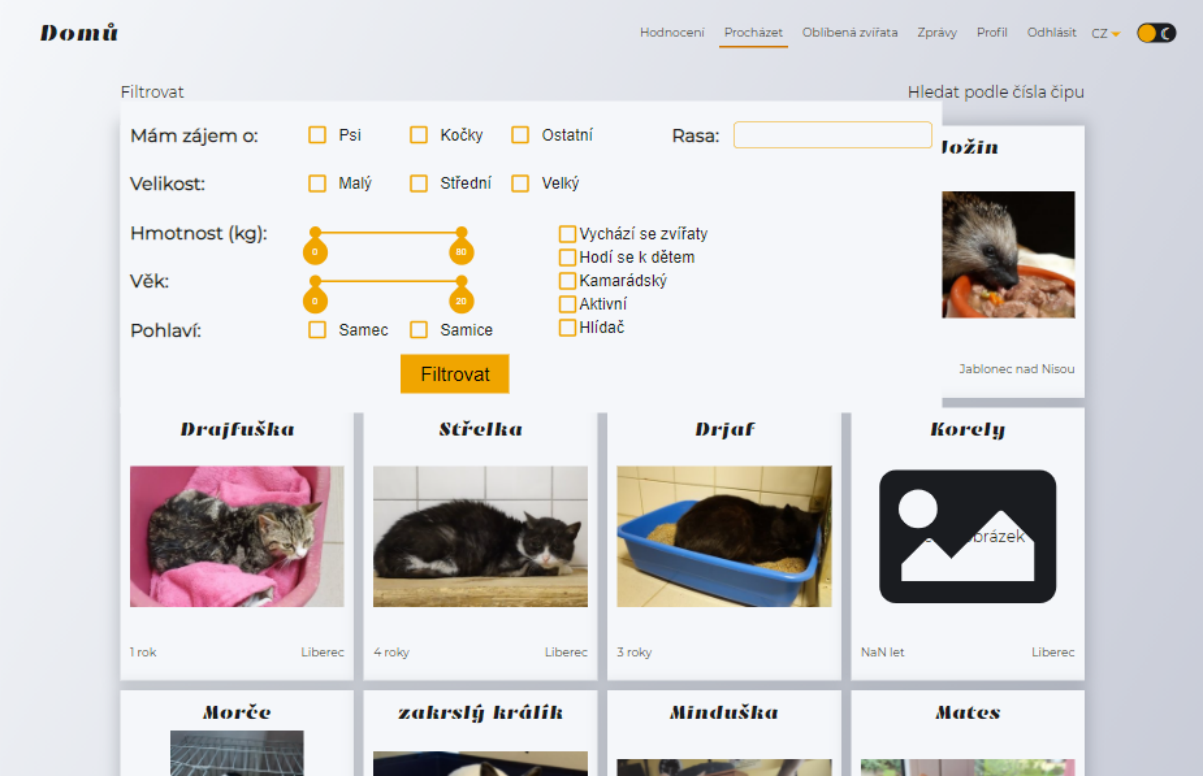
Obrázek 58 - Registrace web



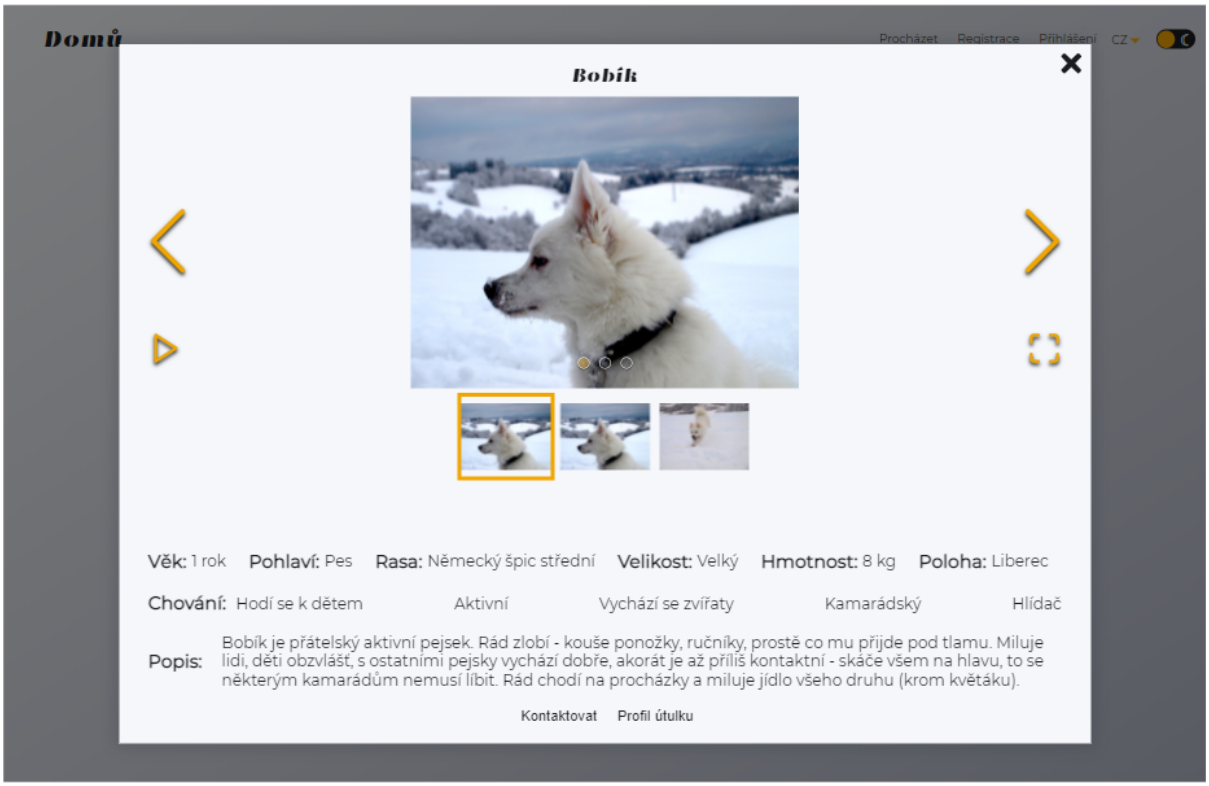
Obrázek 59 - Registrace web - validace



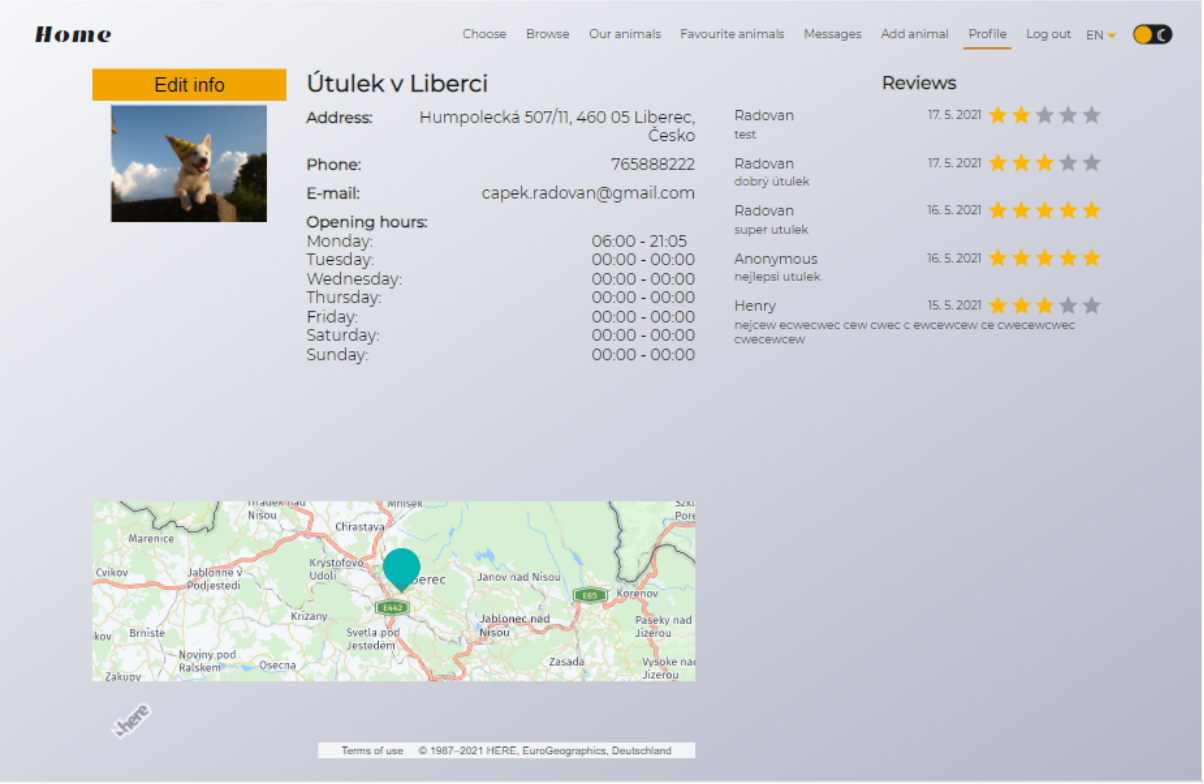
Obrázek 35 - Prohlížení zvířat



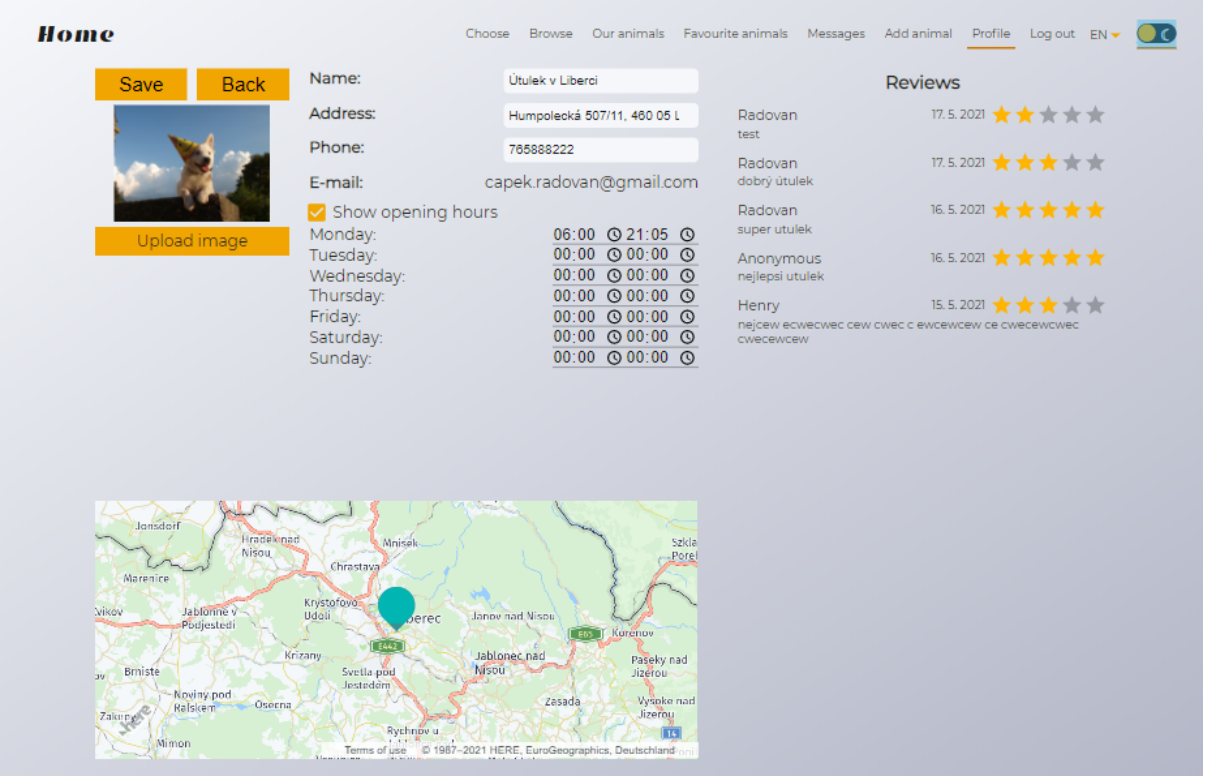
Obrázek 36 - Prohlížení zvířat - filtr



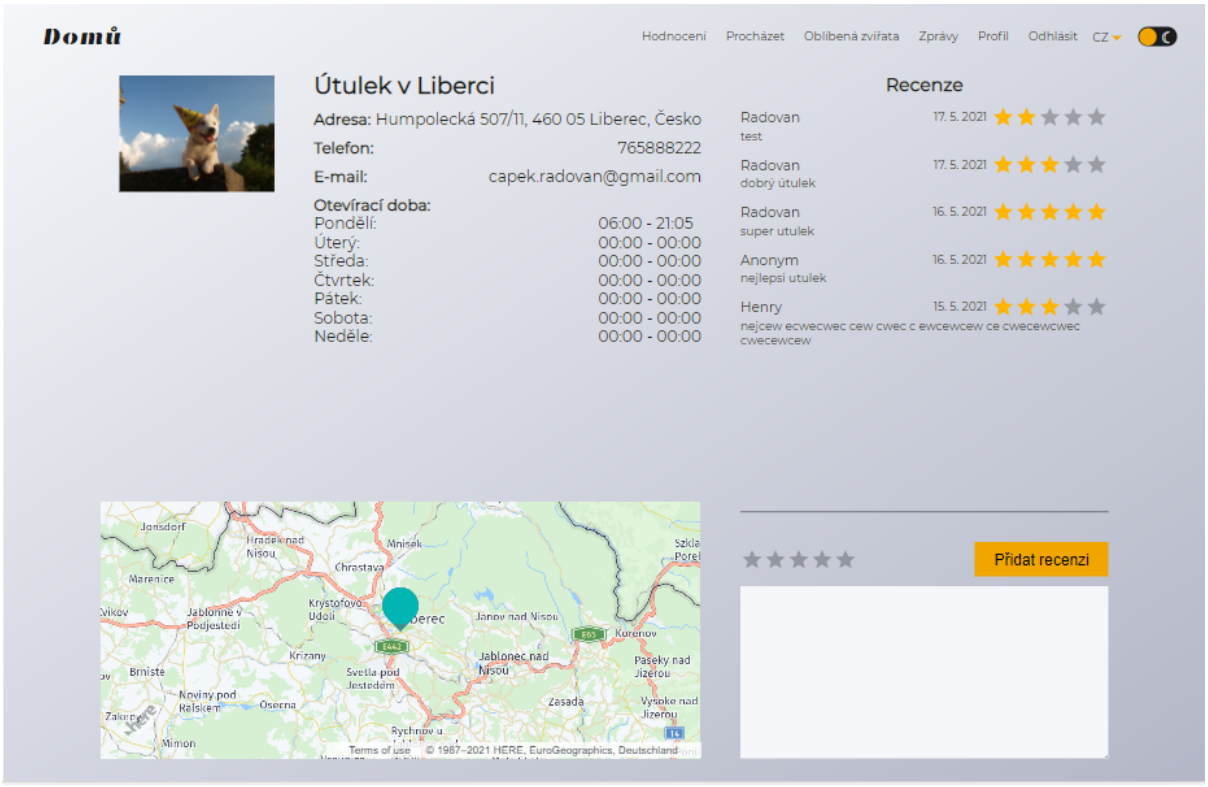
Obrázek 37 - Detail zvířete



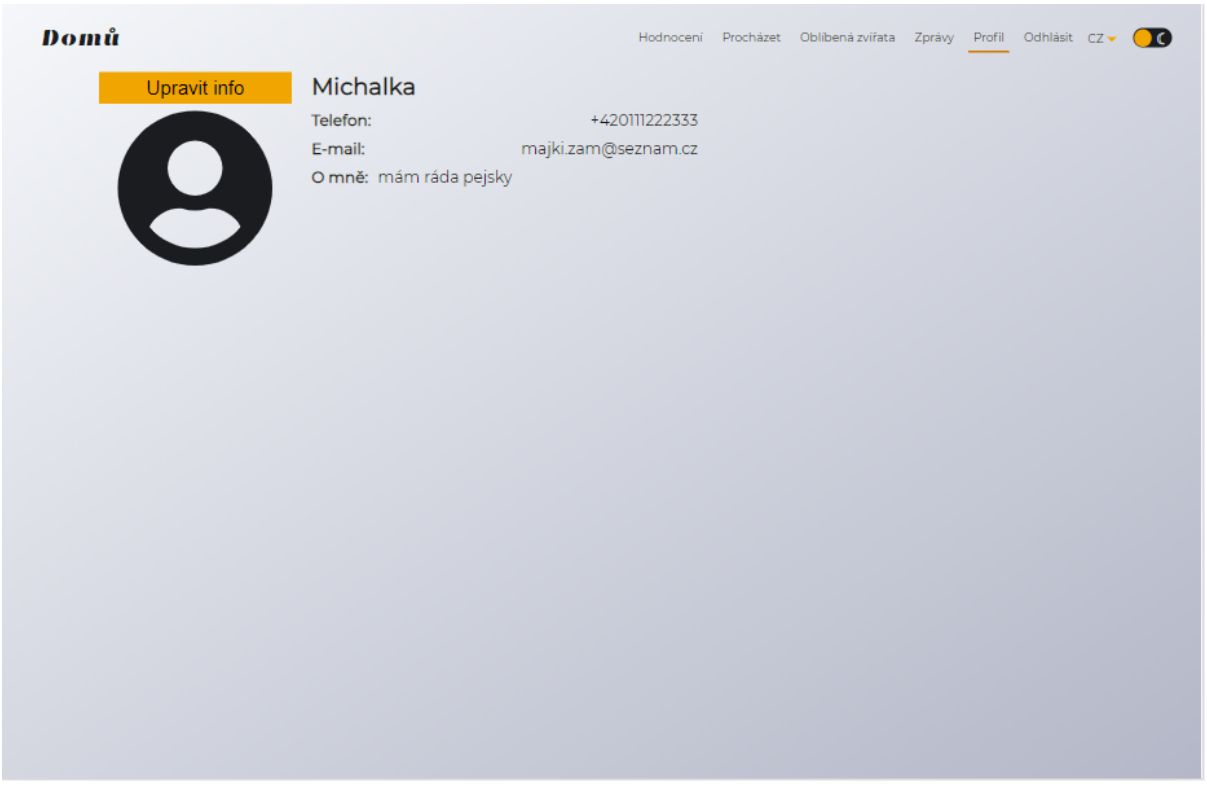
Obrázek 38 - Můj profil - útulek



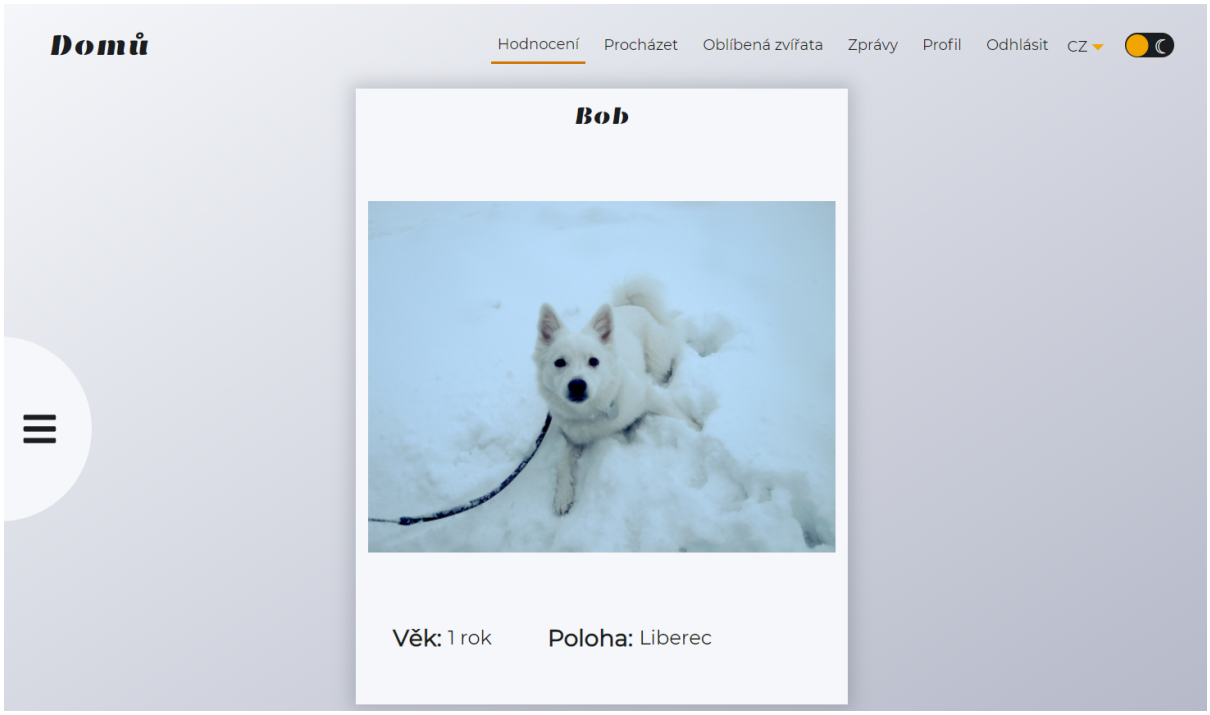
Obrázek 39 - Editace profilu



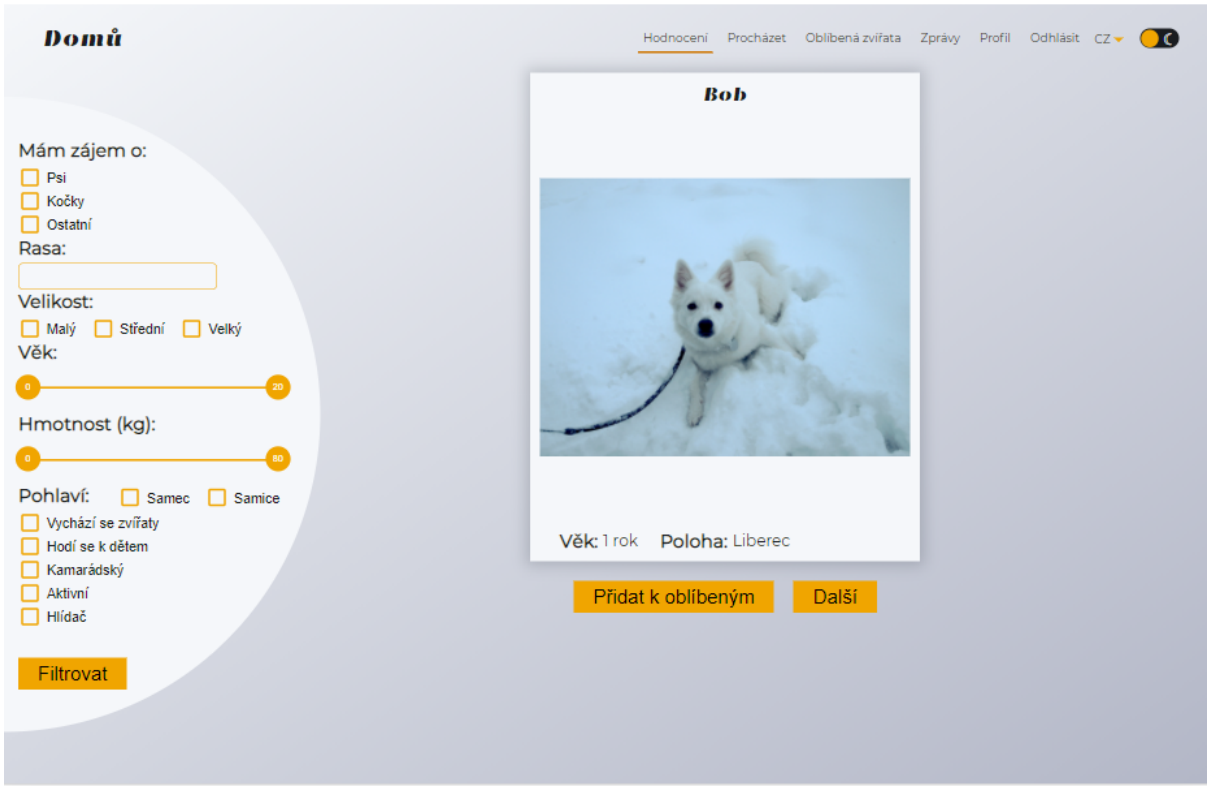
Obrázek 40 - Návštěva profilu útulku



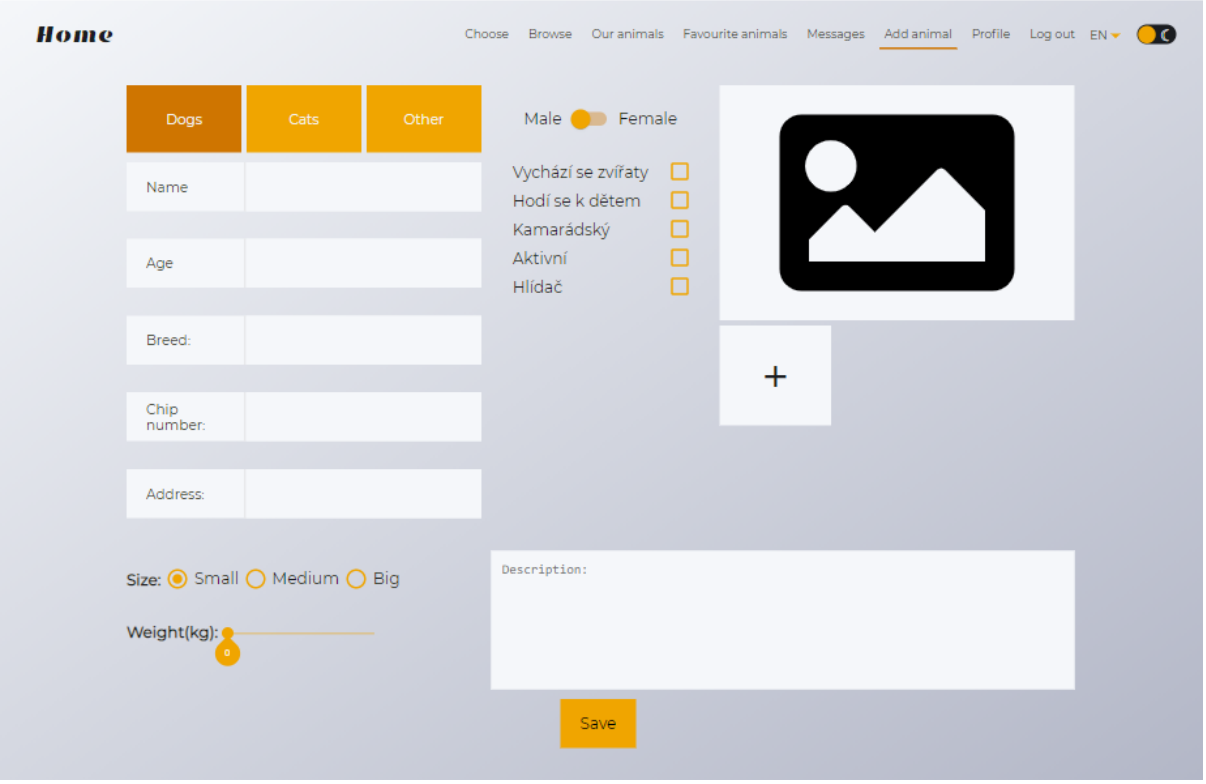
Obrázek 41 - Můj profil - zájemce



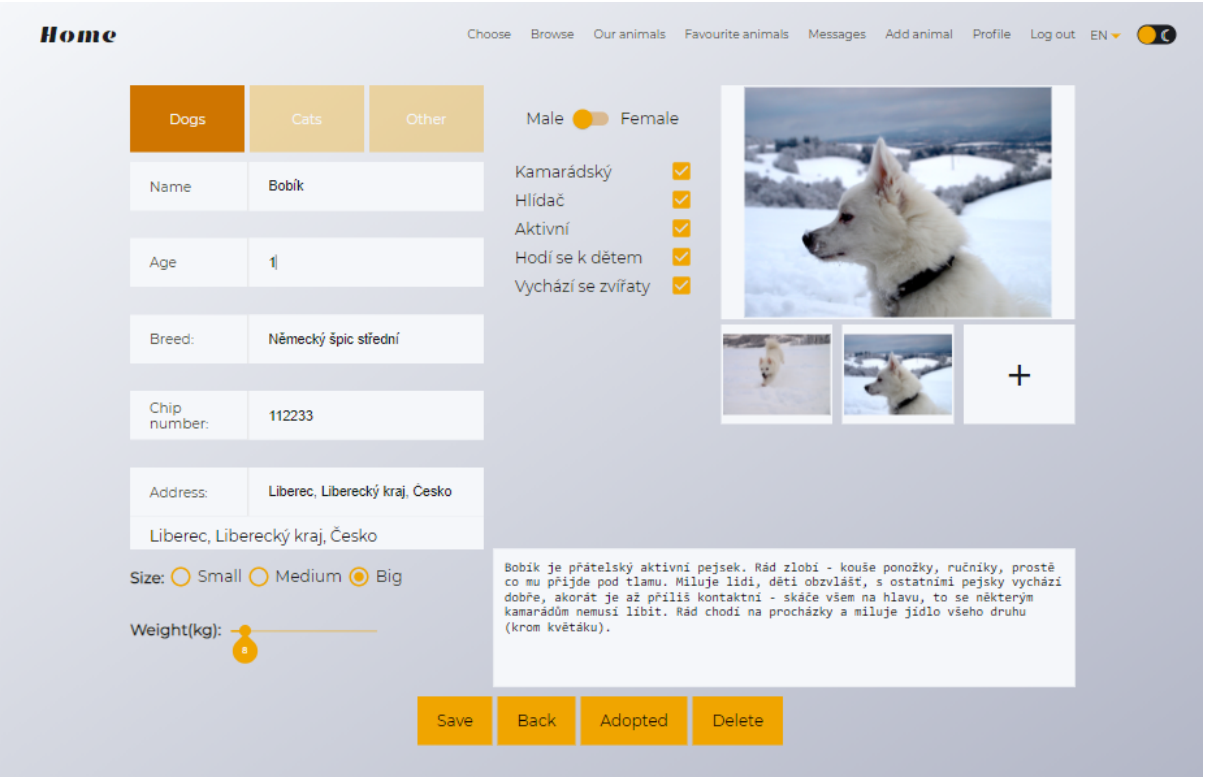
Obrázek 42 - Vybírání zvířat



Obrázek 43 - Vybírání zvířat - filtr

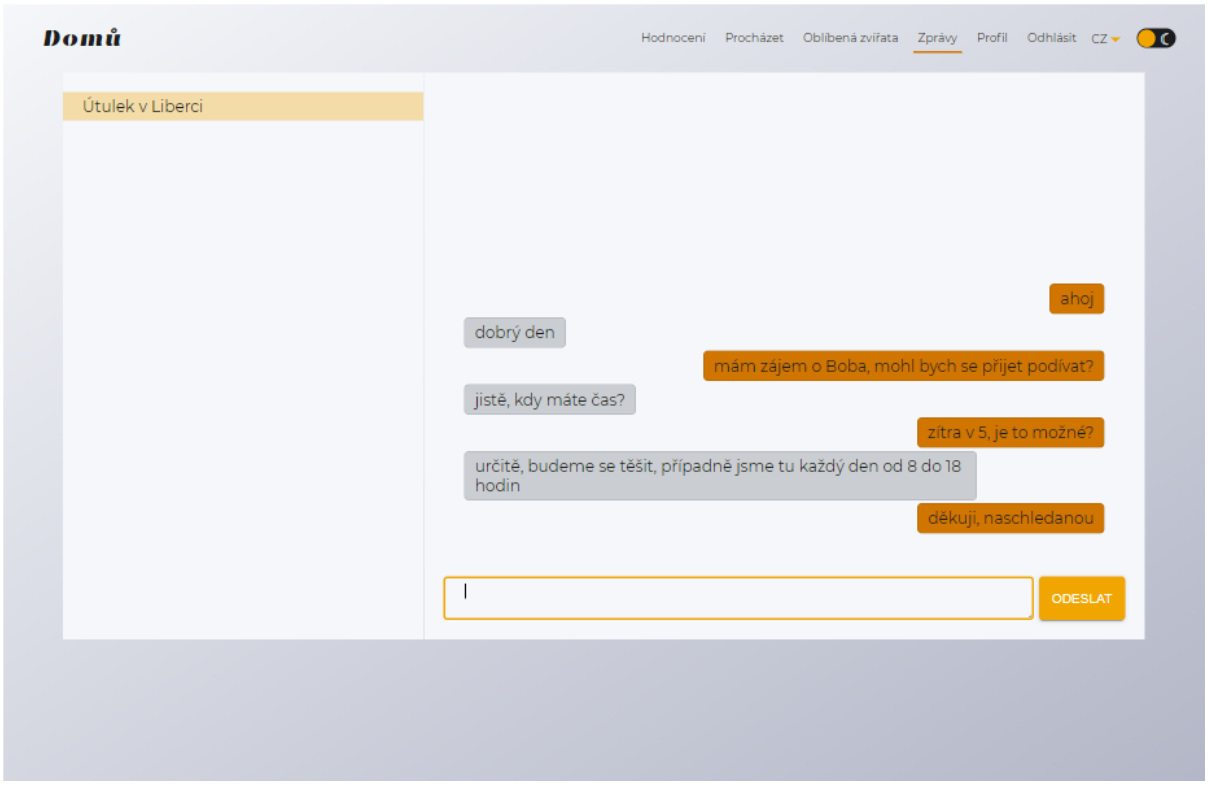


Obrázek 44 - Přidat zvíře

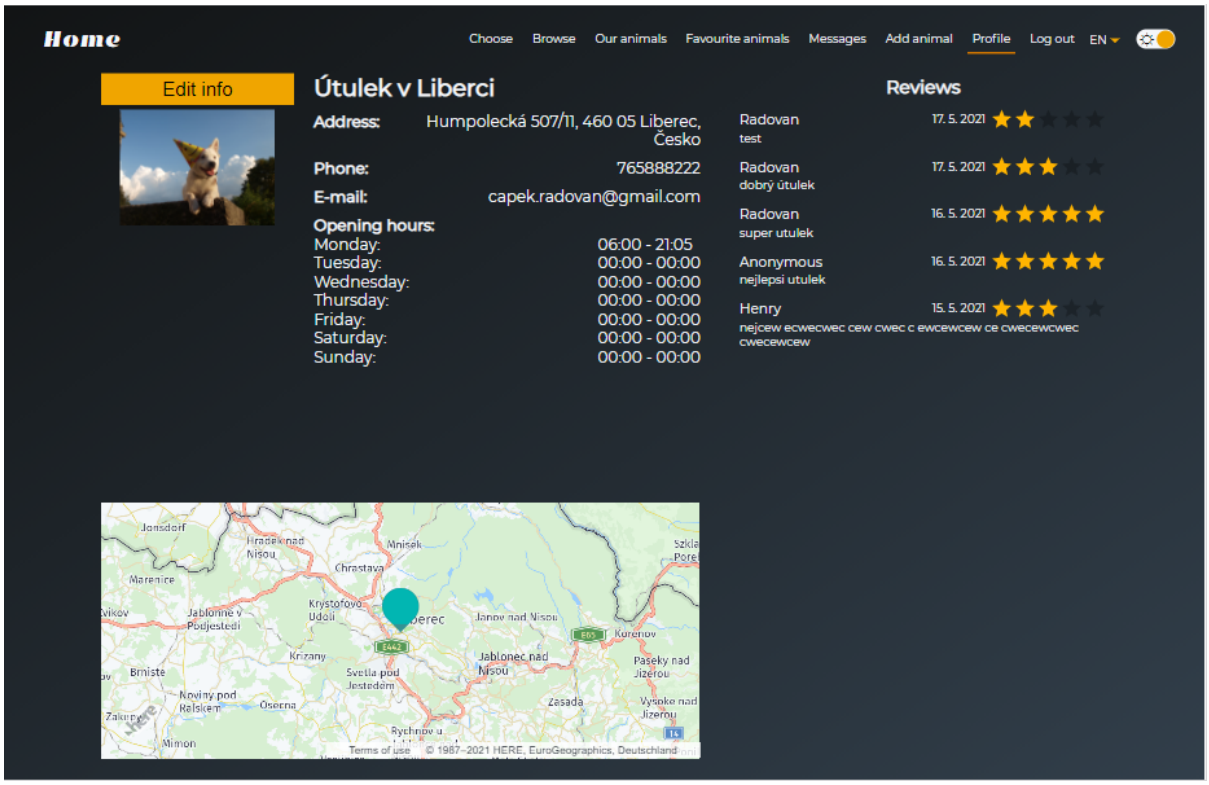


Obrázek 45 - Editovat Zvíře

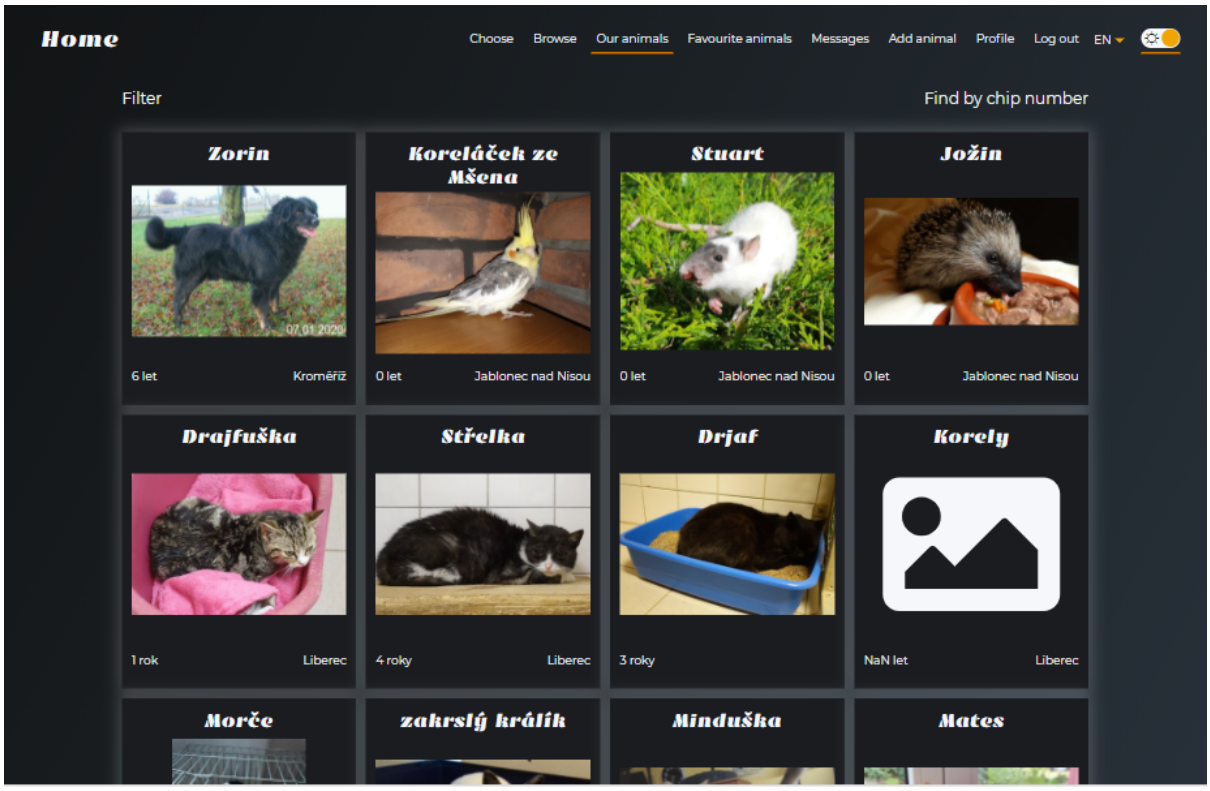
Obrázek 46 - Adoptované zvíře



Obrázek 47 - Zprávy

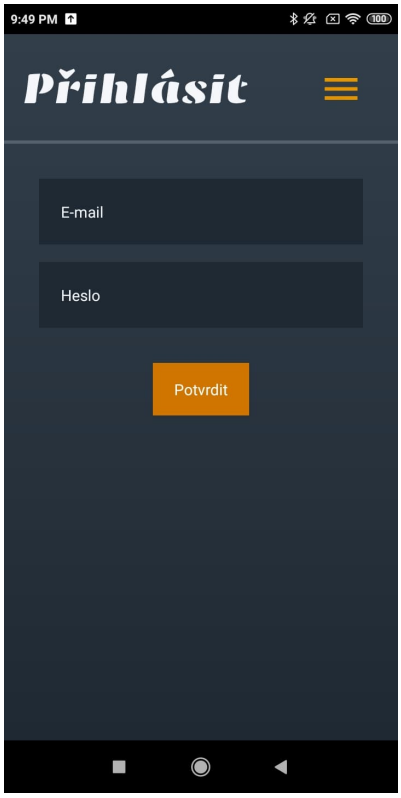


Obrázek 48 - Tmavý režim - profil

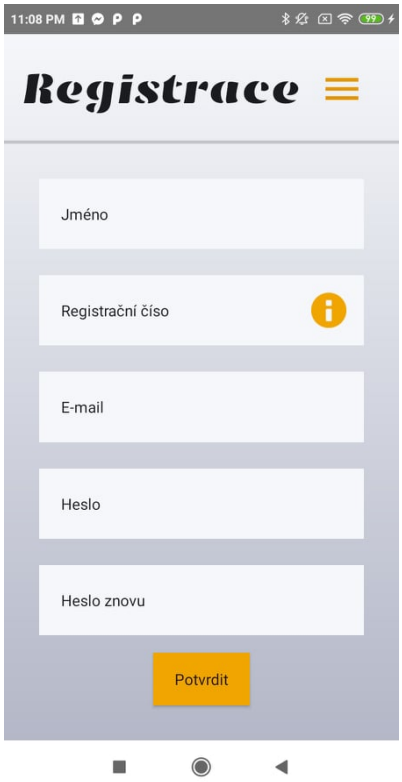


Obrázek 49 - Tmavý režim - Naše zvířata

B Obrázky mobilní aplikace

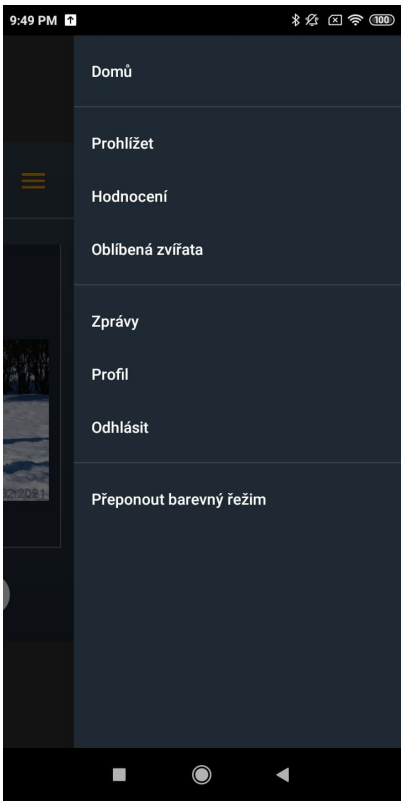


Obrázek 50 - Přihlášení

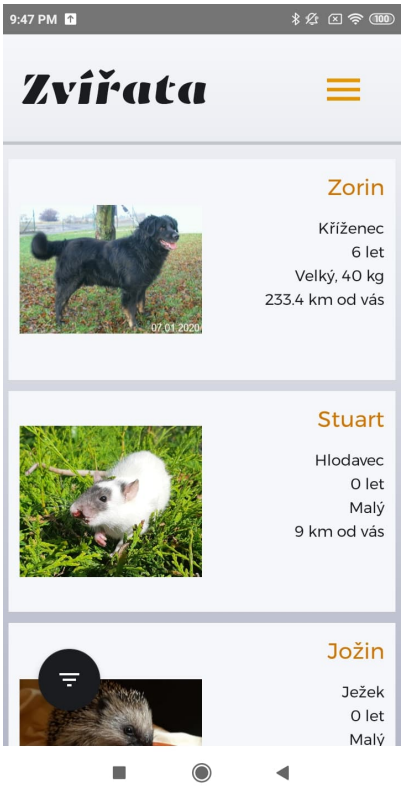


Obrázek 52 - Registrace

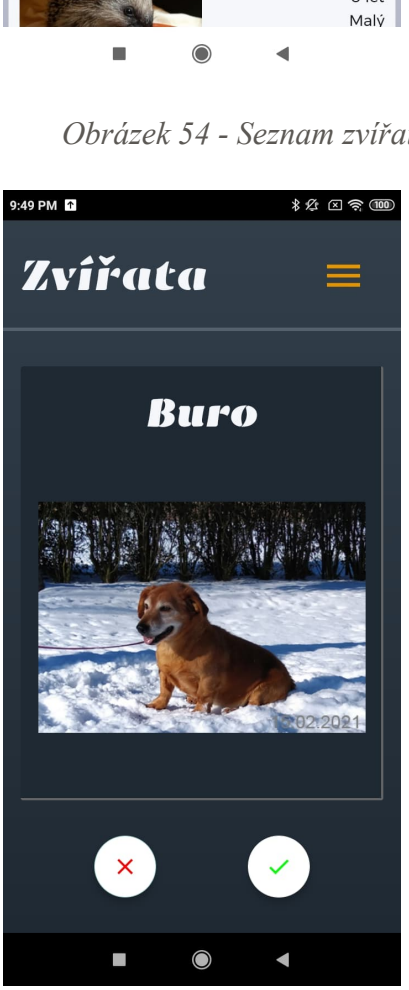
Obrázek 51 - Úvodní stránka



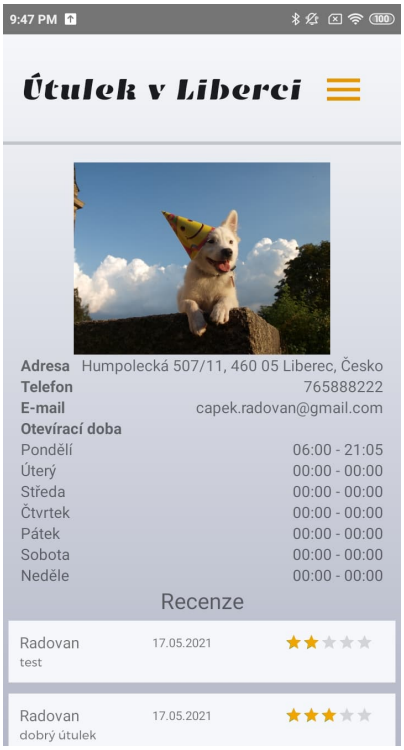
Obrázek 53 - Menu



Obrázek 54 - Seznam zvířat



Obrázek 55 - Swipe



Obrázek 56 - Profil útulku

Obrázek 57 - Detail zvířete