FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Aplikace pro chovatele psů
Student:	Bc. Tomáš Grofek
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce zimního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je navrhnout a realizovat funkční prototyp webové aplikace pro podporu chovatelů psů a pro propojení nabídky a poptávky po štěňatech. Aplikace bude poskytovat zejména tyto funkcionality:
Interaktivní průvodce uchovněním do vybraných klubů.
Evidence psů a vrhů chovatele.
Prezentace vrhů zájemcům o štěňata.
Evidence zájemců a přidělování štěňat zájemcům.
Prohlížení nabídek štěňat.
Rezervace štěněte.
Informace o rezervovaném štěněti.

Postupujte v těchto krocích:
Proveďte průzkum a analýzu požadavků cílové skupiny uživatelů.
Proveďte rešerši a analýzu podobných aplikací.
Navrhněte a implementujte serverovou část aplikace.
Navrhněte a implementujte funkční prototyp klientské aplikace pro webové prohlížeče.
Vhodnými postupy otestujte a ověřte správnost implementace.
Odstraňte zjištěné nedostatky
Zhodnoťte použitelnost výsledného prototypu aplikace, navrhněte způsob uvedení do budoucího provozu

---

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 19. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Aplikace pro chovatele psů

Bc. Tomáš Grofek

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

5. ledna 2022

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Ing. Jiřímu Hunkovi za odborné rady a vedení při psaní této práce. Mé velké díky také patří rodině, přátelům a všem ostatním, kteří mne podporovali při mém studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 5. ledna 2022 . . . . . .

Odkaz na tuto práci

Grofek, Tomáš. *Aplikace pro chovatele psů*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Práce se zabývá kompletním vývojem webové aplikace, která podporuje chovatele psů, zejména se zaměřuje na propojení poptávky a nabídky štěňat s průkazem původu. Aplikace se skládá ze serverové části poskytující REST API implementované v PHP a klientské části implementované v Javascriptu. Práce zahrnuje analýzu současných řešení a uživatelských požadavků, návrh aplikace, popis implementace a ověření správnosti implementace a návrhu. Výstupem je funkční prototyp webové aplikace.

Klíčová slova webová aplikace, Symfony, React, psi, chov psů

Abstract

The thesis deals with the complete development of a web application that supports dog breeders, especially focusing on connecting the demand and supply of puppies with a pedigree. The application consists of a server part providing REST API implemented in PHP and a client part implemented in Javascript. The thesis includes analysis of current solutions and user requirements, application design, implementation description and verification of implementation and design. The result of this thesis is a functional prototype of a web application.

Keywords web application, Symfony, React, dogs, dog breeding

Obsah

Seznam obrázků

Seznam tabulek

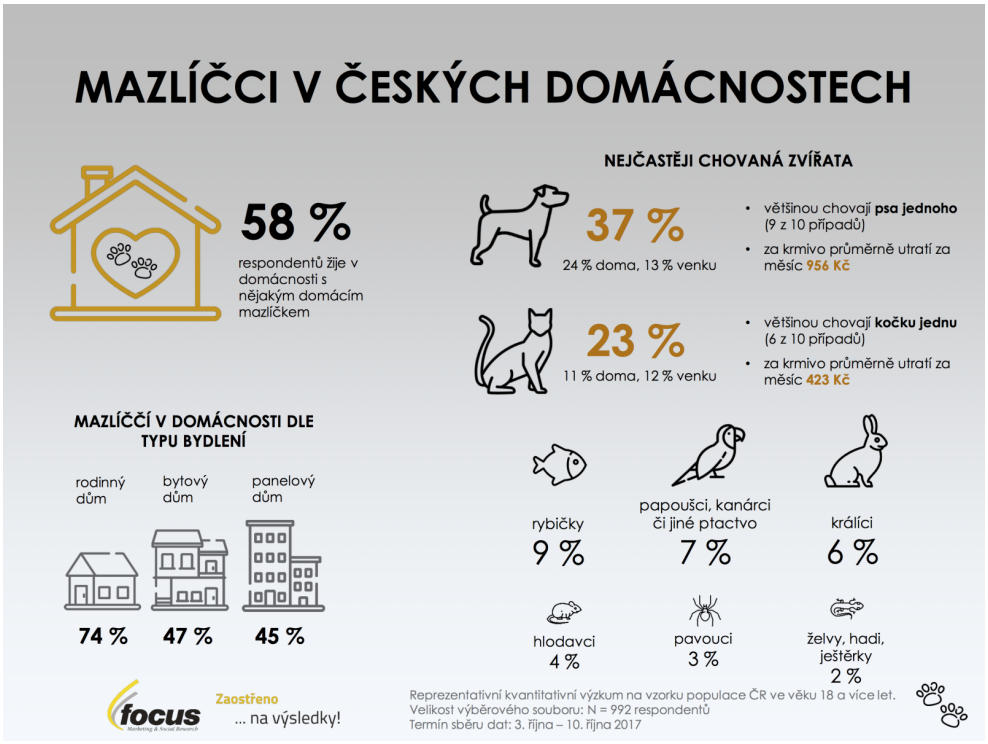
Úvod

Zvířecí mazlíčci jsou v dnešní době již neodmyslitelnou součástí většiny rodin. V roce 2017 ukázal průzkum společnosti FOCUS Marketing & Social Research, že 58 % respondentů z České republiky žije v domácnosti s domácím mazlíčkem, z nichž 37 % žije právě se psem[1]. Na obrázku 0.1 jsou graficky zpracovány hlavní výsledky průzkumu. Dle výzkumu jsou v oblibě právě psi. Při správné výchově a výcviku jsou věrnými a loajálními společníky. Vedle domácích mazlíčků zastávají psi i další důležité role. Canisterapie ve zdravotnictví, hlídání majetku a pomocníci myslivců je jen zlomek z funkcí, pro které se psi využívají.

Problematika

Postupem času se začíná více dbát na zacházení se psy. Jsou větší požadavky na životní úroveň psů, které jsou již stanoveny i pomocí zákonů. Chovatelé psů často kladou vysoké nároky na uchazeče o jejich štěňata, aby si byli stoprocentně jisti, že jimi odchovaná štěňata půjdou do dobrých rukou. Rovněž jsou kladeny nároky na pořizovaná štěňata. Velmi často je podmínkou průkaz původu psa (dále jen PP), který mimo jiné zaručuje čistokrevnost. Štěňata s PP mohou vzejít jen od chovných psů. Pro uchovnění musí pes splňovat určité podmínky, které se liší od plemene. U štěňat s PP je vysoká pravděpodobnost, že bude opět splňovat chovné standardy a pravděpodobnost genetických degenerací a dědičných chorob je minimální.

Dříve nebyl kladen na PP takový důraz a ti, kdo si přáli mít psa s PP, museli kontaktovat chovné stanice a domluvit si pořízení štěněte od nich. V dnešní době poptávka po psech s PP roste, bohužel však způsoby, jak je získat, jsou stále stejné. Najít kontakt na chovatele, kteří produkují psy s PP je poměrně obtížné, protože neexistuje globální databáze očekávaných vrhů, ani databáze chovatelů. Přehled všech chovatelů a chovných psů mají pouze jednotlivé chovatelské kluby psů, ve kterých jsou psi registrováni. Tyto data však



Obrázek 0.1: Grafické zobrazení výsledků průzkumu společnosti FOCUS Marketing & Social Research v roce 2017

[1]

často nejsou veřejná. Každý chovatel se také prezentuje jinak. Někteří mají své webové stránky, jiní se prezentují pomocí sociálních sítí. Při hledání vhodného štěněte tak zájemce musí vynaložit značné úsilí, aby kontaktoval chovatele a našel si štěně dle svých představ.

Produkování štěňat ať už s PP nebo bez PP je rovněž složitý proces. Jako první musí chovatel feny naplánovat vrh, sehnat psa ke krytí a v době hárání provést krytí. Průběžně se jezdí s fenou k veterináři na různá vyšetření. Jakmile se vše vydaří a fena porodí štěňata, musí se o ně starat přibližně 8 týdnů než mohou štěňata opustit fenu. Průběžně při těchto činnostech musí ještě vybírat vhodné zájemce, kterým následně štěňata prodá. Někteří zájemci se ozývají ještě dříve než je vrh naplánovaný, jindy je zase potřeba, aby chovatel sám nabídl štěňata tak, aby skončila v dobrých rukou.

Motivace

Na jaře roku 2020 jsem si pořizoval štěně a zjistil jsem, že když jsem v tom neměl žádné předchozí zkušenosti, bylo velmi obtížné si najít takové,

které by vyhovovalo mým požadavkům. Hledání vhodných štěňat bylo velmi náročné, jelikož neexistovala jednoduchá cesta, jak kontaktovat všechny chovatele daného plemene. Někteří měli kontakt uvedený na webových stránkách chovatelského klubu, někteří nabízeli přes vlastní webové stránky a někteří se prezentovali pouze na sociálních sítích. Zabralo mi několik týdnů, než jsem se zapsal do pořadníku u chovatele, který očekával vrh štěňat, která by byli dle mých představ a ještě neměl pořadník čítající desítky zájemců.

Uvědomil jsem si, že na trhu chybí aplikace, která by mi usnadnila najít štěně dle mých představ. Při rozhovorech s ostatními chovateli jsem se ve svém názoru utvrdil, jelikož mi dávali za pravdu. Na základě těchto skutečností jsem se rozhodl vyvinout aplikaci, která bude řešit tento problém.

Cíl práce

Cílem práce je vývoj aplikace, která usnadní zájemcům si najít štěně dle svých představ. Úspěch této aplikace bude záviset na zájmu chovatelů prezentovat se jejím prostřednictvím, takže je nutné, aby funkcionalita motivovala chovatele k jejímu využití. Toho hodlám docílit funkcionalitou, která značně usnadní chov štěňat, zejména výběr vhodných zájemců.

Součástí práce bude analýza současných řešení a uživatelských požadavků na aplikaci. Na základě analýzy poté navrhnu finální funkcionalitu aplikace. Navrhnu vhodnou architekturu aplikace a její uživatelské rozhraní. Dle návrhu poté implementuji funkční prototyp aplikace. Prototyp podrobím testům, které ověří správnost mého postupu. Závěrem zhodnotím dosažené výsledky a navrhnu další směřování aplikace.

Kapitola 1

Analýza

Nejdříve je nutné zjistit, zda je o aplikaci zájem a upřesnit její očekávanou funkcionalitu. Toho lze docílit pomocí sběru analýzy relevantních dat. Existují dvě metodiky sběru dat, a to kvantitativní a kvalitativní.

Kvantitativní metodika sestává ve sbírání mnoha malých vzorků dat a často lze provádět automatizovaně. Množství dat umožňuje analýzu zobecnit a promítnout do globálního měřítka. Její nevýhodou je velikost vzorků dat, která nelze využít k hlubší analýze problému. Kvantitativním sběrem dat je například uživatelský dotazník.

Kvalitativní výzkum je založen na sběru několika velmi detailních vzorků dat, které nelze zautomatizovat. Díky tomu je možné problém zkoumat do hloubky. Malý počet vzorků dat neumožňuje zobecnění závěrů. Příkladem kvalitativní metodiky sběru dat je řízený dialog.

Jelikož aplikace bude cílit na všechny chovatele a zájemce o štěňata s PP, zvolil jsem kvantitativní metodiku sběru dat pomocí dotazníku. Dotazníkem si přiblížím náhled chovatelů a zájemců o štěňata na současnou situaci. Výsledky vyhodnotím a poté vyhledám a zhodnotím již existující aplikace, které tuto funkcionalitu, nebo alespoň její část, řeší. Od těchto analýz očekávám potvrzení neexistence dostatečného řešení problému a potvrzení zájmu o aplikaci, která by tento problém vyřešila. Na základě výstupu analýz poté definuji funkční a nefunkční požadavky, které dále rozvinu do uživatelských požadavků.

1.1 Dotazník

Nejdříve je nutné zjistit, zda můj názor o neexistenci uspokojivého řešení nalezení psů sdílí i ostatní chovatelé a zájemci o štěňata. Toho docílím pomocí dotazníku, který nechám vyplnit potencionálními uživateli výsledné aplikace.

Od dotazníku vyžaduji aby mou domněnku potvrdil, nebo vyvrátil. Další očekávaný výsledek je upřesnění současných možností poptání a nabízení štěňat.

V neposlední řadě dotazník odhalí, zda a jaké jsou kladeny požadavky na chovatele a zájemce.

1.1.1 Návrh dotazníku

Jelikož jsem již tento problém řešil v minulosti a v oblasti chovu psů se orientuji, otázky jsem formuloval dle svých znalostí a zkušeností. Abych zabránil zkreslení výsledků mými znalostmi, přidal jsem k vybraným otázkám možnost individuální odpovědi, aby dotázaní mohli uvést odpovědi, které jsem nebral v potaz. Jelikož se pro psy často jezdí i do zahraničí, dotazník jsem napsal ve dvou jazykových mutacích: češtině a angličtině. Formulář jsem vytvořil pomocí služby Google formuláře.

Při vytváření formuláře jsem kladl důraz obzvláště na jeho jednoduchost, srozumitelnost a rychlost jeho vyplnění. Na úvod formuláře jsem uvedl cíl formuláře, jeho časovou náročnost a stručně jsem popsal jeho strukturu. Rozdělil jsem jej do dvou, respektive tří částí, z nichž vyplnění každé bylo volitelné. Celkově formulář obsahuje 13 otázek a jeho vyplnění zabere ve většině případů maximálně 5 minut v případě, že dotazovaný odpoví na všechny otázky. Kompletní přehled otázek je k vidění v příloze B.

První část je určena pro zájemce o štěňata. Skládá se z pěti otázek, jejichž cílem je zjistit aktuální způsoby poptávání štěňat, nároky na poptávaná štěňata a preference. Poslední otázka slouží k jednoduchému zjištění zájmu o aplikaci.

Druhá část otázek je určena pro chovatele, kteří nabízí nebo někdy nabízeli svá štěňata. Sestává ze sedmi otázek, které zjišťují způsoby nabízení štěňat a evidování zájemců, zda vědí jak uchovnit psa, a požadavky na případné zájemce a jejich upřesnění.

Na závěr formuláře jsem se rozhodl umožnit zanechání kontaktu, na který budu zasílat informace o průběhu a dokončení vývoje aplikace. Od této poslední „otázky“ si slibuji první potencionální uživatele, kteří mají o aplikaci zájem.

1.1.2 Výběr vhodných respondentů a rozeslání

Pro získání nejvíce relevantních odpovědí je nutno, aby dotazník vyplnila skupina potencionálních uživatelů aplikace, tedy takových, kterým záleží na kvalitě poptávaných štěňat a takových, kterým není lhostejné, komu štěně prodají. Jelikož se od potencionálních uživatelů očekává dovednost s použitím webových aplikací, rozhodl jsem se dotazník rozesílat pouze v elektronické formě.

Pro výběr vhodných respondentů jsem využil svých kontaktů na osoby, o kterých jsem věděl, že se také aktivně podílejí na chovu psů a požádal jsem je o vyplnění dotazníku. Bohužel počet odpovědí nebyl dostatečný, tak jsem se rozhodl pro zvýšení počtu odpovědí metodou pohodlného vzorkování.

Oslovení respondentů metodou pohodlného vzorkování je jednoduše proveditelné a spočívá v sdílení dotazníku ve veřejných kanálech, kde předpokládám výskyt potencionálních respondentů. Vyplnění dotazníku je dobrovolné na základě přiloženého popisu dotazníku, a proto se očekává, že odpovědi budou pouze od osob, které jsou spjaty s dotazovaným tématem.[2]

Při rešerši kontaktních kanálů se mi nepovedlo nalézt žádné aktivní internetové fórum se zaměřením na chov psů. Naopak jsem zjistil, že k tomuto účelu se často využívají specializované skupiny na sociálních sítích, zejména na Facebooku. Dotazník jsem tedy zveřejnil ve facebookových skupinách zaměřených na chov psů.

1.1.3 Vyhodnocení odpovědí

Celkem formulář vyplnilo 195 respondentů. Převážná většina z nich vyplnila dotazník v české jazykové mutaci, takže usuzuji, že většina odpovědí je od českých a slovenských chovatelů. Poměr vyplnění formulářů je zajímavý a mohl by poukazovat na to, že v české a slovenské republice bude o aplikaci mnohem větší zájem. Obecně byl počet uživatelů anglických skupin větší v jednotkách tisíců uživatelů, tudíž procento uživatelů, kteří dotazníku věnovali čas, bylo ještě menší. Naopak od česky mluvících respondentů se mi často dostávalo kladné zpětné vazby v podobě komentářů. Nicméně našlo se i pár takových, kteří v diskuzích vyjádřili obavy o přínosu aplikace. Z tohoto důvodu jsem se rozhodl hodnotit pouze výsledky formuláře v češtině.

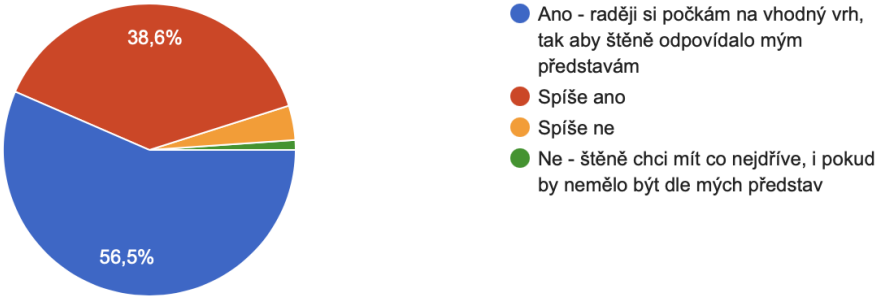
V závěrečné sekci mi zanechala kontakt přibližně třetina dotazovaných osob, což hodnotím velmi kladně. Je vidět, že o aplikaci je zájem a navíc všechny z těchto kontaktů považuji za potencionální první uživatele aplikace, kteří mi dokáží poskytnout zpětnou vazbu.

1.1.3.1 Sekce pro zájemce o štěňata

První otázka byla formulována tak, abych získal přehled současných způsobů vybírání štěňat. Jako předpřipravené odpovědi jsem použil čtyři dle mého názoru nejběžnější způsoby a nechal jsem možnost přidání odpovědi, abych případně zjistil další způsoby. Každý respondent mohl zvolit více odpovědí.Na základě odpovědí je zřejmé, že většina zájemců (68 %) poptává přímo u chovatelů, další nezanedbatelné množství poptává na internetových inzerátech (44 %) a sociálních sítích(39%). Z útulku si psy pořídilo nebo plánuje pořídit přibližně 14 % dotázaných. Mezi volnými odpovědmi se několikrát objevilo poptání u známých. Výsledky této otázky ukazují na vysoký rozptyl poptávání štěňat a vidím zde potencionální přínos aplikace, která by nabídky sjednotila.

Následující tři otázky byly konstruovány tak, aby potvrdily, nebo vyvrátily mou domněnku, že zájemci mají vysoké nároky na poptávané psy. Odpovědi mou domněnku potvrdily. Na otázku „Jste ochotni čekat na štěně dle vašich představ?“ odpovědělo více než 90 % respondentů kladně viz. obrázek 1.1.

U hodnocení kritérií štěňat téměř vždy převládaly odpovědi „důležité“ a „spíše důležité“. Výjimkou pak byla kritéria „vzdálenost majitele od mého bydliště“, kde byla většina odpovědí „nedůležité“ a „spíše nedůležité“. U kritéria „pořizovací cena“ poté odpovědi připomínaly Gaussovo normální rozdělení. Ve volné odpovědi na doplnění dalších důležitých kritérií otázky spíše rozvíjely kritéria z předchozí otázky.



Obrázek 1.1: Uživatelský dotazník - Odpovědi na otázku „Jste ochotni čekat na štěně dle vašich představ?“

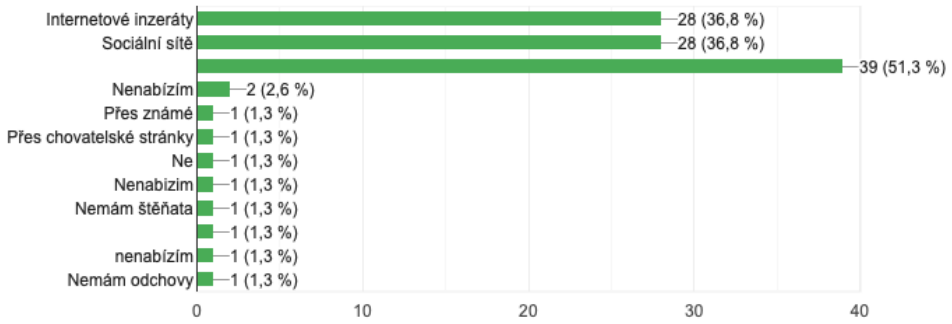
Na závěrečnou otázku sekce pro zájemce, zda by byl zájem o aplikaci, která by usnadnila hledání štěňat, odpovědělo přibližně 90 % dotazovaných kladně. Na základě odpovědí je zřejmé, že nabídky štěňat nejsou sjednocené a je náročné si najít vhodné štěně mezi všemi současnými nabídkami. Rovněž průzkum ukázal, že je zájem o aplikaci, která by nabídky sjednotila a zjednodušila.

1.1.3.2 Sekce pro nabídku štěňat

Úvodní otázka této sekce jednoduše zjišťovala, zda respondenti dokáží uchovnit psa. Přestože většina odpověděla kladně, stále nezanedbatelných 25 % dotázaných neví, jak uchovnit psa. Výsledná aplikace bude mít potenciál tohle procento snížit.

Druhá otázka se ptala na formu nabízení štěňat, měla tři předpřipravené odpovědi, možnost otevřené odpovědi a respondenti mohli zvolit i více odpovědí. Zde 51 % odpovědí obsahovalo volbu „Nemusím nabízet, zájemci mě kontaktují sami“. Volby „Internetové inzeráty“ a „Sociální sítě“ obsahovalo shodně 37 % odpovědí. Ve volných odpovědích se volba „Nenabízím“ objevila v 10 % odpovědí. Procentuální zastoupení všech odpovědí může čtenář vidět na obrázku 1.2. Tyto odpovědi poukazují na to, že nezanedbatelná část chovatelů nenabízí, nebo nemusí nabízet své štěňata. V návrhu aplikace bude nutné tohle zohlednit a navrhnout funkcionalitu tak, aby byla pro tyto chovatele

atraktivní a měli motivaci ji využívat, i když momentálně nemusí vynakládat úsilí pro udání svých štěňat.



Obrázek 1.2: Uživatelský dotazník - Odpovědi na otázku „Jakou formou nabízíte štěňata?“

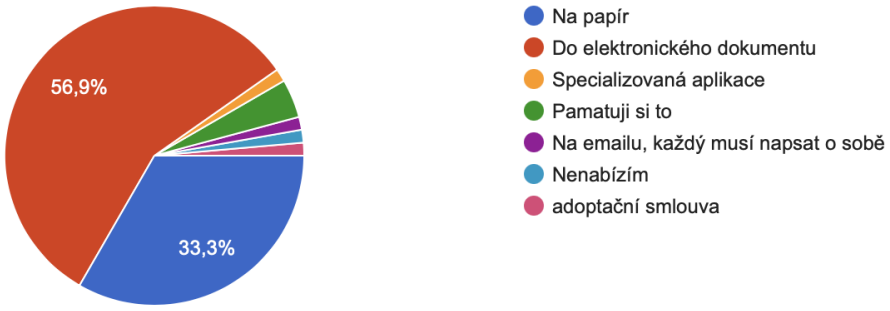
Následující tři otázky měly za úkol potvrdit, nebo vyvrátit mou domněnku, že chovatelé mají vysoké nároky na zájemce o jejich štěňata. Zde 96 % respondentů uvedlo, že jim záleží na tom, komu štěňata prodají. U hodnocení jednotlivých kritérií nikdo neuvedl, že by kritérium bylo nedůležité. Drtivá většina odpovědí měla medián ve volbě „Důležité“ a ty zbylé měly medián „Spíše důležité“. V doplňující volné odpovědi na další kritéria bylo zmíněna spousta odlišných a různorodých kritérií. Tyto odpovědi dokazují, že naprostá většina dotázaných chovatelů má vysoké nároky na zájemce o jejich štěňata.

Předposlední otázka zkoumala současné způsoby evidence zájemců. Zde 57 % zájemců uvedlo možnost „Do elektronického dokumentu“ a třetina možnost „Na papír“. Pouze jeden jediný dotázaný uvedl jako současný způsob evidence zájemců specializovanou aplikaci. Přehled četností odpovědí na tuto otázku je zobrazen na obrázku 1.3. Odpovědi poukázaly na to, že i přes vysoké nároky kladené na zájemce, neexistuje hojně využívaná specializovaná aplikace, která by tuto problematiku usnadnila. Zde vidím velký potenciál ve výsledné aplikaci.

Závěrečná otázka, zjišťující zájem o aplikaci, která by usnadnila prodej štěňat, měla převážně kladné odpovědi (79 %). Avšak 21 % záporných odpovědí nelze zanedbat a bude potřeba je zohlednit při formulaci uživatelských požadavků.

1.2 Rešerše a analýza podobných aplikací

Jelikož výsledná platforma je webová aplikace, při rešerši jsem se zaměřil převážně na webové stránky a aplikace řešící tuto problematiku. Při hodnocení jsem se zaměřil převážně na funkcionality aplikací spojené s chovem. Pro nalezení těchto aplikací jsem použil webový vyhledávač Google, který je v



Obrázek 1.3: Uživatelský dotazník - Odpovědi na otázku „Jak si evidujete zájemce o štěňata?“

současné době nejvíce používaný. Stránky jsem hledal pomocí následujících frází, které dle mého názoru nejvíce popisují vyvíjenou aplikaci. Fráze jsem zvolil následovně: „nabídka štěňat“, „prodej štěňat“, „chov psů“, „aplikace pro chov psů“, „evidence zájemců o štěňata“, „evidence psů“. A jejich anglické ekvivalenty. Rovněž jsem se při hledání omezil jen na prvních 10 výsledků vyhledávání.

1. **Vyhledávání štěňat**

   Tato funkcionalita je základem pro uspokojení náročných zájemců o štěňata. Dle dotazníku většina z nich má na štěně nemalé nároky a pro takové uživatele je jednoduché vyhledávání podle pár kritérií nedostačující.

2. **Prezentace štěňat**

   Aby chovatelé mohli nabídnout svá štěňata náročným zájemcům, je třeba je prezentovat v co možná nejlepším světle.

3. **Evidování psů**

   Každý chovatel potřebuje mít o svých psech přehled, který je s přibývajícím počtem psů stále složitější zachovat.

4. **Prověřování a evidování zájemců o štěňata**

   Dotazník ukázal, že i na zájemce o štěňata jsou kladeny nároky,. Proto by správná aplikace měla pomoci ověřit zájemce a případně odfiltrovat ty nežádoucí, čímž by uživatelům ušetřila spousty času. Také se při analýze neprokázala existence aplikace, která by usnadnila evidenci zájemců.

5. **Jiné funkcionality využitelné pro chov psů**

   Jelikož se nenašla aplikace, která by poskytovala všechny požadované funkcionality, jedná se vždy o pouhé podmnožiny těchto funkcionalit.

Nicméně je možné, že tyto aplikace budou mít i jiné funkcionality pro podporu chovu psů, než na které se v této práci zaměřuji.

1.2.1 Webfordog

Webfordog.cz[3] je jediný český zástupce mezi analyzovanými weby. Jak již název napovídá, je výhradně určen pouze pro psy. Slouží nejen pro inzerci poptávky a nabídky štěňat nebo starších dospělých psů, ale i jako databáze informací o psech. Ve vyhledávači Google jej nebylo obtížné najít, protože se vyskytuje na prvních stránkách vyhledávání pro fráze „nabídka štěňat“ a u podobných frází dosahuje podobných vyhledávacích výsledků. Nicméně neobsazuje úplně první příčky vyhledávání, kde jej předstihují běžné inzertní portály, u kterých je inzerce psů pouze malou podmnožinou inzerátů. Web nedisponuje dostatečnou responsivitou zobrazení, a proto je v dnešní době, kdy většina uživatelů prohlíží weby na telefonech, obtížně použitelný.

1. **Vyhledávání štěňat**

   Vyhledávání nabídek štěňat obsahuje filtr, který umožňuje určit parametry hledaných štěňat. Vyhledávání je zobrazeno na obrázku 1.4. Inzerce je možná pouze pro chovatele s lokalitou z České a Slovenské republiky. Důvěryhodnost vyhledávání je přinejmenším pochybná, jelikož si inzerující mohou zaplatit lepší umístění ve vyhledávání.

2. **Prezentace štěňat**

   K prezentaci štěňat lze využít pouze titulek, textový popis, 0 - 6 fotografií. Samozřejmostí u inzerátu jsou stáří pejska, datum odběru, plemeno, cena, pohlaví a kontaktní informace. Jelikož web slouží i jako databáze plemen, inzeráty jsou propojeny na detailní popisy plemene, kde si potencionální zájemci mohou o plemeni zjistit důležité informace.

3. **Evidování psů**

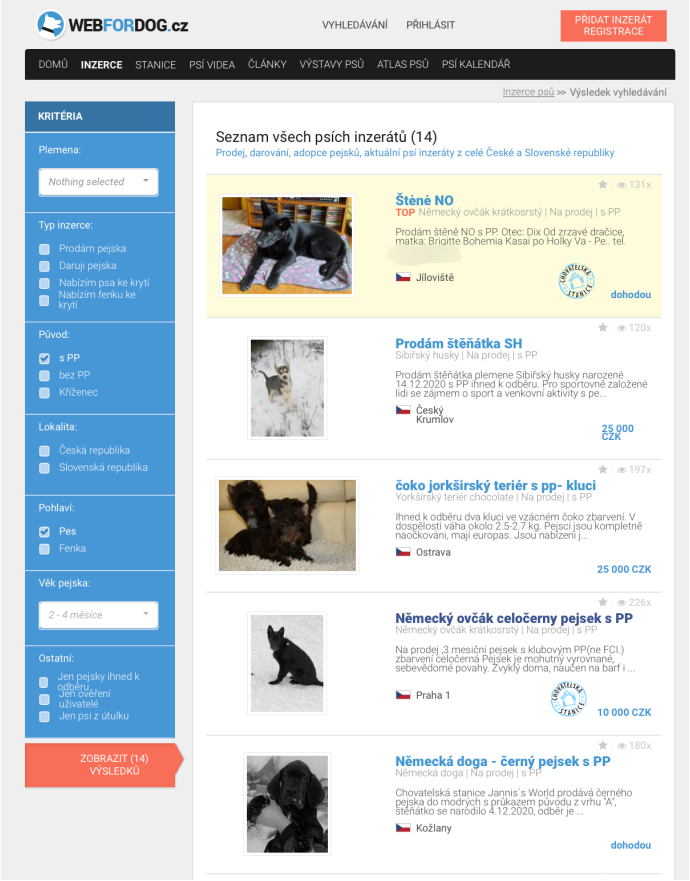
   Tuto funkcionalitu Webfordog neposkytuje.

4. **Prověřování a evidování zájemců o štěňata**

   Tuto funkcionalitu Webfordog neposkytuje.

5. **Jiné funkcionality využitelné pro chov psů**

   Jak jsem se již zmínil, web funguje mimo inzercí i jako databáze informací o psech. Lze zde najít podrobný atlas plemen s detailními popisy. Dále web obsahuje databázi chovatelských stanic, která ale není zdaleka úplná, a kalendář psích jmen. Velmi kladně hodnotím sekci videí, která obsahuje mnoho video-návodů užitečných při chovu a výcviku psů a také sekci článků, kde lze nalézt velmi zajímavé tipy, návody a informace spojené se psy.



Obrázek 1.4: Ukázka Webfordog - Vyhledávání štěňat

1.2.2 Puppyfind

Server puppyfind.com[4] je jednoduchým portálem pro prodej a nabídku štěňat. Je určen pro chovatele ze Spojených států amerických. Web nedisponuje dostatečnou responsivitou zobrazení. Dle mého názoru je web velmi nepřehledný a složitý pro orientaci.

1. **Vyhledávání štěňat a prezentace štěňat**

   Web obsahuje velmi základní filtr, který je velmi neintuitivní, protože jednotlivé položky filtru se jeví jako filtr pro nabídky štěňat, avšak každá filtruje něco jiného a položky nelze kombinovat. První komponenta filtru je fulltextové vyhledávání, které po vyplnění vyhledá plemena psů, dle kterého lze přejít na detail plemene s jeho popisem a až ze stránky plemene se lze dostat na nabídky štěňat daného plemene. Druhá položka je filtrování dle státu, která po zvolení vyhledá konkrétní nabídky psů. Třetí položka umožňuje filtrovat dle velikosti, náročnosti a

účelu pořízení. Po vyplnění této položky se objeví výsledky vyhledávání mezi plemeny. Jednotlivé výsledky vyhledávání postrádají další možné možnosti filtrování a jediné, jak lze mezi nimi efektivně procházet, je možnost změny jejich řazení. Tento způsob procházení nabídek je naprosto nedostačující. Procházení nabídek pro zvolené plemeno lze vidět na obrázku 1.5.

2. **Prezentace štěňat**

   Prezentace jednotlivých štěňat obsahuje pár základních detailů jako jsou jméno, textový popis, fotografie, datum narození, pohlaví a zda je z vrhu šampionů.

3. **Evidování psů**

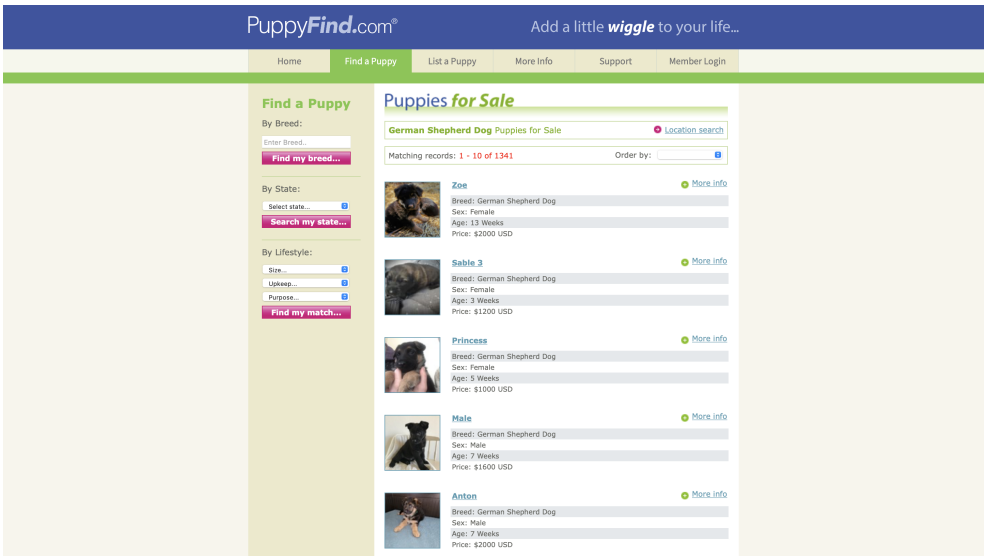
   Tuto funkcionalitu Puppyfind neposkytuje.

4. **Prověřování a evidování zájemců o štěňata**

   Tuto funkcionalitu Puppyfind neposkytuje.

5. **Jiné funkcionality využitelné pro chov psů**

   Jedinou doplňující funkcionalitou je databáze plemen psů s jejich popisy.



Obrázek 1.5: Ukázka Puppyfind - výpis nabídek pro zvolené plemeno

1.2.3 Puppyfat

Puppyfat je určen pouze pro chovatele psů. Prestože je aplikace prezentována pomocí webové stránky puppyfatapp.com[5], ve skutečnosti se jedná o mobilní

aplikaci, která je ke stažení na Google play[6] a Appstore[7]. Puppyfat slouží jako databáze chovatelových psů, vrhů a štěňat. Tato aplikace bohužel není dostupná zdarma a za její využívání si účtuje roční poplatek 99,99 USD.

1. **Vyhledávání štěňat**

   Tuto funkcionalitu Puppyfat neposkytuje.

2. **Prezentace štěňat**

   Tuto funkcionalitu Puppyfat neposkytuje, ale na webových stránkách je mezi vyvíjenými funkcionalitami uveden export vrhů do PDF, což usnadní prezentaci štěňat.

3. **Evidování psů**

   Puppyfat poskytuje mocné nástroje pro evidování psů. Samozřejmostí je evidence základních dat jako například fotografie, jméno, datum narození , číslo čipu, atd. Editaci základních údajů lze vidět na obrázku 1.6. Mimo to umožňuje také ke každému psovi evidování podrobných záznamů, které lze přidávat na denní bázi. Příkladem takovýchto záznamů je například krmení - kdy proběhlo, co pes snědl, jaké množství, od jakého výrobce, atd. Další podrobné záznamy jsou například váha psa, veterinární záznamy, záznamy o krytí a mnohé další. Tyto záznamy poté lze přehledně zobrazit v grafech.

4. **Prověřování a evidování zájemců o štěňata**

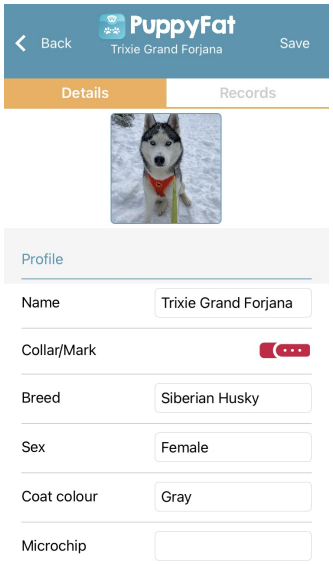
   Puppyfat poskytuje funkcionalitu evidování kontaktů souvisejících s chovem. V této funkcionalitě lze mimo jiné přidat kontaky na zájemce o štěňata, přiřadit jim jednotlivá štěňata a také si o kontaktech vést poznámky.

5. **Jiné funkcionality využitelné pro chov psů**

   Funkcionalita evidování kontaktů umožňuje evidovat lékaře, chůvy nebo například jiné chovatele.

1.2.4 Dogbreederpro

Dogbreederpro.com[8] je webová aplikace sloužící jako komplexní evidenční systém psů chovatele. Aplikace disponuje responzivním webdesignem a funkcionalitami podporujícími chovatele psů, kteří si zde mohou vést detailní dokumentaci stavu svého chovu. V rozcestníku aplikace lze přehledně vidět všechny nadcházející události, ať už je to hárání feny nebo plánované krytí. Aplikace je dostupná ve dvou verzích (základní a rozšířená) za měsíční poplatek 4,99 USD, respektive 9,99 USD.



Obrázek 1.6: Ukázka aplikace Puppyfat - evidování základních údajů psa

1. **Vyhledávání štěňat**

   Aplikace nedisponuje funkcionalitou pro vyhledávání štěňat.

2. **Prezentace štěňat**

   Pro prezentaci psů lze využít exportu do formátu PDF nebo CSV. Rovněž jde exportovat pouze informace nesoucí informace nutné pro chov nebo zdravotní záznamy.

3. **Evidování psů**

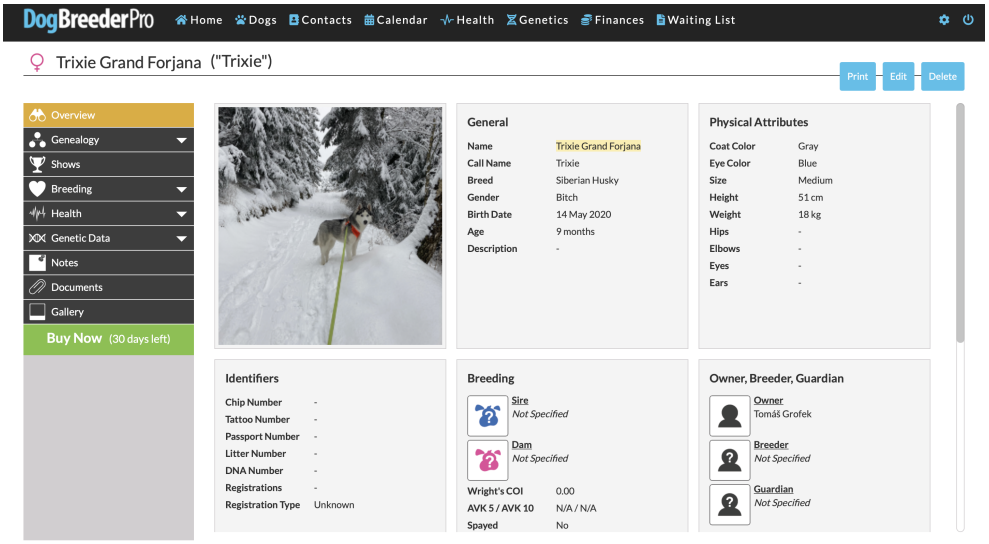
   Dobrgeederpro umožňuje komplexní evidenci psů, kde u každého psa lze evidovat základní parametry, zdravotní informace, genetické informace, rodokmen a mnoho dalšího. Data lze poté prohlížet v přehledné formě. Ukázku profilu psa lze vidět na obrázku 1.7 .

4. **Prověřování a evidování zájemců o štěňata**

   Aplikace umožňuje spravovat pořadník zájemců k jednotlivým vrhům a štěňatům. Přidat zájemce lze ještě před naplánováním vrhu a je možné evidovat jejich požadavky na štěně. Následně po porodu lze štěňata přiřadit jednotlivým zájemcům.

5. **Jiné funkcionality využitelné pro chov psů**

   Za zmínku stojí rozhodně genetická analýza, kde na základě rodokmenů aplikace dokáže vyhodnotit a předpovědět genetické předpoklady a deformace. Lze také evidovat navštívené výstavy a jejich výsledky, psát si poznámky nebo vést kalendář.



Obrázek 1.7: Ukázka aplikace Dogbreederpro - detailní profil psa

1.2.5 Výsledky rešerše a analýzy

Rešerše a následná analýza potvrdily mou domněnku o neexistenci aplikace, která by usnadňovala chov a evidenci psů a zároveň ji propojovala s poptávkou po štěňatech. Existující aplikace umí buď to pouze inzerovat štěňata a psy, nebo je evidovat a zjednodušovat jejich chov. Souhrn podporovaných funkcionalit je zaznamenán v tabulce 1.1. Z výsledků dotazníku je navíc zřejmé, že o takovou aplikaci je zájem.

Analyzované aplikace pro chov psů sloužily pouze jako databáze údajů, které si uživatel musel do aplikací vložit vlastnoručně. Rovněž inzertní stránky byly dosti omezené. Zde vidím obrovský potenciál sdílené aplikace, která by mohla využívat dat z obou hlavních funkcionalit, a tak uživatelům značně ulehčit práci při zadávání údajů. Inzertní aplikace byly omezeny pouze na několik regionů, což je další značná nevýhoda, jelikož v dnešní době se pro psy jezdí i stovky kilometrů do zahraničí.

Tabulka 1.1: Přehled funkcionalit analyzovaných aplikací

Funkcionalita	Webfordog	Puppyfind	Puppyfat	Dogbreeder
Vyhledávání štěňat	✓	✓	—	—
Prezentace štěňat	✓	✓	—	✓
Evidování psů	—	—	✓	✓
Evidování zájemců	—	—	✓	✓
Jiné funkcionality	✓	✓	✓	✓

1.3 Požadavky na aplikaci

Pohled na aplikaci může být velmi rozdílný v závislosti na osobě, která požadavky popisuje. Pohledy koncových uživatelů na funkcionalitu jsou často odlišné od pohledů designerů aplikace nebo programátorů. Popisy funkcionalit se mohou často lišit i pokud v podstatě popisují tu stejnou funkcionalitu. To nastává například z důvodů odlišně používaných terminologií nebo rozdílné míry abstrakce popisu. Z tohoto důvodu je na začátku projektu nutné definovat požadavky na aplikaci tak, aby byly jednoznačné pro všechny zainteresované strany. Vymezení požadavků často může usnadnit mnoho úsilí a vynaložených prostředků, protože díky evidenci požadavků jde odhalit postupy, které jsou prováděny navíc.[9]

Na základě provedených analýz a rešerší jsem identifikoval a formuloval požadavky na aplikaci. Jedná se o seznam základních požadavků na aplikaci, které vymezí její obsah a stručně popíší její funkcionality a vlastnosti. Tento seznam poté bude sloužit jako kontrola správnosti návrhu a implementace, kde lze jednoduše sledovat, zda aplikace splňuje všechny definované požadavky. Proto by požadavky měly být jednoznačně a co možná nejsnadněji ověřitelné. Tyto požadavky se dělí na funkční a nefunkční. Pro jednoduchou orientaci mezi požadavky jsem každému přidělil jednoznačný identifikátor.

1.3.1 Funkční požadavky

Funkční požadavky slouží pro vymezení funkcionalit, kterými bude aplikace disponovat. Jedná se o seznam stručně popsaných hlavních funkcionalit aplikace. Ke každému funkčnímu požadavku je vhodné přiřadit prioritu a složitost, což pomůže k posouzení závažnosti nedostatků při ověřování správnosti funkcionalit. Prioritu a složitost budu hodnotit na škále od 1 do 5, kde 1 znamená nejnižší prioritu, respektive složitost a 5 nejvyšší prioritu, respektive složitost.

Dle odpovědí na dotazník by byl zájem o aplikaci evidující inzerce na vrhy štěňat z chovatelských stanic. Nicméně vyplynula také nedostatečná motivace a zájem chovatelských stanic zveřejňovat své nabídky. Tento problém jsem se rozhodl vyřešit navržením funkcionality tak, aby aplikace podporovala chov psů a zároveň měla pro chovatele i jiný přínos než pouze nabídku jejich štěňat.

Z dotazníku také vyplynulo, že drtivá většina chovatelů nevyužívá specializované aplikace pro evidenci zájemců o své štěňata. Na základě této skutečnosti jsem navrhl hlavní funkcionalitu aplikace tak, aby podporovala evidenci psů a vrhů chovatele a také podporovala správu rezervačních pořadníků pro jednotlivé vrhy. Od této funkcionality si slibuji, že motivuje chovatele k využívání aplikace a evidování jejich dat pomocí aplikace. Jakmile bude chovatel mít data uložená v aplikaci, bude jednoduché z těchto dat sestrojit nabídku a zveřejnit ji, aniž by uživatel musel vynaložit úsilí navíc. Od této funkcionality očekávám, že bude motivovat chovatele ke zveřejňování vrhů v aplikaci.

- **FR01 - Správa uživatelského účtu a jazyka aplikace**

  Většina funkcionalit aplikace bude zpřístupněna až po přihlášení do uživatelského účtu. Aplikace umožní registraci uživatele, editaci údajů uživatele a přihlášení a odhlášení. Aplikace bude dostupná ve více jazykových mutacích.

  - **Priorita:** 5
  - **Složitost:** 3

- **FR02 - Evidence a správa psů**

  Aplikace umožní evidovat a spravovat chované psy. Součástí evidence bude možnost přidávat záznamy s postupem času, které monitorují vývoj psa. Jedná se zejména o záznamy veterinárních záznamů a životní záznamy psa. Další funkcionalitou bude převedení vlastnictví psa na jiného uživatele.

  - **Priorita:** 5
  - **Složitost:** 4

- **FR03 - Evidence a správa vrhů**

  Součástí aplikace bude možnost evidovat vrhy a jejich životní cyklus od plánování vrhu až po rezervace štěňat. Ke každému vrhu bude možno definovat podmínky pro rezervaci. Po narození štěňat bude možnost k vrhu přiřadit jednotlivá štěňata a ty dále evidovat viz. FR2.

  - **Priorita:** 4
  - **Složitost:** 4

- **FR04 - Prezentace vrhu**

  Evidované vrhy bude možno zveřejnit tak, aby k nim měli přístup i ostatní návštěvníci webu a mohli případně projevit zájem o rezervaci.

  - **Priorita:** 3
  - **Složitost:** 3

- **FR05 - Projevení zájmu o psa**

  Při prohlížení veřejných nabídek bude možno projevit zájem o psa a požádat o jeho rezervaci. Pokud bude mít vrh definované požadavky na zájemce, bude nutnou součástí žádosti prokázání splnění těchto požadavků.

  - **Priorita:** 3
  - **Složitost:** 2

- **FR06 - Evidence a správa rezervací**

  Uživatel si bude ke svým nabídkám moci evidovat rezervace. U veřejných nabídek si budou moci zájemci vytvořit rezervaci sami, případně si je bude moci uživatel vytvořit svépomocí. Rezervacím poté může po narození přiřadit rezervované štěně.

  - **Priorita:** 3
  - **Složitost:** 4

- **FR07 - Informace o rezervovaném psu**

  Jakmile majitel nabídky přiřadí psa zájemci, zájemce uvidí evidované údaje o štěněti.

  - **Priorita:** 2
  - **Složitost:** 3

- **FR08 - Vyhledávání v nabídkách vrhů**

  Návštěvníci webu budou moci vyhledávat a procházet veřejné nabídky vrhů.

  - **Priorita:** 4
  - **Složitost:** 4

- **FR09 - Prohlížení a správa splněných podmínek registrace do klubu**

  Pro vybrané chovatelské kluby budou definovány podmínky registrace psa. Majitel psa si poté bude moci zvolit klub, do kterého chce psa registrovat a evidovat splnění jednotlivých podmínek.

  - **Priorita:** 1
  - **Složitost:** 2

1.3.2 Nefunkční požadavky

Nefunkční požadavky jsou převážně technické požadavky na zpracování a parametry aplikace. U jejich definice je třeba dbát na to, aby byly měřitelné a tedy i existoval způsob ověření jejich splnění.

- **NFR01 - Webová aplikace**

  Aplikace bude dostupná výhradně přes web.

- **NFR02 - HTTPS**

  Aplikace bude komunikovat výhradně za pomoci protokolu HTTPS.

- **NFR0 - Odpověď na požadavek**

  Aplikace na každý požadavek odpoví do 2 sekund.

- **NFR04 - Spustitelnost**

  Aplikace bude spustitelná minimálně v následujících prohlížečích ve verzích, které byly vyvinuty v roce 2020 a později.

  - Google chrome
  - Safari
  - Microsoft Edge
  - Mozilla Firefox

- **NFR05 - Zálohování**

  Databáze musí být zálohovaná tak, aby při výpadku nebyla ztracena data starší než 24 hodin.

- **NFR06 - Dostupnost**

  Aplikace musí být dostupná po 99,5 % času ročně.

- **NFR07 - Paralelní zpracování**

  Aplikace musí být schopna paralelně zpracovat minimálně 200 požadavků za sekundu.

- **NFR08 - Počet dotazů do databáze**

  Aplikace při obsloužení každého požadavku provede maximálně 50 dotazů do databáze.

1.3.3 Případy užití

Případy užití (zkráceně UC - z anglického „use case“) rozvíjí funkční požadavky a dále je specifikují a člení na jednotlivé stručné případy užití. Pomocí UC lze detailně definovat a popsat funkcionalitu aplikace, a také je lze využít k jednoduchému ověření, zda implementace pokrývá veškerou plánovanou funkcionalitu.

Každý UC je identifikován unikátním identifikátorem, popsán názvem a stručným popisem pro upřesnění. Ke každému UC jsou přiřazeni aktéři, kteří jej mohou vykonat. Současný návrh využívá 2 aktéry - přihlášeného uživatele a nepřihlášeného uživatele. Pokud vykonání UC způsobí změnu stavu aplikace, je ve výčtu popsána.

Celkem jsem identifikoval 31 uživatelských požadavků, které jsem následně rozdělil do 3 skupin dle jejich účelu. Některé uživatelské požadavky se nachází ve více skupinách, jelikož slouží pro více účelů. Níže naleznete seznam skupin a jejich případů užití. Detailní popis všech případů užití je k vidění v příloze

C. Diagram zobrazující všechny UC a vazby mezi nimi lze vidět na obrázku 1.8.

Pokrytí všech funkčních požadavků je znázorněno v tabulce 1.2. Pro 100% pokrytí je nutné, aby v každém řádku a každém sloupci byl alespoň jeden znak „✓“. Pokud by znak chyběl ve sloupci, daný funkční požadavek by nebyl pokryt žádným případem užití. Pokud by naopak znak chyběl v řádku, daný případ užití by nepokrýval žádný funkční požadavek a tím pádem by byl zbytečný.

1. **Správa uživatelského účtu**
    - UC01 - Registrace
    - UC02 - Přihlášení
    - UC03 - Odhlášení
    - UC04 - Změna uživatelských údajů
    - UC05 - Změna jazyka aplikace
2. **Podpora chovatelů psů**
    - UC06 - Vytvoření psa
    - UC07 - Editace psa
    - UC08 - Odstranění psa
    - UC09 - Zobrazení psa
    - UC10 - Převedení vlastnictví psa
    - UC11 - Vytvoření životního záznamu
    - UC12 - Zobrazení životních záznamů
    - UC13 - Odebrání životního záznamu
    - UC14 - Editace životního záznamu
    - UC15 - Vytvoření vrhu
    - UC16 - Editace vrhu
    - UC17 - Odstranění vrhu
    - UC18 - Zobrazení vrhu
    - UC20 - Potvrzení rezervace
    - UC21 - Definice podmínek rezervace
    - UC22 - Stornování rezervace
    - UC23 - Zobrazení rezervace
    - UC24 - Přiřazení psa k rezervaci
    - UC26 - Zobrazení chovatelských klubů
    - UC27 - Zobrazení podmínek registrace

- UC28 - Volba chovatelského klubu pro registraci
- UC29 - Označení podmínky registrace do chovatelského klubu za splněnou
- UC30 - Označení podmínky registrace do chovatelského klubu za nesplněnou
- UC31 - Zobrazení splněných a nesplněných podmínek pro registraci

3. **Podpora zájemců o psy**

- UC09 - Zobrazení psa
- UC18 - Zobrazení vrhu
- UC19 - Žádost o rezervaci
- UC22 - Stornování rezervace
- UC23 - Zobrazení rezervace
- UC25 - Vyhledání vrhu

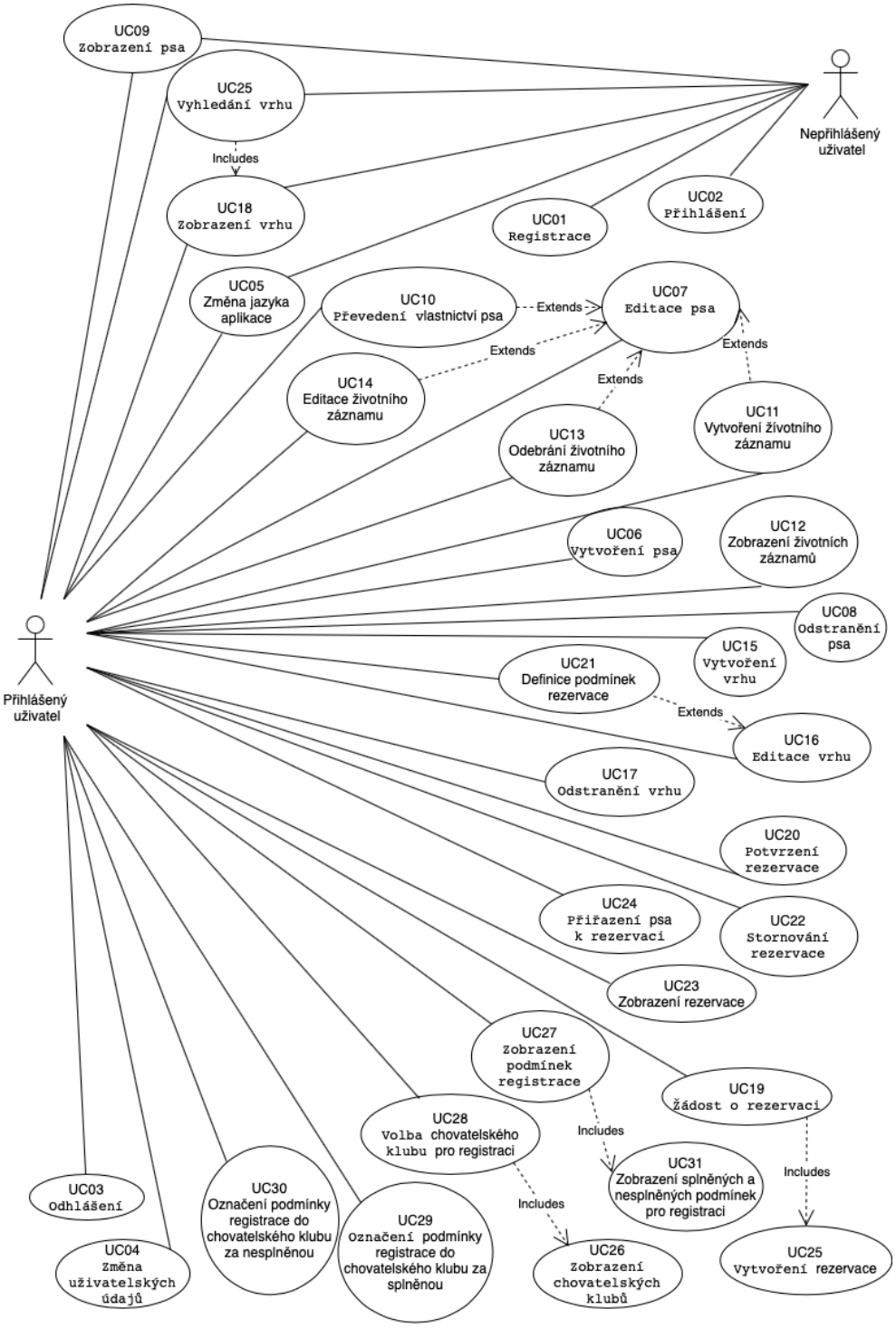
1.3.4 Diagram aktivit

Funkcionalitu aplikace lze popsat i z pohledu posloupnosti a návazností vykonávaných úkolů. Používají se zejména pro popis a upřesnění složitých akcí, které nemusí být vždy nutně jednoznačně pochopitelné. K popisu se využívá UML diagram aktivit.

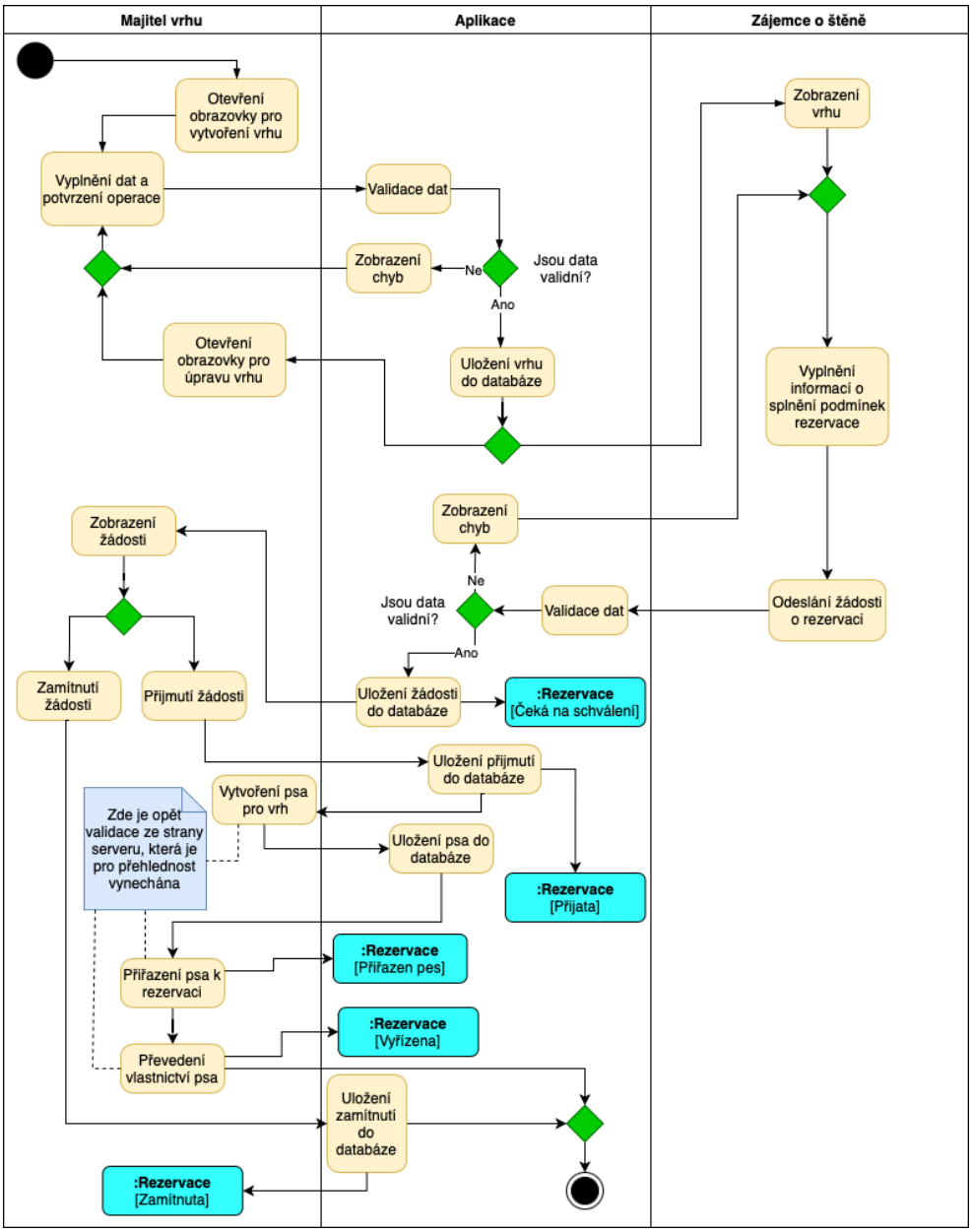
Mnou navržená aplikace, obsahuje jednu složitou funkcionalitu, a to je proces rezervace psa. V tomto procesu figurují tři aktéři: majitel vrhu, aplikace a zájemce o štěně. Proces začíná založením vrhu, zahrnuje úpravy vrhu a vytvoření žádosti o rezervaci a končí až se zamítnutou žádostí, nebo s potvrzenou žádostí a přiřazeným štěnětem k rezervaci. Diagram znázorňující tento proces je k vidění na obrázku 1.9.

Tabulka 1.2: Tabulka pokrytí funkčních požadavků případy užití

	FR01	FR02	FR03	FR04	FR05	FR06	FR07	FR08	FR09
UC01	✓	—	—	—	—	—	—	—	—
UC02	✓	—	—	—	—	—	—	—	—
UC03	✓	—	—	—	—	—	—	—	—
UC04	✓	—	—	—	—	—	—	—	—
UC05	✓	—	—	—	—	—	—	—	—
UC06	—	✓	—	—	—	—	—	—	—
UC07	—	✓	—	—	—	—	—	—	—
UC08	—	✓	—	—	—	—	—	—	—
UC09	—	✓	—	✓	—	—	✓	—	—
UC10	—	✓	—	—	—	—	—	—	—
UC11	—	✓	—	—	—	—	✓	—	—
UC12	—	✓	—	—	—	—	✓	—	—
UC13	—	✓	—	—	—	—	—	—	—
UC14	—	✓	—	—	—	—	—	—	—
UC15	—	—	✓	—	—	—	—	—	—
UC16	—	—	✓	—	—	—	—	—	—
UC17	—	—	✓	—	—	—	—	—	—
UC18	—	—	✓	✓	—	—	—	✓	—
UC19	—	—	—	—	✓	—	—	—	—
UC20	—	—	—	—	—	✓	—	—	—
UC21	—	—	✓	—	—	✓	—	—	—
UC22	—	—	—	—	—	✓	—	—	—
UC23	—	—	—	—	✓	✓	✓	—	—
UC24	—	—	—	—	—	✓	—	—	—
UC25	—	—	—	—	—	—	—	✓	—
UC26	—	—	—	—	—	—	—	—	✓
UC27	—	—	—	—	—	—	—	—	✓
UC28	—	—	—	—	—	—	—	—	✓
UC29	—	—	—	—	—	—	—	—	✓
UC30	—	—	—	—	—	—	—	—	✓
UC31	—	—	—	—	—	—	—	—	✓



Obrázek 1.8: Diagram případů užití



Obrázek 1.9: Diagram případů užití

Kapitola 2

Návrh

Důležitou součástí vývoje aplikace je její návrh. Návrh je pomyslným můstkem mezi analýzou a implementací. Zpracovává výstupy analýzy a na jejich základě popisuje aplikaci z technického hlediska. Návrh slouží jako „předpis“ pro implementaci a může také sloužit jako dokumentace k aplikaci. Úkolem návrhu je zpracovat analýzu a přetvořit ji v popis výsledné aplikace. Při návrhu je nutno rozmyslet a rozhodnout základní postupy a principy vývoje aplikace, jako jsou například použité technologie, architektura a rozdělení do jednotlivých komponent a jejich vzájemná komunikace. Důležitý je také návrh doménového modelu a uživatelského rozhraní, které výrazně usnadní následný vývoj aplikace.

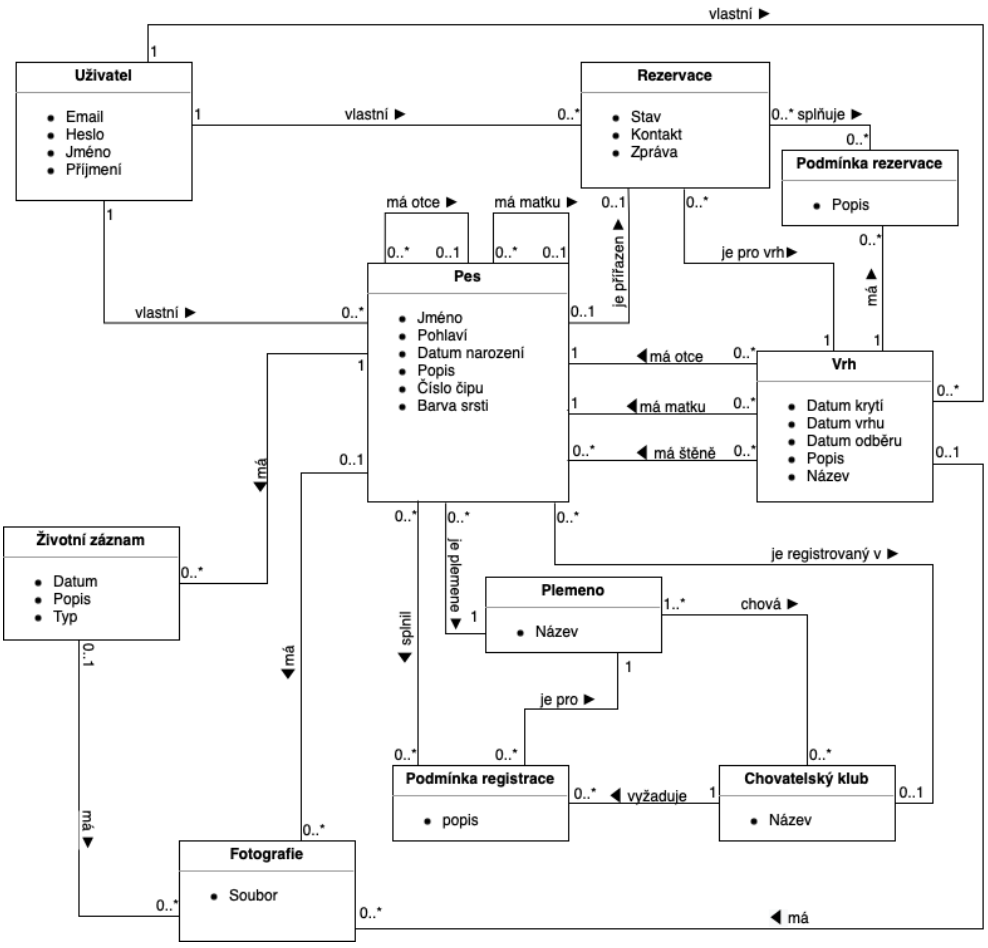
2.1 Doménový model

Doménový model slouží k definování domény aplikace, tedy jednotlivých entit, jejich atributů a vztahů mezi entitami. Je nezávislý na platformně, tzn. není závislý na konkrétním programovacím jazyce, architektuře aplikace nebo použitých technologiích. Zpravidla bývá zachycen formou UML diagramu. Diagram slouží k pochopení a popisu základních datových struktur v aplikaci a slouží jako podklad pro další návrhy či implementace. Z doménového návrhu lze jednoduše vytvořit databázový model a také je využíván k definici základních tříd pokud je implementace prováděna v objektově orientovaném programovacím paradigmatu.

Doménový model se obvykle vytváří na základě poskytnutých případů užití, případně se modeluje zároveň s jejich definicí. Pro automatizované vytvoření doménového modelu z případů užití existují nástroje a postupy, např. nástroj UCEd,pro který je postup popsán v článku „Generating a Domain Model from a Use Case Model“[10]. Často se také používají zkušenosti a analýza případů užití, kde je potřeba z případů užití extrahovat jednotlivé entity se základními atributy a vztahy mezi nimi. Při použití tohoto způsobu je potřeba zkušeností a značného přehledu v problematice, často je vhodné,

aby doménový diagram modelovala ta samá osoba, která definovala případy užití.

Protože celou aplikaci a analýzu jsem prováděl já (včetně definice případů užití) a v dané problematice se dle mého názoru obstojně orientuji, rozhodl jsem se pro modelování doménového diagramu na základě analýzy případů užití a vlastních zkušeností. Výsledný doménový model je zachycen formou UML diagramu na obrázku 2.1.



Obrázek 2.1: UML diagram doménového modelu

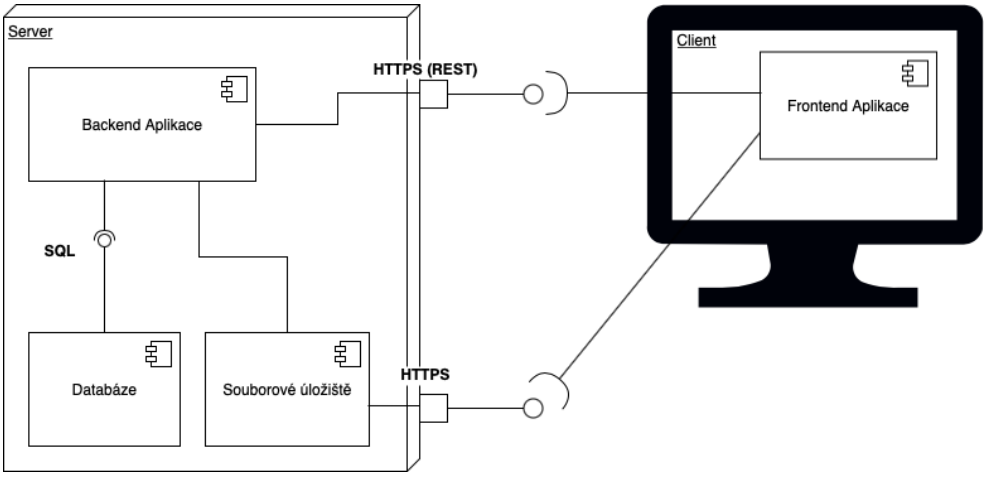
2.2 Struktura a Architektura

Aplikace bude tvořena dvěma hlavními komponentami - klientskou aplikací, která poběží v internetových prohlížečích, a serverovou aplikací, která bude spuštěna na serveru. Síťová architektura bude tedy klient-server. Hlavní část funkcionality bude obstarávat serverová aplikace, mezi její hlavní úkoly patří

persistence dat, autorizace, autentizace, a také bude vykonávat business logiku. K serverové aplikaci se poté budou připojovat klientské aplikace, které budou sloužit k zobrazení dat uživateli a budou umožňovat interakci uživatele s aplikací.

Architektura klient-server je velmi efektivní při případném rozšiřování aplikace na jiné platformy. Srdce aplikace (serverová část) zůstane neměnná a společná pro všechny platformy, stačí tedy pouze vyvinout nové klientské aplikace pro dané platformy. Do budoucna bude tedy možné rozšíření aplikace například do mobilních telefonů nebo do desktopových aplikací.

Klientská a serverová část spolu budou komunikovat za pomoci HTTPS protokolu. Serverová část bude poskytovat REST API, pomocí kterého bude klientská aplikace zpracovávat uživatelské interakce. Serverová aplikace rovněž umožní přístup k uloženým fotografiím z důvodu, aby při načítání obrázku nedocházelo ke zbytečnému spouštění skriptu serverové aplikace, a tedy zatěžování serveru.



Obrázek 2.2: Diagram architektury aplikace

2.2.1 Serverová aplikace

Pro dodržení Separation of concerns (dále jen SoC) je vhodné rozdělit aplikaci do více vrstev, které mezi sebou komunikují a každá obstarává přesně definovanou funkcionalitu. Pří návrhu architektury serverové aplikace jsem se inspiroval třívrstvou model-view-controller architekturou, jelikož však view v tomto případě obstarává klientská část aplikace, upravil jsem tuto architekturu na následující dvě vrstvy:

- **Controller**

  Controller bude obsluhovat API. Bude zachycovat serverové požadavky klientské části aplikace, ty autentizuje a autorizuje, na jejich základě předá modelu správné pokyny a data pro vykonání aplikační logiky. Následně serializuje výsledek operace modelu a odešle jej zpátky klientské části aplikace.

- **Model**

  Model bude vykonávat veškerou aplikační logiku a starat se o ukládání a načítání dat z databáze.

2.2.2 Klientská aplikace

Klientská aplikace slouží hlavně pro prezentaci dat a obsluhu uživatelské interakce. Pro tuhle činnost stačí následující dvouvrstvá architektura:

- **Store**

  Store neboli úložiště bude ukládat data sdílená všemi komponentami napříč klientskou aplikací.

- **View**

  View bude sloužit k načítání dat ze serverové části aplikace a jejich prezentaci. Pro jejich uchování bude využívat vrstvu store. Rovněž bude odchytávat uživatelskou interakci a odesílat ji na serverovou část aplikace.

2.3 Technologie

Při volbě technologií jsem se zaměřil převážně na jejich dostupnost, jednoduchost vývoje a nízkou náročnost na údržbu. Volil jsem je tedy tak, aby měla širokou uživatelskou základnu a měla dostatečnou dokumentaci. V potaz při volbě technologií jsem také vzal své osobní zkušenosti s danými technologiemi. V této kapitole použité technologie popíšu a vyzdvihnu jejich přednosti, pro které jsem je zvolil.

2.3.1 Technologie serverové části

Jako server, kde poběží serverová aplikace jsem zvolil Apache HTTP Server verze 2.4. Jedná se o velmi rozšířený open-source webový server, který je jednoduchý na správu a údržbu. Disponuje mnoha funkcionalitami, které jsou dostačující pro běh webových aplikací. Skutečnost, že je tento server žádaný pro VMS a další cloudové řešení hostingu webů a webových aplikací poukazuje na to, že se pro tyto účely často používá i v praxi.

Programovacím jazykem, ve kterém bude napsána serverová aplikace, jsem zvolil PHP verze 7.4. Jedná se o skriptovací programovací jazyk, který se využívá převážně pro tvorbu webových stránek a aplikací. PHP stále patři k nejčastěji používaným programovacím jazykům pro tvorbu webových stránek a má velkou uživatelskou základnu, takže při řešení problémů je velká pravděpodobnost, že stejný problém už někdo jiný řešil a postup řešení někde publikoval. PHP se stále rozvíjí a s každou verzí přichází vyšší rychlosti a snižuje se paměťová náročnost programů. Verze 7.4 již relativně dobře podporuje objektově orientované programování. Obrovskou výhodou je počet dostupných frameworků[1], které usnadňují vývoj.

Framework pro serverovou aplikaci jsem zvolil Symfony verze 5.2. Symfony je opět velmi rozšířený mezi komunitou vývojářů a disponuje vynikající dokumentací. Rovněž je stále udržovaný a vyvíjí se nové verze. Hlavní předností Symfony je její rozšiřitelnost a obměnitelnost. Symfony sestává se spojení mnoha komponent tak, aby spolu navzájem spolupracovaly a vývoj v nich byl co nejvíce efektivní. Největší výhodou Symfony je fakt, že nevyžaduje striktní využívání všech jejich nástrojů. Pokud jednotlivé komponenty nevyhovují pro účely projektu, lze je jednoduše nahradit jinými nebo si je naprogramovat vlastnoručně. To většinou není nutné, protože protože základní komponenty Symfony jsou flexibilní a lze s jejich pomocí vyvinout širokou škálu funkcionalit. Další nesmírnou výhodou je jednoduchá rozšiřitelnost a existence mnoha „Symfony bundles“, které lze do aplikace jednoduše přidat a poskytují dodatečné funkcionality. Doporučená architektura Symfony aplikací, která je k vidění na obrázku 2.3, je shodná se mnou navrženou architekturou.

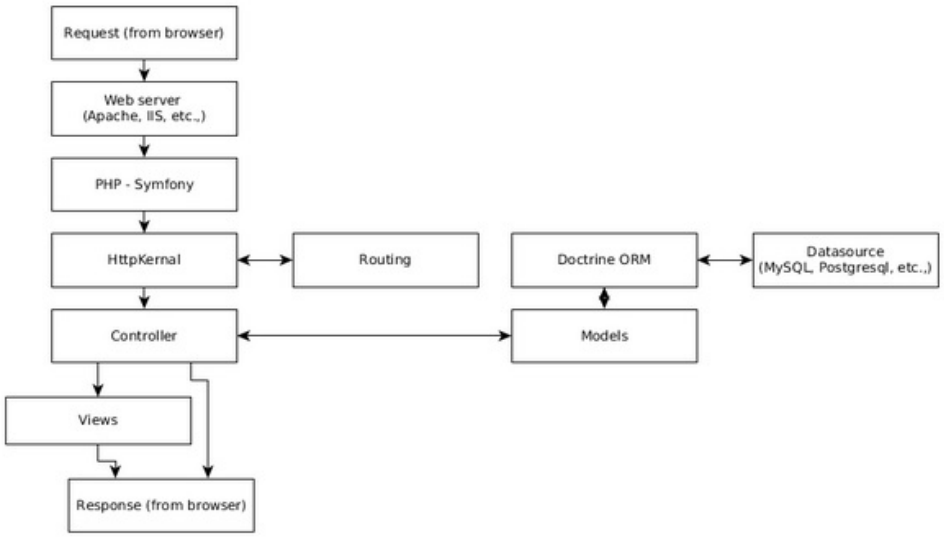
> „Symfony je sada PHP komponent, framework webové aplikace, filosofie a komunita - to vše pracující společně v harmonii“ [11]

Databázový server jsem zvolil MariaDB verze 10.5. Jedná se o populární open-source relační databázi, která vychází z MySQL. Narozdíl od MySQL je u MariaDB garantováno, že nebude komercializována a zůstane open-source. Velmi často je součástí linuxových cloudových distribucí.[13]

2.3.2 Technologie klientské části

Klientská část aplikace bude vyvinuta v programovacím jazyce Javascript v jeho normované verzi ECMAScript. Jedná se o scriptovací jazyk, který je narozdíl od většiny ostatních programovacích jazyků založen na prototypech. Javascript jsem zvolil, protože jej dokáže spustit většina internetových prohlížečů a v současné době jej používá naprostá většina webových stránek a aplikací pro

[1] Framework je připravená struktura aplikace a sada nástrojů pro řešení konkrétních problémů. Většinou se jedná o implementované funkcionality, které jsou snadno rozšiřitelné a modifikovatelné a zjednoduší implementaci, a které jsou obsaženy ve většině aplikací daného charakteru



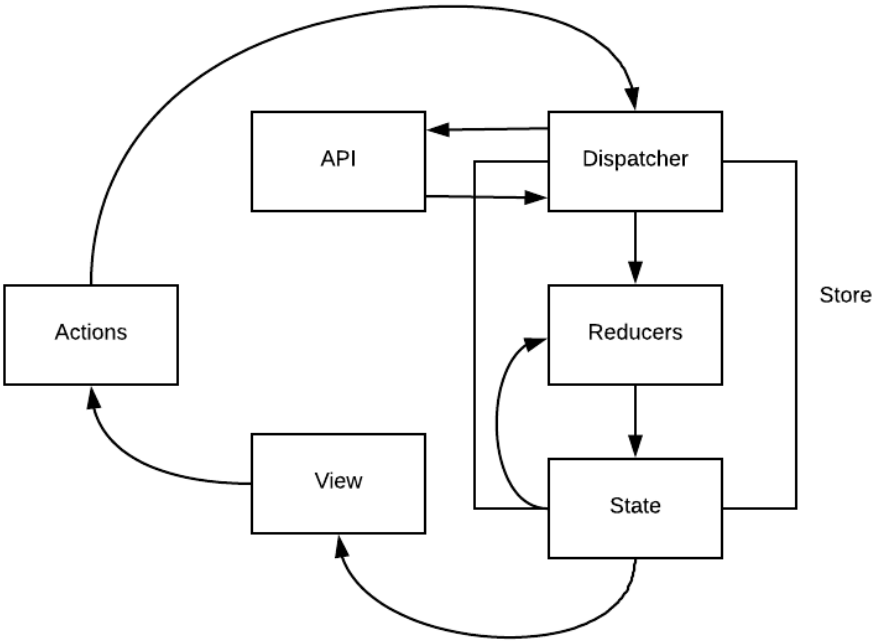
Obrázek 2.3: Architektura Symfony aplikace

[12]

dynamické zobrazení svého obsahu. Možnost dynamických změn obsahu není jedinou předností využití javascriptu, další nesmírnou předností je minimalizace dat, která se musí posílat po síti mezi klientskou a serverovou částí. To je největším přínosem pro uživatele, kteří budou web procházet na zařízeních s limitovaným objemem dat, která mohou posílat a přijímat po síti. Tato skutečnost je způsobena tím, že zdrojový kód aplikace a jejich komponent, který je oproti ostatním posílaným datům velký, je uložen ve scriptu, který si většina prohlížečů dokáže uložit do paměti cache. Počet přenosů tohoto skriptu po síti je tedy minimalizován a většina přenosů po síti sestává pouze z přenosu nezbytných dat a akcí. I přes velký počet výhod, je zde nevýhoda, že takto implementovaná aplikace nebude dostupná pro uživatele, kteří si nepřejí nebo mají zakázané, aby jejich internetový prohlížeč spouštěl javascriptové skripty. Funkcionalita aplikace bude tedy podmíněna povolením spouštění javascriptu v internetovém prohlížeči, což na jednu stranu může být omezující, ale na stranu druhou je to běžný postup vývoje webových aplikací, protože vyvíjet aplikaci ve dvou verzích - jedné, která je založena na javascriptu a druhé, která dokáže běžet i bez javascriptu, by bylo velmi náročné a podíl uživatelů, kteří si nepřejí spouštět javascript je zanedbatelný.

Jako framework pro implementaci klientské části aplikace jsem zvolil React Redux. Jedná se o kombinaci dvou frameworků založených na javascriptu, které umožňují dynamickou prezentaci HTML stránek a komunikaci se serverovým API. Jedná se o často používaný způsob vývoje klientských aplikací, který má opět početnou uživatelskou základnu s dokumentací a mnoha

návody dohledatelnými na internetu. Tyto frameworky umožňují rozdělení logiky aplikace do dvou vrstev přímo dle mého návrhu architektury. React obstarává funkcionalitu vrstvy View k prezentaci dat a obsluze uživatelských interakcí a Redux obstarává funkcionalitu vrstvy store, která uchovává data pro view. Komunikace mezi jednotlivými komponentami React Redux aplikace je znázorněna na obrázku/citefig:react-redux-components.



Obrázek 2.4: Diagram komunikace jednotlivých části aplikace React Redux [14]

2.4 API

Dle mého názoru je správný návrh API aplikace jedním z nejdůležitějších částí při návrhu aplikace. Usuzuji tak dle skutečnosti, že API definuje komunikaci mezi hlavními komponentami, takže návrh a implementace jednotlivých komponent je vhodné přizpůsobit návrhu API. Případná chyba v návrhu a její oprava by se tak propagovala do všech zainteresovaných komponent a znamenala by nutnost jejich úpravy. V současně navržené aplikaci by to tedy znamenalo úpravu serverové a klientské části aplikace, což by bylo velmi pracné. V budoucnu, kdy by na API záviselo více komponent, kde by některé mohly

být třetích stran, by bylo úprav potřeba mnohem více a komplikace by mohly být fatální.

Pro jednoduchou rozšiřitelnost API jsem se rozhodl využít REST architektonické vzory. Formát dat pro komunikaci s API jsem zvolil JSON.

2.4.1 REST

Representational state transfer (REST) je sada architektonických návrhových vzorů pro komunikaci mezi komponentami, které cílí na minimalizaci latence a komunikace po síti a maximalizují rozšiřitelnost a škálovatelnost aplikace. REST umožňuje cachování a znovupoužitelnost interakcí, dynamickou zastupitelnost komponent a zpracování akcí prostředníky, což splňuje požadavky na internetově škálovatelný hypermediální systém.[15]

Na rozdíl od ostatních přístupů ke komunikaci mezi komponentami je REST datově orientovaný, což znamená, že komponenty si mezi sebou posílají data a informaci o jejich reprezentaci. Základem REST je zdroj. Zdrojem může být jakákoliv informace. Každý zdroj je identifikován pomocí URI. Akce na zdrojích jsou vykonávány pomocí reprezentace dat, která zachycují současný, nebo chtěný stav komponent. Reprezentace je definována jako sada bitů a jejich metadata která je popisují. Komunikace s REST rozhraním probíhá pomocí HTTP, kde data jsou v těle zprávy a metadata, která je popisují, se posílají pomocí HTTP hlaviček. Součástí dat jsou také ovládací data, která definují účel zaslání dat. Těmto ovládacím datům se říká metody. 4 základní metody pro operaci s daty pomocí REST jsou:

- **GET**

  Tato metoda slouží pro získání dat ze zdroje.

- **POST**

  Tato meta slouží pro vytvoření zdroje, pokud daný zdroj už existuje, skončí s chybou.

- **PUT**

  Tato meta slouží pro vytvoření nebo úpravu zdroje. Pokud zdroj neexistuje, vytvoří jej, pokud existuje, upraví jeho data,. Každopádně je výsledkem operace zdroj obsahující zaslané data.

- **DELETE**

  Tato meta slouží k odstranění zdroje.

Požadavky také mohou být upřesněny pomocí parametrů, které se nepředávají v obsahu požadavku. V případě metody GET, se tyto parametry posílají v URL, v ostatních metodách se posílají jako součást HTTP požadavku. Pomocí těchto parametrů se například definuje rozsah odpovědi, např. počet položek.

Ke kategorizaci výsledku akce na zdroji se v odpovědi používá stavový HTTP kód. Jedná se o sadu definovaných tříciferných dekadických kódů. Druhá a třetí cifra upřesní výsledek akce, nejdůležitější je první cifra, na jejímž základě se výsledky dělí do pěti kategorií výsledků:

- **1xx - Informační**

  Tato skupina slouží pro dodatečné informace o stavu akce.

- **2xx - Úspěch**

  Tato skupina znamená úspěch operace., což je žádaný a očekávaný výsledek většiny akcí.

- **3xx - Přesměrování**

  Tato skupina informuje, že na daném URI se daný zdroj již nenachází. Návratové kódy z této skupiny dávají informaci, jak a kde chtěný zdroj najít a jak k němu přistoupit.

- **4xx - Chyba požadavku**

  Tato skupina oznamuje neúspěch ve vykonání akce z důvodu špatného požadavku.

- **5xx - Chyba serveru**

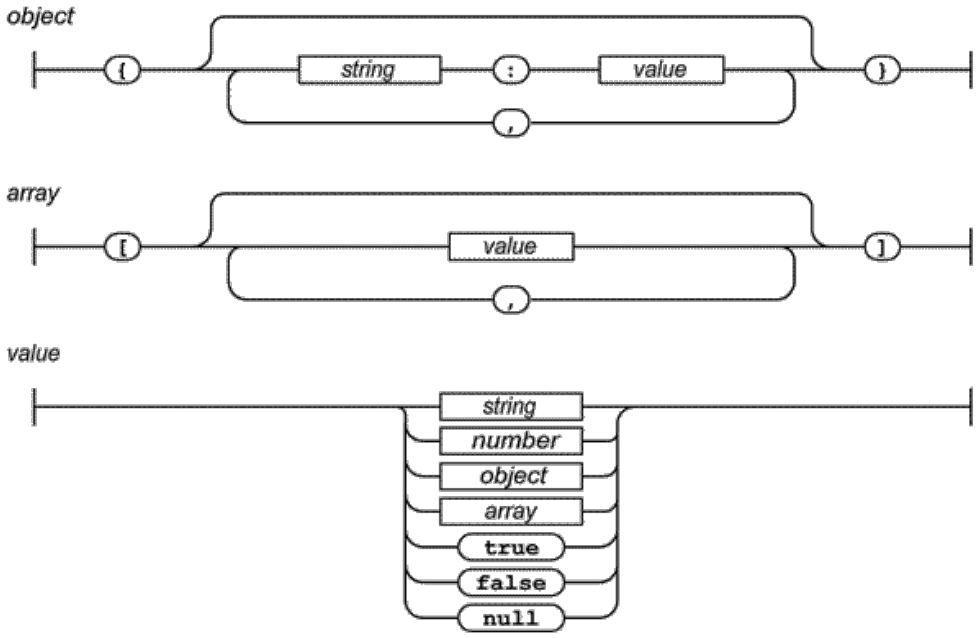
  Tato skupina informuje o neúspěchu ve vykonání akce, ale z důvodu chyby v API, tzn. že požadavek byl formulován správně, ale jeho obsluha selhala na straně serveru.

2.4.2 JSON

> „JSON je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Je založen na podmnožině Programovacího jazyka JavaScript, Standard ECMA-262 3rd Edition - December 1999. JSON je textový, na jazyce zcela nezávislý formát, využívající však konvence dobře známé programátorům jazyků rodiny C (C, C++, C#, Java, JavaScript, Perl, Python a dalších). Díky tomu je JSON pro výměnu dat opravdu ideálním jazykem.“ [16]

JSON jsem zvolil díky jeho nezávislosti a vysoké podpoře napříč programovacími jazyky. Jeho další výhodou je způsob zápisu dat, což minimalizuje přenos dat po síti. JSON se skládá z hodnot, které mohou nabývat dvou složených hodnot (objekt, seznam) a 4 primitivních hodnot (řetězec, číslo, boolean, null). Z těchto hodnot poté lze sestrojit veškeré JSON konstrukce. Pravidla pro skládání JSON formátu jsou znázorněna na obrázku2.5. Konkrétní příklad JSON formátu je k vidění v sekci implementace. Mezi nevýhody patří

nutnost definice struktury posílaných dat tak, aby si tato data mohly jednotlivé komponenty převést na své interní reprezentace.



Obrázek 2.5: Grafické znázornění JSON formátu

[17]

2.4.3 Návrh API

Popis REST API sestává převážně z identifikace a popisu jednotlivých zdrojů a operací, které s nimi lze provádět. Každý zdroj je identifikován pomocí URI v tomto případě URL, který jednoznačně identifikuje zdroj pomocí jeho adresy na síti. Je nutno dodržet, aby každý URL identifikoval maximálně jeden zdroj. Jelikož všechny zdroje API budou na stejné doméně a přístupné pouze pod protokolem HTTPS, URL pro zdroje budu vypisovat pouze jejich část nazývanou HTTP path, která se bude pro jednotlivé zdroje lišit.

Pro stručnost vynechám popis dat požadavků a odpovědí jednotlivých zdrojů, protože návratové HTTP kódy jsou použity podle jejich definic a data jen kopírují jednotlivé entity doménového modelu. Uvedu tedy jen vybrané zdroje a popis jejich funkcionality. Při definici jednotlivých zdrojů API jsem vycházel z případů užití a doménového modelu, kde zdroje jsou dle entit doménového modelu a akce podporují proveditelnost všech případů užití. Následující výpis byl v průběhu implementace modifikován.

Následující skupina zdrojů slouží pro obsluhu uživatelských účtů aplikace. Z bezpečnostních důvodů funkcionality pro resetování hesla a registraci uživatele vyžadují dvoufázové ověření.

- **/api/login**

  POST Přihlášení uživatele do aplikace. Přihlášení uživatele bude probíhat pomocí PHP sessions, které jsou implementovány pomocí HTTP cookies.

- **/api/logout**

  DELETE Odhlášení uživatele z aplikace a nastavení příslušných cookies.

- **/api/users/current**

  GET Získání dat aktuálně přihlášeného uživatele.

  PUT Úprava uživatelských dat aktuálně přihlášeného uživatele,.

- **/api/users/current/password**

  PUT Změna uživatelského hesla aktuálně přihlášeného uživatele,.

- **/api/users/register**

  POST Registrace uživatele. Aplikace používá dvoufázové ověření, kde na tomhle zdroji uživatel založí registraci a aplikace mu na emailovou adresu pošle druhý zdroj, kde registraci dokončí.

- **/api/users/register/verification**

  POST Druhá fáze registrace uživatele, kde dojde k dokončení registrace a poté se bude uživatel moci přihlásit.

- **/api/users/password-reset**

  POST Vytvoření žádosti o reset hesla. Aplikace používá doufázový reset hesla, kdy v první fázi se vytvoří žádost, systém odešle informační email s novým zdrojem, na kterém se musí akce dokončit.

- **/api/users/password-reset/hash-verification**

  POST Ověření platnosti hashe pro reset hesla.

- **/api/users/password-reset/finish**

  POST Dokončení dvoufázového resetu hesla.

Následující skupiny slouží převážně pro CRUD operace nad uchovávanými daty. Některé tyto zdroje vracejí na stejný požadavek rozdílné odpovědi z důvodů autorizace uživatele, tudíž odpovědi jsou závislé na konkrétním přihlášeném uživateli. Z důvodu úspory jsem některé zdroje vynechal, a jejich funkcionalitu obsluhují jiné zdroje, které jsou s nimi úzce spjaty.

- **/api/my-dogs**

  GET Slouží k získání evidovaných psů uživatele.Odpovědi se budou lišit dle aktuálně přihlášeného uživatele.

  POST Vytvoření nového psa pro aktuálně přihlášeného uživatele.

- **/api/my-dogs/{id}**

  DELETE Smazání psa.

  PUT Úprava evidovaných dat daného psa.

- **/api/my-dogs/{id}/life-records**

  GET Slouží k získání životních záznamů daného psa.

  POST Vytvoření nového životního záznamu pro daného psa.

- **/api/my-dogs/{id}/life-records/{id}**

  DELETE Smazání daného životního záznamu.

  PUT Úprava daného životního záznamu.

- **/api/my-dogs/{id}/registration-conditions/{conditionId}**

  PUT Potvrzení / zrušení splnění určité podmínky rezervace pro určitého psa.

- **/api/my-litters**

  GET Slouží k získání evidovaných vrhů aktuálně přihlášeného uživatele. Odpovědí se budou lišit na základě přihlášeného uživatele.

  POST Vytvoření nového vrhu.

- **/api/litters**

  GET Slouží k všech dostupných vrhů. Může být modifikován pomocí parametrů pro potřeby filtrování.

  POST Vytvoření nového vrhu.

- **/api/my-litters/{id}**

  DELETE Smazání určitého vrhu.

PUT Úprava určitého vrhu.

- **/api/my-litters/{id}/bookings**

  GET Získání rezervací určitého vrhu.

  POST Vytvoření nové rezervace.

- **/api/litters/{id}/bookings/{id}**

  DELETE Smazání určité rezervace.

  PUT Úprava určité rezervace. Zde lze provést zasláním konkrétních dat akce pro úpravu údajů, přijmutí rezervace, přiřazení psa i stornování rezervace.

- **/api/breeds**

  GET Získání všech evidovaných plemen.

- **/api/breeding-clubs**

  GET Získání všech evidovaných chovatelských klubů. Pro úsporu zdrojů tak výpis chovatelských klubů obsahuje chovaná plemena v klubu i podmínky pro uchovnění. Výsledek lze modifikovat parametry zejména pro získání chovatelských klubů určitých plemen.

2.5 Uživatelské rozhraní

Uživatelské rozhraní je rovněž velmi důležitou částí aplikace. Zastává roli prostředníka mezi aplikací a uživateli. Interaguje s uživatelem, sbírá od něj data a požadavky a ty překládá do formátu akceptovaného aplikací. A naopak získává data z aplikace a překládá je do formy, která je zase čitelná pro uživatele. Mimo to, že uživatelské rozhraní musí být 100% funkční, je velmi důležité aby jej uživatelé uměli používat. Je tedy kladen důraz na jednoduchost a intuitivnost ovládání uživatelského rozhraní všemi potencionálními uživateli.

2.5.1 Použitelnost

Pří návrhu uživatelského rozhraní je důležitá jeho použitelnost. Jacob Nielsen definoval použitelnost jako jeden z atributů pro hodnocení přijatelnosti systému. Jsou znázorněny na obrázku2.6. Samotnou použitelnost popsal jako multidimenzionální vlastnost uživatelského rozhraní, která se zpravidla dělí na následující pětici částí:[18]

- **Pochopitelnost**

  Uživatelské rozhraní by mělo být jednoduché a intuitivní pro používání pro všechny uživatele. Každý uživatel, začátečník i pokročilý, by měl být schopen co nejrychleji pochopit funkcionalitu a ihned začít rozhraní aktivně používat.

- **Efektivita**

  Uživatelské rozhraní musí zvyšovat efektivitu práce. Jakmile se uživatel naučí aplikaci používat, jeho produktivita by měla rapidně vzrůst. Expertním uživatelům aplikace by mělo používání pokročilých funkcionalit ušetřit více času, než kolik je stálo se je naučit.

- **Zapamatovatelnost**

  Postup jednotlivých akcí by měl být jednoduše zapamatovatelný tak, aby uživatel který provedl jednu akci, byl schopen tu samu akci zopakovat později bez nutnosti se akci znovu učit. Tato vlastnost je důležitá převážně pro příležitostné uživatele aplikace.

- **Chyby**

  Systém by měl dovolit uživatelům dělat co nejméně chyb, aby uživatelé nemuseli řešit jejich nápravu. Pokud se uživatel chyby dopustí, měl by mu systém srozumitelně chybu popsat a vysvětlit tak, aby byl uživatel schopen ji co nejrychleji opravit.

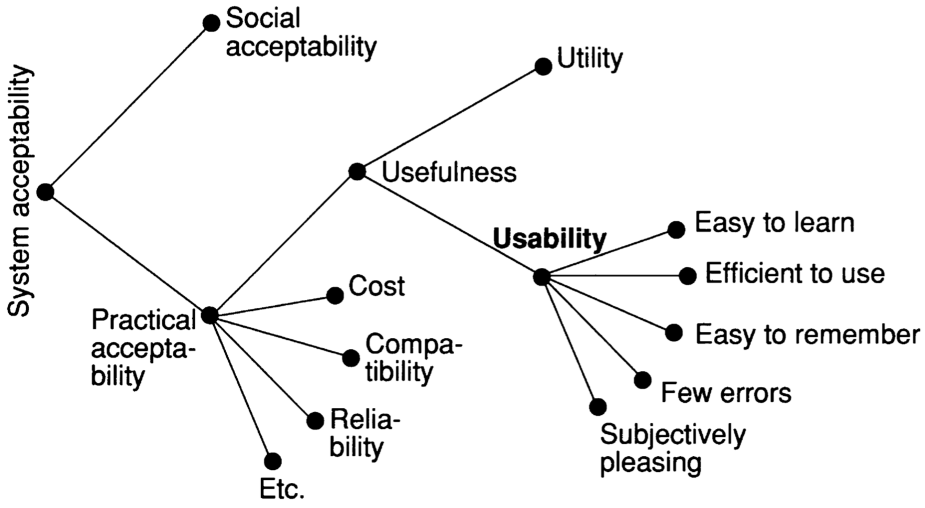
- **Spokojenost**

  Systém by měl být pro uživatele příjemný na používání tak, aby si jeho používání oblíbili. Je potřeba brát v potaz fakt, že spokojenost se systémem je subjektivní a může se u jednotlivých uživatelů výrazně lišit.

2.5.2 Úkoly aplikace

Při návrhu uživatelského rozhraní je jako první potřeba komplexně analyzovat a popsat očekávanou funkcionalitu aplikace. Před začátkem návrhu je nutné si definovat, k čemu se bude aplikace používat, kdo budou její uživatelé a jakou funkcionalitu budou od ní očekávat. Poté je tyto poznatky potřeba rozvinout a popsat v případech užití. Tyto kroky jsem učinil již v kapitole 1. Jakmile máme definovanou funkcionalitu aplikace, můžeme z jejího popisu definovat seznam úkolů aplikace.

Jednotlivé úkoly aplikace jsou atomické činnosti, které uživatelé budou v aplikaci provádět. Úkoly aplikace se popisují z uživatelského úhlu pohledu. Zpravidla se dělí na dva typy - zobrazit informaci a provést určitou akci. Jakmile jsou úkoly definovány, je vhodné je rozdělit do skupin tak, aby úkoly každé skupiny spolu úzce souvisely.



Obrázek 2.6: Graf vlastností přijatelnosti systému

[18]

2.5.2.1 Seznam úkolů aplikace

Seznam všech úkolů aplikace rozdělený do skupin je k vidění v příloze D. Každý úkol má svůj unikátní identifikátor a je k němu přiřazena informace o jeho typu, důležitosti, frekvenci používání a násobku hodnoty důležitosti a frekvence používání. Typ nabývá dvou hodnot: akce a zobrazení. Důležitost znamená dopad absence[2] úkolu na použitelnost celé aplikace a je hodnocena na škále 1-5. Frekvence používání je hodnocena na škále 1-3 a prezentuje hodnotu, jak často bude uživatel daný úkol vykonávat. Podrobnější význam hodnot je popsán v následujících výčtech.

Hodnoty důležitosti:

1: Zanedbatelná - V případě absence tohoto úkolu bude dopad na použitelnost aplikace minimální.

2: Nízká - Nemožnost vykonání tohoto úkolu sníží uživatelský prožitek z používání aplikace, ale nebude mít zásadní vliv na její použitelnost.

3: Střední - Špatný návrh tohoto úkolu nebude mít vážné dopady na funkcionalitu aplikace, ale její používání bude pro uživatele nepříjemné.

[2]Absence v tomto případě neznamená pouze úplné vynechání úkolu, ale i nemožnost uživatele vykonat tento úkol. To může nastat například v případě neintuitivního používání úkolu, pro uživatele nečekaného umístění úkolu v uživatelském rozhraní, ale i příliš obtížné vykonání úkolu.

4: Vysoká - Při nemožnosti vykonání těchto úkolů bude aplikace sice funkční, ale nebude mít žádný přínos pro efektivitu práce.

5: Kritická - Pokud uživatelé nebudou schopni vykonat tento úkol, aplikace bude nepoužitelná.

Hodnoty frekvence používání:

1: Výjimečná - Tato hodnota reprezentuje frekvenci akcí, kterou uživatel provádí velmi zřídka, většinou se jedná o úkoly, které nejsou spojené s hlavní funkcionalitou aplikace nebo o úkoly, které je třeba v běžném životním cyklu funkcionalit aplikace vykonat jen jednou.

2: Běžná - Úkoly, které se nevykonávají denně, ale stále jsou vykonávány několikrát v životním cyklu jednotlivých funkcionalit.

3: Vysoká - Úkoly, které se provádí téměř při každém využití hlavních funkcionalit aplikace.

2.5.2.2 Rozdělení úkolů aplikace do skupin

Úkoly jsem rozdělil do následujících kategorií:

- **Uživatelské účty**

  Skupina uživatelské účty obsahuje veškeré úkoly týkající se uživatelských účtů, od registrace přes přihlášení, až po odhlášení. Většinu těchto úkolů jsem ohodnotil důležitostí stupně 5, protože většina funkcionalit aplikace se váže ke konkrétním uživatelům a bez těchto úkolů by byly nedostupné.

- **Lokalizace**

  Sem patří veškeré úkoly spojené s jazykovou lokalizací aplikace. Úkoly jsou ohodnotil důležitostí stupně 3, protože bez funkčních překladů aplikace pořád umožňuje vykonávání svých funkcionalit, avšak bez jazykových překladů může být její používání pro uživatele značně nepohodlné, v krajním případě pro některé i nemožné.

- **Základní úkoly aplikace**

  Tato skupina obsahuje globální funkcionality aplikace, které budou součástí některých ostatních akcí. Jedná se o úkoly, které nesouvisí s navrženou funkcionalitou aplikace, ale pomáhají zvyšovat její přehlednost a tím pádem i použitelnost. Typickými zástupci jsou informace o provedené operaci, které zlepšují přehled o stavu systému, a dvoufázové ověření kritických operací pro prevenci uživatelských chyb.

- **Psi**

  Sem patří veškeré úkoly související s evidencí uživatelových psů.

- **Vrhy**

  Skupina vrhy seskupuje dvě velmi podobné a prolínající se skupiny úkolů. Početnější skupina zahrnuje správu evidence uživatelových vrhů, který dané vrhy vlastní. Druhá zase zastupuje úkoly související s procházením, vyhledáváním a zobrazením vrhů, které uživateli nepatří, pro účel případné rezervace štěněte.

- **Rezervace**

  Tato skupina opět obsahuje dva různé pohledy na úkoly. Jeden je správa rezervací pro uživatelovy vrhy. Druhý naopak slouží k rezervování štěňat u vrhu cizího majitele.

- **Chovatelské kluby**

  Sem patří veškeré úkoly spojené s registrací vlastněných psů v chovatelském klubu.

2.5.3 Lo-fi prototyp

Prototypy aplikací se vytváří z důvodů ušetření nákladů na vývoj. Jejich vývoj a modifikace jsou několikanásobně rychlejší a méně nákladné, než tomu je u finální aplikace. Prototypy lze podrobit testování, což umožňuje rozmyslet návrh uživatelského rozhraní a jednoduše odstranit jeho nedostatky, ještě než začne implementace samotné aplikace. Prototyp poté může sloužit jako předloha nebo šablona při implementaci výsledné aplikace.

Lo- fi prototyp, nebo také wireframe, je prvotní návrh uživatelského rozhraní, aplikace, který většinou neobsahuje aplikační logiku ani designové prvky. Jeho hlavním účelem je vytyčit základní rozložení a prioritu komponent, funkcionality a chování uživatelského rozhraní. Wireframe je zpravidla reprezentován grafickými nákresy obrazovek, ze kterých lze vyčíst jednotlivé funkcionality a přechody mezi jednotlivými obrazovkami. Nemusí být nutně v elektronické podobě, často se také kreslí na papír, díky čemuž se mu také přezdívá „papírový model“.

2.5.3.1 Návrh lo-fi prototypu

Díky podrobnému popsání úkolů aplikace jsem na jeho základě vytvořil lo-fi prototyp aplikace. U návrhu jednotlivých wireframe jsem bral v potaz veškeré definované a mně známé vlastnosti a vztahy úkolů. Rovněž jsem zohlednil své znalosti o zvyklostech uživatelských rozhraní webových aplikací a znalosti problematiky chovu psů. Do hlavního menu aplikace jsem se rozhodl umístit akce, které mají důležitost stupně 5 a odkazy na vydedukované reprezentativní zobrazovací akce s nejvyšší hodnotou D x F. Dále jsem zohledňoval převážně hodnotu D x F, kde úkoly s vyšší hodnotou jsem prioritizoval a umisťoval na více viditelná místa a blíže k hlavnímu rozvržení rozhraní. Úkoly jednotlivých

skupin jsem se snažil seskupit na stejné stránky. Rovněž jsem bral ohled na množství informací na jednotlivých stránkách tak, aby některé stránky neobsahovaly příliš mnoho informací, což by snižovalo jejich přehlednost.

V aplikaci se v podstatě protínají dva přístupy uživatelů. Jedním jsou majitelé psů a vrhů, kteří je evidují a nabízí. Tím druhým jsou pak zájemci o pořízení štěňat, kteří je vyhledávají, prohlíží a poptávají. Oba tyto přístupy mají mnoho stránek velmi podobných, lišících se pouze v možnosti vykonání určitých akcí, plynoucích z jejich pravomocí. Z tohoto důvodu jsem se rozhodl, až na pár výjimek, jednotlivé stránky navrhovat jen jednou, kde jsou zaznamenány všechny možné komponenty, které může aplikace obsahovat. Ve finální implementaci aplikace se poté budou vykreslovat jen ty komponenty, ke kterým bude mít daný uživatel přístup.

Jelikož výsledným typem aplikace bude mobile first, pro zjednodušení jsem se omezil pouze na návrh wireframů pro obrazovky mobilních telefonů. Tohle omezení nepředstavuje velký problém, protože existuje mnoho předpřipravených nástrojů a šablon, které obsah psaný pro displeje mobilních telefonů dokáží „rozumně“ zobrazit na i jiných rozměrech obrazovek.

Návrhy obrazovek jsou v angličtině, protože je zvolena jako hlavní jazyk aplikace. Finální wireframes jsou k vidění v příloze E.

2.5.3.2 Heuristická analýza lo-fi prototypu

Heuristická analýza v podstatě spočívá v prohlížení uživatelského rozhraní a přemýšlení o jeho silných stránkách a nedostatcích. Jedná se o systematickou kontrolu uživatelského rozhraní. Cílem je najít nedostatky v použitelnosti tak, aby mohly být opraveny v iterativním návrhovém procesu. Sestává v posuzování souladu rozhraní s definovanou sadou principů použitelnosti (heuristik). [18]

Analýzu jsem prováděl průběžně při návrhu prototypu a zjištěné nedostatky jsem ihned opravoval. Jako hodnocené principy použitelnosti jsem si zvolil desatero z Nielsenovi knihy Usability engineering [18], které popisuji níže včetně odhalených nedostatků a opatření které jsem vykonal aby rozhraní heuristiky splňovalo.

Jednoduchost a přirozenost Rozhraní by mělo být co nejjednoduší a mělo by obsahovat pouze nezbytně nutné komponenty. Každá komponenta na stránce zaměstnává uživatele, který je nucen jí věnovat pozornost a pochopit její význam a použití. S rostoucím počtem informací a funkcionalit na stránku roste pravděpodobnost nepochopení nebo přehlédnutí některých akcí.

Při návrhu jsem analýzou odhalil problém nadměrného množství informací na některých stránkách. Pro odstranění jsem je rozdělil na více stránek, kde každá vykonává určitou podmnožinu funckionalit z původní velké stránky.

Používej jazyk uživatele Každá skupina lidí s konkrétním zaměřením používá vlastní názvosloví a výrazy, které nutně nemusí být spisovné nebo gramaticky bezchybné. Tito uživatelé jsou ale na tyto výrazy navyklí a častokrát ani netuší korektní formu těchto výrazů. Je proto vhodné aplikaci přizpůsobit uživatelům a věci pojmenovávat jejich terminologií, což minimalizuje míru nepochopení jednotlivých funkcionalit.

Jelikož se v daném oboru pohybuji, jsem seznámen s používanými výrazy a ty jsem rovnou použil i v návrhu.

Minimalizuj zatížení paměti uživatele Pro lidskou paměť je mnohem jednodušší si vzpomenout, pokud člověk danou věc právě vidí. Zobrazování informací a nápověd na vhodných místech pomocí této vlastnosti zvyšuje použitelnost rozhraní.

Uživatelské rozhraní obsahuje na každé stránce hlavní menu, které uživatele může nasměrovat na jednotlivé funkcionality aplikace. Dále jsou při editaci dat předvyplňena stará data a notifikace po provedení jednotlivých akcí obsahují nápovědu, jakou další akci může uživatel v návaznosti vykonat.

Konzistence Pokud více komponent poskytuje stejné akce, je pro uživatele mnohem jednoduší je vykonávat právě, když jsou v každé komponentě stejně reprezentovány a totožně se i chovají.

Rozložení jednotlivých stránek je stejné. Jako první komponenta je menu, následuje drobečková navigace a nadpis stránky. Až poté následují konkrétní komponenty stránky. Umístění tlačítek je u všech komponent stejné - smazání komponenty ve výpisu je vždy v pravém horním rohu, editace v levém dolním, v detailu je tlačítko pro smazání až na úplném konci stránky, atd.

Zpětná vazba Systém by měl uživateli průběžně poskytovat zpětnou vazbu o jeho aktuálním stavu. Typicky se jedná nejen o chybové hlášky, ale také o oznámení úspěšného provedení akce či o průběžné informování o průběhu aktuálně prováděné akce.

V prototypu jsem navrhl komponentu notifikace, která se bude zobrazovat po každém vykonání akce a bude obsahovat výsledek. Rovněž formulářové prvky budou při zadání nesprávné hodnoty obsahovat informaci o chybě.

Jasně vyznačené odvolání akce Žádný uživatel nemá rád, pokud nemá plnou kontrolu nad systémem. Rovněž každý dělá chyby a někdy uživatel provede nezamýšlenou akci. Je tedy vhodné, aby v každém případě měl možnost tuto akci přerušit a vrátit se do předchozího stavu.

Tato heuristika je vyřešena platformou aplikace. Není mi znám žádný webový prohlížeč, který neumožňuje krok zpět. Navíc je na každé stránce přítomno menu a drobečková navigace, které umožňují uživateli snadno přejít na jinou stránku.

Zkratky Jakmile uživatelé začnou být zkušenými v používání aplikace, budou mít potřebu často opakované akce vykonávat co nejrychleji. Z tohoto důvodu je vhodné pro dané operace vymezit v aplikaci zkratky pro dané operace, aby uživatelům urychlily práci.

Tato heuristika je opět částečně řešena platformou aplikace. Webové prohlížeče často disponují spoustou zkratek a funkcionalit pro zjednodušení prohlížení webu. V aktuálním návrhu aplikace jsem nedetekoval funkcionalitu, pro niž by bylo vhodné vyvinout zkratku. Nicméně je možné, že je tato domněnka mylná a až se spuštěním aplikace se ukáží funkcionality, které uživatele zpomalují.

Smysluplné chybové hlášky Nikdo není dokonalý a každý dělá chyby. Často uživatelé ani nevědí, v čem udělali chybu. Pro maximalizaci použitelnosti je proto vhodné psát srozumitelné chybové hlášky, které obsahují popis chyby a v ideálním případě nápovědu, jak chybu odstranit.

Všechny chybové hlášky jsou navržené podle zmíněných principů.

Prevence chyb uživatele Dělání chyb není pro uživatele příjemné. Uživatelský prožitek značně vylepšuje, pokud systém minimalizuje možné chyby.

Tuto heuristiku řeší například návrh formulářů, kde jsou jednotlivé prvky voleny tak, aby bylo možné zadávat pouze validní hodnoty. Například pro volbu rodičů psa se pro matku a otce nabízí pouze psi daného pohlaví. Dále tuto problematiku řeší návrh obsahu stránek na principu HATEOAS kdy uživatelské rozhraní zobrazuje uživateli pouze ty akce, které může daný uživatel vykonat.

Nápověda a dokumentace I když se zdá, že uživatelské rozhraní je intuitivní a snadno použitelné, stále se mohou najít uživatelé, kteří jej nepochopí.

Z tohoto důvodu jsem navrhl homepage rozcestník s popisem jednotlivých hlavních funkcionalit.

Kapitola 3

Implementace

V této kapitole popíšu postup a použité principy při implementaci. Výsledek implementace (zdrojový kód, a počáteční testovací data) je obsažen v přiložené SD kartě.

3.1 Serverová část

Serverová část aplikace byla dle návrhu implementována v PHP verze 7.4 ve frameworku Symfony verze 5.2. Kompletní framework Symfony je určen pro tvorbu webových stránek za pomoci architektury MVC. Jelikož serverová aplikace poskytuje pouze REST API, použil jsem pouze část Symfony komponent. Zejména jsem vynechal vrstvu view.

Při implementaci jsem se snažil dodržovat následující základní implementační principy jako například KISS (Keep it simple stupid), DRY (Don't repeat yourself), SoC (Separation of concerns) a dalších. Zdrojový kód jsem se rozhodl rozdělit dle funkcionalit: kde v jednom adresáři jsem umístil všechny controllery a další adresáře byly poté voleny dle hlavních entit databázového modelu a obsahovaly veškeré třídy vykonávající aplikační logiku spojenou s danou entitou.

3.1.1 Controllery

Vrstva controller obsahuje třídy obsluhující REST rozhraní. Jedná se o poměrně jednoduché metody, které identifikují konkrétní zdroj a požadovanou operaci a provedou obsluhu požadavku. Controller provede autentizaci a autorizaci uživatele, poté s pomocí vrstvy model vykoná danou operaci a výsledek operace serializuje a odešle zpět klientské aplikaci. Ukázka zdrojového kódu controlleru je k vidění v úryvku zdrojového kódu 3.1.

Přiřazení URL ke controllerům (routing) zprostředkovává Symfony. Jednotlivé URL lze přiřadit několika způsoby: konfigurací v YAML,XML, PHP nebo přímo v kódu jednotlivých controller tříd za pomocí PHP anotací. Nově

pro verze PHP 8.0 a vyšší je možné požít nativní atributy. Já jsem v implementaci vyžil routing pomocí PHP anotací. Symfony aplikace v podstatě funguje tak, že odchytí každý požadavek na server, dle nastavení routování spustí příslušnou controller metodu s daným parametry a návratovou hodnotu transformuje do validní HTTP odpovědi a odešle zpět. Úkolem programátora je správně nakonfigurovat routing a poté implementovat vlastní logiku do controller metod.

```
/**
 * @Route(
 *       "/api/my-litters/{id<\d+>}",
 *        name="get_my_litter",
 *        methods={"GET"}
 * )
 */
public function getMyLitter(
    int $id,
    LitterRepository $litterRepository
) : Response
{
    $litter = $litterRepository->findById($id);

    if(!$litter){
        throw $this->createNotFoundException();
    }

    $this->denyAccessUnlessGranted(
                Actions::READ,
                $litter
    );

    return $this->createJsonResponse($litter);
}
```

Úryvek zdrojového kódu 3.1: Controller pro získání uživatelova vrhu

3.1.2 Autentizace a autorizace

Symfony umožňuje mnoho různých implementací autorizace a autentizace. Ty nejběžněji využívané už má vestavěné, avšak umožňuje i vlastní implementaci. Velmi zjednodušeně řečeno, základem bezpečnosti je validace přihlašovacích údajů, rozeznání a poskytnutí konkrétního uživatele a rozhodování o jeho pravomocech.

3.1.2.1 Autentizace

V této práci jsem se rozhodl pro implementaci autentizace za pomoci již vestavěné funkcionality, kdy uživatelé se uchovávají v databázi a přihlášení mezi jednotlivými požadavky je přenášeno pomocí PHP sessions. Každý uživatel je reprezentován jednou entitou v databázi a je u něj mimo jiné uloženo unikátní přihlašovací jméno (v tomto případě email) a hash hesla. Při přihlášení aplikace porovná přihlašovací údaje vůči databázi uživatelů a rozhodne, zda se lze s údaji přihlásit. Pokud ano, serializuje příslušnou entitu uživatele do PHP sessions[3]. Na základě této funkcionality je aplikace schopna při vykonávání dalších požadavků identifikovat uživatele bez nutnosti opětného přihlášení.

Většina výše popsané funkcionality je již v Symfony obsažena a mně tedy zbývalo pouze vše správně nakonfigurovat, implementovat třídu Authenticator a entitu User. Tato třída slouží pro ověření přihlašovacích údajů a přiřazení jím dané entity uživatele. Rovněž také generuje HTTP odpovědi na požadavky ohledně přihlášení a odhlášení.

3.1.2.2 Autorizace

Autorizace je proces, ve kterém aplikace rozhoduje, zda klient může provést danou operaci či nikoliv. Symfony má tuto funkcionalitu zabudovanou již v sobě, nicméně její použití závisí na konkrétní implementaci a konfiguraci. V podstatě je proces v autorizaci implementován tak, že pro dvojici parametrů atribut a subjekt Symfony vrátí hodnotu ano/ne. Kde atribut reprezentuje prováděnou akci nebo vlastnost a subjekt může upřesnit specifikaci, na co se bude atribut vztahovat.

Autorizace může být nakonfigurována pomocí ACL, kde lze předpisu URL pomocí regulárního výrazu přiřadit notný atribut pro přístup na tuto URL. ACL poté symfony hlídá sama a pro nevyhovující požadavky ihned vrací odpovídající HTTP odpovědi. Pro více detailní rozlišení uživatelových práv je tato funkcionalita implementovaná pomocí služby Security, která disponuje metodou isGranted. Rovněž lze tuto funkcionalitu použít v controlleru za pomoci metody denyAccesUnlessGranted, která interně rozhodne o povolení přístupu a v případě zamítnutí ukončí aplikaci a vrátí vhodnou HTTP 403 odpověď stejně jako při ACL.

Pravidla pro rozhodnutí, zda má uživatel potřebné oprávnění, lze konfigurovat následujícími způsoby:

- **Uživatelské role**

  Symfony umožňuje uživatelům přiřazovat role, ke kterým lze poté přiřadit jejich pravomoce. Role se poté dá použít v ACL nebo jako atribut pro

[3]PHP sessions se přenáší pomocí HTTP Cookies. Server ukládá jednotlivá data do svého úložiště pod jednoznačným identifikátorem. Identifikátor poté přenáší pomocí HTTP Cookies a s jeho pomocí je schopen data přiřadit i k následujícím požadavkům od stejného klienta.

autorizaci. U rolí lze definovat jejich hierarchii,. Role mohou převzít pravomoce jiných rolí, s jejich pomocí je tedy možné pokrýt obrovskou část autorizace. Ve výchozím stavu Symfony přiřazuje uživatelům systémové role jako je například nepřihlášený uživatel nebo přihlášený uživatel. Další role lze definovat a přiřadit uživatelům na základě jejich implementace.

- **Votery**

  V případě, že autorizace na základě uživatelských rolí není dostačující, je možnost implementace Voterů. Voter je třída, která umožňuje vlastní implementaci logiky pro rozhodování o pravomocech. Je tedy možné definovat vlastní atributy a tuto funkcionalitu vztáhnout ještě na konkrétní subjekt. Typický případ užití Voterů je správa komentářů, kdy komentáře může přidávat jakýkoliv přihlášený uživatel, ale upravovat konkrétní komentář může pouze ten uživatel, který jej napsal.

Dle návrhu je v serverové části pro autorizaci potřeba rozlišovat, zda je uživatel přihlášený, či nikoliv, a také v jakém je stavu ke konkrétnímu subjektu (Například rezervaci si může zobrazit pouze majitel vrhu nebo majitel samotné rezervace). V implementaci jsem proto vyžil výchozí Symfony role pro rozlišení přihlášeného a nepřihlášeného uživatele v kombinaci s individuálně implementovanými Votery. Ukázka voteru je znázorněna v úryvku zdrojového kódu 3.2.

3.1.3 Práce s databází

Ve výchozím nastavení v Symfony aplikaci zprostředkovává spojení s databází knihovna Doctrine 2[19]. Doctrine 2 funguje na principu Object-relational mapping (ORM).

3.1.3.1 ORM

ORM spočivá v mapování dat z databáze do objektů používaných v aplikaci a naopak. Používá se v aplikacích vyvíjených v objektově orientovaném programovacím paradigmatu. Místo přímého kontaktu s databází a psaní SQL dotazů, využívají programátoři pouze objekty, které reprezentují databázové entity a obsahují proměnné, do kterých jsou promítnuty jednotlivé atributy databázového modelu. Doctrine 2 umožňuje ORM i jiných, než relačních databází (např. některé NoSQL databáze). V této práci je však používán pro spojení s relační databází MariaDB, proto se v popisu omezím pouze na relační databáze.

Entitní třídy Jak již bylo řečeno, ORM mapuje databázové entity do objektů aplikace. Tyto objekty se nazývají entitní třídy Programátor tedy musí

```
    /**
     * @param mixed|Dog $subject
     */
    protected function voteOnAttribute(
        string $attribute,
        $subject,
        TokenInterface $token
    ) : bool
    {
        $currentUser = $token->getUser();

        if($attribute === Actions::READ){
            return true;
        }

        if(!$currentUser instanceof User){
            return false;
        }

        return(
             $currentUser->getId()
                 ===
             $subject->getOwner()->getId()
         );
    }
```

Úryvek zdrojového kódu 3.2: Voter definující pravomoce pro psy

implementovat jednotlivé entitní třídy, do kterých poté budou mapovány databázové entity. Tyto třídy je nutno nakonfigurovat tak, aby Doctrine 2 věděl, jak je překládat do databáze, což je opět možné několika způsoby. V implementaci této práce jsem opět zvolil PHP anotace. Příklad entitní třídy je k vidění v úryvku kódu 3.3.

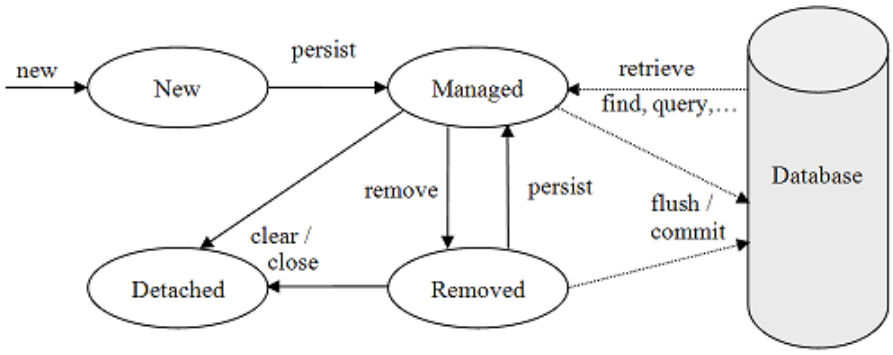
Entitní třídy se mohou nacházet ve čtyřech fázích svého životního cyklu,na základě kterých poté entityManager vykonává dotazy do databáze. Diagram přechodů mezi jednotlivými fázemi je k vidění na obrázku 3.1.

- **New** Entitní třída je nová, existuje v aplikaci, ale entityManager o ní ještě neví, a tedy ji ignoruje. Zpravidla se jedná o entity, které aplikace vytvoří a ještě nebyly persistovány entityManagerem.

- **Managed** Entitní třída je monitorována EntityManagerem a při zavolání metody flush budou její data propsána do databáze.

- **Detached** Entitní třída momentálně není monitorována entityManagerem a při zavolání metody flush nebudou její data propsána do databáze i v případě že by se změnila.

- **Removed** Entitní třída je monitorována entityManagerem a při zavolání metody flush bude z databáze odstraněna.



Obrázek 3.1: Diagram fází životního cyklu entitních tříd a jejich přechody [20]

EntityManager Práci s entitními třídami umožňuje entityManager. Jedná se o službu, která si uchovává informace o entitách a zprostředkovává jejich překlad z/do databáze. Princip ORM nepoužívá jednotlivé dotazy pro uložení dat do databáze, ale v jednom momentu (při zavolání metody flush) propíše obsah všech entitních tříd ve stavu managed do databáze a zároveň z databáze odstraní entity entitních tříd ve stavu removed.

Doctrine query language Jelikož ORM nepracuje s databázovými entitami, ale s objekty, které je reprezentují, není možné se dotazovat na objekty pomocí SQL, ale za pomoci jiného (objektově orientovaného) dotazovacího jazyka. U Doctrine 2 se jedná o Doctrine query language (DQL). Je velmi podobný s SQL, nicméně je založen na dotazování se na objekty, což často mate začátečníky s ORM. Doctrine poskytuje queryBuilder, který umožňuje skládat DQL dotazy takzvaným „fluent“ zápisem. Ukázka skládání DQL dotazu je k vidění v úryvku zdrojového kódu 3.4.

Hydratace Pojem hydratace označuje proces mapování dat z výsledku databázového dotazu do objektů v aplikaci. Doctrine umožňuje několik typů hydratace, a také je možné implementovat vlastní hydrataci.

```
/**
 * @ORM\Entity()
 */
class CoatColor
{

    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private int $id;

    /**
     * @ORM\Column(type="string")
     */
    private string $colorName;

    public function __construct(
        string $colorName
    )
    {
        $this->colorName = $colorName;
    }

    public function getId(): int
    {
        return $this->id;
    }

    public function getColorName(): string
    {
        return $this->colorName;
    }

}
```

Úryvek zdrojového kódu 3.3: Entitní třída reprezentující psí plemeno

```
$qb = $this->entityManager->createQueryBuilder();

$qb
    ->from(Breed::class, 'breed')
    ->select('breed')
    ->orderBy('breed.breedName');
```

Úryvek zdrojového kódu 3.4: Skládání DQL dotazu pomocí queryBuilderu

Migrace Doctrine 2 disponuje funkcionalitou validace databázového schématu oproti nakonfigurovaným entitním třídám. Dokonce dokáže i vygenerovat SQL dotaz tak, aby databázové entity synchronizoval s konfigurací entitních tříd. Díky tomuto lze pomocí Doctrine 2 generovat migrace. Jedná se o třídy, které dokáží do databáze spustit sekvenci SQL dotazů s cílem předpřipravit databázový model.

3.1.3.2 Výkonnostní optimalizace

Doctrine 2 je velmi mocný nástroj pro práci s databází, bohužel s sebou přináší i řadu výkonnostních problémů, o kterých by programátor měl vědět a umět je vyřešit.

Načítání relací Z principu ORM musí entitní třídy obsahovat ostatní entitní třídy, se kterými jsou v relaci. To by ale v praxi mohlo znamenat načítání obrovského množství dat z databáze. Rozlišujeme tedy 3 typy načítání relací:

- **Eager** Načte data relací ihned při hydrataci původní entitní třídy
- **Lazy** Nahradí relaci tzv. Proxy třídou, která se chová stejně jako očekávaná třída, ale neobsahuje data z databáze. Data se načtou dodatečným požadavkem do databáze až při prvním přístupu.
- **Extra lazy** Speciální varianta Lazy načítání, která umožňuje data načítat po částech.

Mezi těmito způsoby načítání musí programátor volit tak, aby se data načetla co nejoptimálněji. Na jedné straně je extrém, kdy by se všechna data načítala eager metodou a ve výsledku by se načetlo ohromné množství dat, které by ani nemuselo být potřeba načítat. Na druhé straně je extrém kdy by se všechny relace načítaly metodou lazy. To by sice zajistilo minimální objem načtených dat, zato by se každá využitá relace načítala dodatečným dotazem do databáze a databáze by byla zahlcena spoustou jednoduchých dotazů, které ovšem mohly být načteny pouze jediným dotazem. Při implementaci DQL dotazů je tedy nutno, aby si programátor rozmyslel které relace budou následně

využity (a ty načítat metodou eager) a které nebudou (ty načítat metodou lazy).

N+1 problém N+1 problém nastává při načítání 1:M a M:N relací metodou eager. Pro demonstraci si představme dvě entity, Plemeno a Pes, které jsou v relaci 1:N. Pokud spustíme dotaz pro získání všech plemen a všech jejich psů metodou eager, bude do databáze odeslán SQL dotaz, který vrátí všechny tyto entity pomocí klauzule JOIN. Nicméně výsledek bude obsahovat M * N n-tic, kde bude ale pouze M unikátních n-tic atributů entity plemeno, tudíž výsledky budou obsahovat M * N - M duplicitních n-tic atributů plemeno. Pokud bychom uvažovali další 1:N relaci entity psa (například Štěně), kterou bychom načetli také metodou eager, bylo by ve výsledku N * M * O n-tic. Unikátních n-tic entity plemeno bude M a unikátních n-tic entity Pes bude M*N, tedy bude M * N * O - M duplicitních n-tic atributů entity Plemeno a N * M * O - N * M duplicitních n-tic atributů entrity Pes. Počet duplicitních n-tic dat tedy roste exponenciálně s každým eager načtením relace 1:N nebo N:M!

V tomto případě je vhodné načíst nejdříve jedním dotazem všechny chtěné entity Plemeno, následně na základě načtených entit Plemeno dalším dotazem načíst veškeré entity Pes a poté třetím dotazem načíst entity Štěně. V tomto případě jsme se vyhnuli veškerým duplicitním n-ticím atributů za cenu několika málo dotazů do databáze navíc. Počet těchto dotazů roste pouze lineárně s počtem načítaných relací.

Velké množství managed entitních tříd a nadměrné používání metody flush EntityManager v momentě zavolání metody flush zkontroluje veškeré managed entitní třídy, zda se obsahují změnu. Na základě změněných entitních tříd poté sestaví a odešle SQL dotaz do databáze. Pokud však entity manager eviduje mnoho managed entit, které nejsou změněny, provádí jejich kontrolu zbytečně a zpomaluje a zatěžuje aplikaci. Pokud se navíc flush volá často v momentech, kdy to není třeba, tyto zbytečné kontroly se násobí. Je tedy vhodné, aby to programátor uvážil a nepotřebné entity uchovával ve stavu detached. Rovněž je vhodné volat funkci flush co nejméně, ale zase je nutno zajistit alespoň jedno její zavolání na konci veškerých úprav entitních tříd, aby se změny propsaly do databáze.

3.1.3.3 Implementace v aplikaci

Pro dodržení SoC byly třídy pro práci s databází rozděleny do tří typu. Již výše zmíněné entitní třídy, repository a facade. Každá entitní třída, u které to dává smysl, má vlastní repository a facade.

Repository slouží k načítání dat z databáze. Uvnitř metod těchto tříd probíhá skládání a optimalizace DQL dotazů a jejich spouštění a návratové hodnoty jsou výsledky těchto dotazů.

Facade slouží k vytváření, modifikování a mazání entitních tříd. Jedná se o zapouzdření těchto operací, které obsahuje příslušné volání a optimalizaci entityManageru.

3.1.4 Ostatní funckionality

Mezi ostatní funkcionality patří zejména pomocné třídy pro dodržení principu DRY, a také třídy, které slouží k transformaci a validaci dat z HTTP požadavků do podoby, kterou umí zpracovat jednotlivé facade.

Nosné objekty, které reprezentují extrahovaná data z HTTP požadavků, jsem pojmenoval Request. Umožňují objektově reprezentovat data požadavků a rovněž slouží k jejich validaci. K definici očekávaných dat jsem použil Symfony formuláře. Je to velmi mocný nástroj pro tvorbu a validaci HTML formulářů. Jelikož v této práci serverová část aplikace slouží pouze pro obsluhu REST API, velkou část jsem nevyužil. Každá skupina dat očekávaných v požadavku má své FormType a Request objekty. Validace spočívá ve využití validátorů již obsažených v Symfony v kombinaci s vlastními validátory, které jsou implementovány přímo pro konkrétní potřeby této aplikace. Příklad vlastního validátoru je možné vidět v úryvku zdrojového kódu 3.5. Případné chyby jsou poté serializovány a odeslány jako součást HTTP 400 odpovědi. Kvůli jazykové lokalizaci jsou reprezentovány pouze kódy, které následně překládá klientská aplikace.

3.2 Klientská část

Klientská část byla implementována v javascriptovém frameworku React verze 17.0 a s pomocí dalších javascriptových knihoven. Při návrhu jsem vycházel především z navrženého lo-fi prototypu.

3.2.1 Rozdělení do komponent

React je framework pomocí kterého lze poměrně jednoduše vytvářet interaktivní uživatelské rozhraní. Je založený na samostatných komponentách, které jsou zapouzdřeny a udržují si svůj interní stav. Komponenty poté mezi sebou interagují a lze je skládat a vytvořit z nich komplexní uživatelské rozhraní.

Při rozdělování uživatelského rozhraní do komponent jsem vytvořil samostatné komponenty pro každou obrazovku rozhraní. Tyto komponenty obsahují veškerou interakční logiku a starají se o načtení hlavních dat pro stránky, která potom distribuují do dalších subkomponent. Další komponenty jsem vytvořil dle opakujících se bloků prvků v návrhu lo-fi prototypu. Tyto komponenty zpravidla vykreslují obdržená data z rodičovské komponenty a odchytávají uživatelskou interakci, kterou distribuují do rodičovské komponenty.

```
final class DogSexValidator extends ConstraintValidator
{

    public function validate(
        $value,
        Constraint $constraint
    )
    {
        if(!$constraint instanceof DogSexConstraint){
            throw new UnexpectedTypeException(
                    $constraint,
                    DogSexConstraint::class
            );
        }

        if(!$value instanceof Dog){
            return;
        }

        if($value->getSex() !== $constraint->sex){
            $message = $constraint->sex === DogSex::MALE
                    ? $constraint->maleMessage
                    : $constraint->femaleMessage;

            $this->context->buildViolation($message)
                ->addViolation();
        }
    }
}
```

Úryvek zdrojového kódu 3.5: Validátor pohlaví psa

Pro sdílení dat, která jsou společná pro celou aplikaci, jsem použil úložiště Redux. Jedná se např. o údaje přihlášeného uživatele nebo aktuálně zvolený jazyk aplikace.

3.2.2 Vzhled a rozložení aplikace

Pro vzhled aplikace jsem použil knihovnu Shards React[21]. Jedná se o knihovnu, která obsahuje předpřipravené grafické React komponenty, které se běžně využívají ve webových aplikacích. Tyto komponenty jsou jednoduše konfigurovatelné. Většina komponent pro vzhled a rozložení uživatelského rozhraní je sestavena z komponent open source verze této knihovny. Ukázka použití je k vidění v úryvku zdrojového kódu 3.6.

```
<Breadcrumb>
    <BreadcrumbItem>
        <Link to={APP_URL_HOMEPAGE}>
            <Translate value="breadcrumb.homepage" />
        </Link>
    </BreadcrumbItem>
    <BreadcrumbItem>
        <Link to={APP_URL_MY_LITTERS}>
            <Translate value="breadcrumb.myLitters" />
        </Link>
    </BreadcrumbItem>
    <BreadcrumbItem>
        {this.state.litter.name}
    </BreadcrumbItem>
</Breadcrumb>
```

Úryvek zdrojového kódu 3.6: Implementace drobečkové navigace pomocí knihovny React Shards

3.2.3 Jazyková lokalizace

Aplikace má potenciál být celosvětová, tudíž je potřeba ji distribuovat ve více jazykových mutacích. Pro tento účel jsem zvolil knihovnu react-redux-i18n, která rozšiřuje knihovnu react-i18nify o propojení na reduxové úložiště. Aktuální jazyk aplikace je uložen v reduxovém úložišti. Jednotlivé překlady se nadefinují do jednoho nebo i více javascriptových objektů a knihovna poté poskytuje relevantní překlady. Pro implementaci do Aplikace poskytuje knihovna komponentu Translate, která je napojena na reduxové úložiště a překresluje svůj obsah po změně aktuálního jazyka.

Pro začátek jsem aplikaci přeložil do českého a anglického jazyka. Český jazyk jsem zvolil z důvodu, že aplikace bude prvotně spuštěna pro české

uživatele a anglický jazyk jsem zvolil, protože je ve světě velmi rozšířený. Veškeré překlady se drží v klientské aplikaci, serverová část pouze odesílá kódy odpovědí, které jsou poté v klientské aplikaci přeloženy do konkrétního jazykového ekvivalentu. Jedná se převážně o chybové hlášky.

3.2.4 Routing

Routing jsem implementoval pomocí knihovny react-router-redux. Tato knihovna umožňuje podmíněné vykreslování komponent na základě url konkrétní URL. Navíc dokáže svou historii uchovávat v úložišti reduxu a s jeho pomocí umožňuje procházet jednotlivé stránky a historii aplikace bez nutnosti znovunačtení celé stránky. Rovněž obsahuje připravené komponenty pro generování odkazů, které umožňují procházet mezi jednotlivými stavy aplikace rovněž bez znovunačtení celé stránky.

3.2.5 Komunikace s API

Pro komunikaci s API jsem použil knihovnu axios. Jedná se o jednoduchou knihovnu umožňující vykonávat HTTP požadavky na základě javascriptových promises. Důraz jsem kladl převážně na to, aby výsledek každého požadavku byl v uživatelském rozhraní zřetelně vidět. Ukázka provedení HTTP požadavku pomocí knihovny axios je zobrazena na úryvku zdrojového kódu 3.7.

3.3 Spustitelné prostředí

Mnou vyvinutá webová aplikace potřebuje specifické prostředí pro běh. Správně nakonfigurovaný Apache server s nainstalovaným PHP a některými jeho rozšířeními a navíc také databázový server. Jelikož se zatím jedná o prototyp aplikace, který není nikde veřejně spuštěn, může být spuštění aplikace v lokálním prostředí obtížné. Z tohoto důvodu jsem se rozhodl prototyp aplikace „kontejnerizovat“ pomocí aplikace Docker. V této sekci jsem čerpal převážně ze své bakalářské práce[22].

3.3.1 Docker

Docker je nástroj, který umožňuje „kontejnerizaci“ softwarových aplikací.Poskytuje platformu s jednotným prostředím pro běh aplikací, které jsou uzavřeny do tzv. kontejnerů. Jednotlivé kontejnery obsahují izolované prostředí umožňující spuštění různých verzí aplikací nezávisle na verzi a stavu hostitelského operačního systému, na kterém je Docker spuštěn. Kontejnery neovlivňují nastavení hostitelského systému, tudíž je lze libovolně vytvářet a mazat bez rizika narušení funkcionality hostitelského systému.

Funkcionalita kontejnerizace je velmi podobná virtualizaci s tím rozdílem, že kontejnery neobsahují vlastní operační systém, ale mají přístup ke zdrojům

```
axios.put(API_BOOKING_ACCEPT(this.state.booking.id))
    .then(success => {
        this.props.notifySuccess({
            title: I18n.t('bookingDetail.acceptSuccess'),
            position: 'tc',
            autoDismiss: 5
        });
        this.setState({
            ...this.state,
            ...{
                booking: success.data
            }
        })
    })
    .catch(error => {
        this.props.notifyError({
            title: I18n.t('bookingDetail.acceptError'),
            position: 'tc',
            autoDismiss: 5
        });
    })
```

Úryvek zdrojového kódu 3.7: Vykonání HTTP požadavku pro potvrzení rezervace

operačního systému, na kterém jsou hostovány a propagují je dovnitř sebe. Díky tomu jsou několikanásobně méně náročné na výpočetní výkon.

Hlavní výhodou, proč jsem se rozhodl využít Docker, je jeho nenáročnost na systémové požadavky. Docker je jednoduše dostupný pro Windows, Linux i MacOs, což umožňuje jednoduché spuštění prototypu aplikace na těchto platformách. Stačí tedy jen, aby na dotyčném počítači byl nainstalován Docker respektive Docker compose, který už podle konfigurace dokáže plnohodnotně spustit prototyp aplikace v kontejnerech tak, aby neovlivnili hostitelský systém.

3.3.2 Konfigurace prostředí

Prostředí aplikace sestává celkově ze tří kontejnerů propojených do jedné sítě. Interně fungují jako celek a jsou kompletně izolované od operačního systému. Do systému jsou namapovány pouze 2 porty, které slouží pro komunikaci s aplikací. Celkové prostředí se sestaví i se základními daty pro lepší prezentaci funkcionality aplikace.

MariaDB kontejner je kontejner, který uchovává server s plně nakonfigurovanou a inicializovanou databází.

App kontejner je kontejner, který obsahuje nainstalovaný apache web server s PHP, který slouží pro spouštění skriptů aplikace. Do hostujícího systému propisuje port 8888, na kterém server poslouchá. Slouží pro přístup a spouštění k prototypu aplikace.

Mailhog kontejner je pomocný kontejner, který obsahuje nástroj MailHog. Mailhog je nástroj usnadňující vývoj odesílání e-mailů. Simuluje SMTP server a zachytává veškeré odesílané zprávy. Ty je poté možno detailně prohlížet v jeho grafickém rozhraní. Kontejner jsem do prostředí umístil z důvodu, že aplikace používá dvoufázové ověření pomocí e-mailu, a proto je nutno e-maily zachytávat.Tento kontejner do hostujícího systému propisuje port 8889, kde poslouchá webová aplikace pro procházení odesílaných emailů.

Kapitola 4

Testování

Testování je neodmyslitelnou součástí životního cyklu vývoje jakékoli aplikace. Jeho hlavním úkolem je zjistit chyby a nedostatky vyvinuté aplikace, ale také zjistit její použitelnost koncovými uživateli. Neodhalení některých chyb před spuštěním aplikace by mohlo mít fatální následky, ať už se jedná o zbytečné náklady navíc, ztráta prostředků nebo například únik citlivých dat. Některé typy testování je vhodné provádět již v průběhu návrhu a vývoje, protože čím dříve je chyba odhalena, tím méně nákladné je její odstranění.

Při testování je potřeba brát na vědomí, že testování může pouze dokázat přítomnost chyb, nemůže však zaručit bezchybnost aplikace.. [23]

4.1 Automatizované testování

Automatizované testování je prováděno pomocí počítače. Zpravidla se jedná o pevně definované postupy, které jsou strojově vykonávány. Jeho největší předností je snadná znovupoužitelnost, kdy je potřeba test pouze jednou definovat, a poté jej lze automatizovaně vykonávat dle libosti. Šetří tedy lidský čas. Automatizované testy jsou tedy nejvíce efektivní, pokud se vykonávají vícekrát. Jsou tedy ideální pro průběžnou kontrolu při vývoji, zda nově implementovaná funkcionalita neporušila tu starou již funkční. Jejich výrazným nedostatkem je obtížnost, časová náročnost a mnohdy i nerealizovatelnost pokrytí veškerých funkcionalit. Většinou se nevyplácí mít automatizované testy pro veškerou funkcionalitu a v praxi se takto často testuje jen část důležité funkcionality. Zbylé testování je prováděno lidskými testery.

V této práci jsem automatizované testování využil pouze na serverovou část aplikace. Testy jsem implementoval až při dokončování implementace, čímž jsem se ochudil o jejich přínos v průběhu implementace, na druhou stranu budou velmi užitečné při dalším vývoji.

4.1.1 Statická analýza kódu

PHP patří mezi skriptovací jazyky, které jsou kompilovány až při spuštění a mají dynamickou typovou kontrolu. To mimo jiné přináší nevýhodu v detekci chyb způsobenou použitím špatných typů proměnných. U kompilovaných jazyků tyto chyby dokáže odhalit kompilátor, tudíž nejdou s těmito chybami vůbec spustit. Skriptovací jazyky tuhle vlastnost (z jejich principu) postrádají, proto se tyto chyby projeví až při spuštění skriptu. Pokud taková chyba nastane v produkčním prostředí, může mít fatální následky.

Pro kompenzaci tohoto nedostatku existují nástroje, které provádí statickou analýzu kódu. Statická analýza má za úkol odhalit možné nedostatky ve zdrojovém kódu bez nutnosti kód spouštět. Skriptovací jazyky často nemají pevně definované datové typy, je tedy nutno nástroji pro statickou analýzu definovat očekávané typy proměnných. V PHP se tak dá učinit pomocí anotací, v novějších verzích je možné využít pevně definované datové typy.

V mém případě se jednalo převážně o špatné použití datových typů, používání nedefinovaných funkcí a proměnných. Jako nástroj pro statickou analýzu jsem využil PHPStan - PHP Static Analysis Tool.

> „PHPStan se zaměřuje na hledání chyb ve vašem kódu, aniž byste jej skutečně spouštěli. Zachytí celé třídy chyb ještě předtím, než pro něj napíšete testy. Posouvá PHP blíže ke kompilovaným jazykům v tom smyslu, že správnost každého řádku kódu lze zkontrolovat před jeho spuštěním.“ [24]

4.1.2 Unit testování

Unit testování (česky taktéž jednotkové testování) je způsob automatizovaného testování zdrojového kódu, který se zaměřuje na ověření správnosti implementace jednotlivých atomických částí (jednotek) zdrojového kódu. Jednotkové testování je hojně využíváno v extrémním programování a v testy řízeném vývoji, nicméně je vhodné i pro další metodiky vývoje software.

Návrhy jednotkových testů je vhodné dělat ještě před psaním zdrojového kódu (testy řízený vývoj) nebo při jeho psaní. Tento přístup přináší benefity v podobě lepšího pochopení použitelnosti zdrojového kódu a jeho uskupení do logických, jednoduše testovatelných jednotek s pevně definovanými vlastnostmi a funkcionalitou. Rovněž ulehčí rozvrhnutí jednotlivých komponent pro jejich lepší znovupoužitelnost.

Pro nejefektivnější pokrytí zdrojového kódu je vhodné navrhovat pozitivní i negativní jednotkové testování. Při pozitivním testování se testuje, zda správně použitá jednotka se chová dle očekávání a vrací správný výsledek. Při negativním testování se zase testuje, zda při špatném použití jednotka rozezná a evokuje chybu dle očekávání. [23]

Pro jednotkové testování jsem použil nástroj PHPUnit, který je velmi rozšířený pro vykonávání jednotkových testů aplikací psaných v jazyce PHP[25].

4.1.3 Symfony validátory

Jelikož je Symfony sestavená z jednotlivých knihoven napsaných v PHP. Některé vyžadují specifické použití nebo konfiguraci. Symfony obsahuje nástroje pro validaci použití těchto konfigurací. V podstatě se jedná o rozšíření statické analýzy kódu, která je specificky zaměřená na správnost použití určitých Symfony komponent.

V projektu jsem z těchto nástrojů využil YAML linter, což je nástroj pro validaci syntaktické správnosti konfiguračních souborů ve formátu YAML. Další nástroj jsem využil Doctrine validator, který provádí validaci konfigurace Doctrine entitních tříd, a také porovnává správnost konfigurace oproti databázovým entitám.

4.1.4 Code sniffer

Nástroj code sniffer se nepoužívá k testování správnosti implementace, nýbrž pomáhá udržovat kvalitu a čitelnost zdrojového kódu. Zdrojový kód může být napsán mnoha způsoby (způsoby odsazení, využívání nových řádků, psaní komentářů, atd.). Kombinace těchto způsobů razantně snižuje jeho čitelnost, je tedy žádoucí definovat jeden programovací standard a toho se držet.

Jako code sniffer jsem použil PHP Code sniffer [26]. Dokáže validovat dodržení definovaného standardu a v mnoha případech dokáže i zdrojový kód automatizovaně upravit tak, aby standard dodržoval.

4.2 Uživatelské testování

Uživatelské testování je jednou ze základních testovacích metod softwaru a často je nenahraditelné. Dokáže přiblížit, jak aplikaci využívají její koncoví uživatelé, a odhalit problémy, se kterými se stýkají. Často se objeví nedostatky v aplikaci, které vývojáři vůbec neočekávali, jelikož koncoví uživatelé nahlíží na aplikaci rozdílným pohledem. Roli také hrají jejich zkušenosti s využíváním podobných technologií a aplikací. V této sekci jsem čerpal převážně z knihy Usability Engineering[18].

Uživatelské testování se řadí mezi kvalitativní metody ověření použitelnosti aplikace. Probíhá na vybrané skupině respondentů, kteří reprezentují cílovou skupinu uživatelů aplikace. Tito uživatelé pod dohledem moderátora testu provádí jednotlivé úkoly s aplikací. Jejich reakce je poté vyhodnocena ve výsledcích testu.

Uživatelské testování jsem provedl, protože dokáže otestovat aplikaci z více pohledů. Přiblíží reakce finálních uživatelů na aplikaci, ověří intuitivnost a použitelnost uživatelského rozhraní a také může odhalit případné chyby ve funkcionalitě.

4.2.1 Návrh testovacích scénářů

Prvním krokem při návrhu uživatelských testů je ujasnit si, jaké výsledky jsou od testu očekávány. Poté je potřeba určit jejich průběh, prostředí ve kterém budou vykonávány a definovat scénáře. V této práci od testů očekávám informaci o použitelnosti výsledného prototypu aplikace. Jednotlivé uživatelské testy by tedy měly vykonat hlavní funkcionality aplikace. Výsledky testů by poté měly obsahovat problémové funkcionality aplikace, které byly pro uživatele obtížné nebo nemožné vykonat.

Před začátkem uživatelského testu, moderátor vše důkladně vysvětlí testovacímu uživateli a ujistí se, že uživatel pochopil průběh a důvod testování. Je nutné uživateli zdůraznit, že se netestují jeho schopnosti, ale aplikace tak, aby uživatel se necítil frustrován pokud nedokáže některý z úkolů splnit. Následně začne samotné testování, moderátor bude pokládat uživateli jednotlivé úkoly, a uživatel se je bude snažit vykonat. Zde je důležité, aby moderátor průběh testu nijak neovlivňoval. Jelikož některé úkoly vyžadují úspěšné vykonání předchozích úkolů, moderátor poskytne uživateli nápovědu, pokud nebude v plnění úkolu projevovat úspěšný postup po déle jak 1 minutu. Samotné testování bude nahráváno. Je nutné zaznamenat obrazovku mobilních telefonů i reakce testovacích uživatelů.

Testy budou uživatelé vykonávat na svých mobilních telefonech. Server spustím na svém počítači, kde poběží aplikace s předpřipravenými základními daty potřebnými pro vykonání testu. Rovněž na svém počítači poskytnu testovacím uživatelům aplikaci MailHog, která bude simulovat jejich e-mailového klienta. Moderátor před začátkem testu seznámí uživatele s aplikací MailHog.

Jednotlivé úkoly testovacího scénáře je vhodné navrhovat tak, aby co nejvíce kopírovaly jednotlivé očekávané reálné využití aplikace. Měly by pokrývat všechny důležité funkcionality, a také by neměly být příliš komplexní ale ani triviální. Pokud uživatel hned při prvním úkolu selže, může se začít cítit nepohodlně, což může mít negativní vliv na vykonávání dalších testů. Je tedy vhodné začít jednoduššími úkoly. Při navrhování testovacích scénářů jsem využil funkční požadavky aplikace tak, aby úkoly pokrývaly veškeré funkční požadavky s prioritou vyšší než 3. Níže jsou k vidění jednotlivé úkoly:

1. Zaregistrujte a přihlaste se do aplikace. Můžete použít nepravdivé údaje.
2. Plánujete si pořídit štěně. Vyhledejte si vrh, který Vám vyhovuje a požádejte o rezervaci štěněte.
3. Pro následující úkoly je připraven uživatelský účet s již předpřipravenými daty. Prosím odhlaste se z vašeho účtu a přihlaste se do účtu pod údaji e-mail: john.doe@example.com, heslo: 123456789.
4. Pořídil(a) jste si nového psa, fenu plemene Sibiřský Husky. Přidejte jej do aplikace.Jmenuje se Sněženka, narodila se 19.11.2020, její rodiče neznáte a zatím ještě nebyla očipována. Barvu srsti má černou.

5. Právě jste přijel(a) od veterináře, který Sněženku očipoval. Má čip s číslem 123456. Upravte Sněžence číslo čipu.

6. Se Sněženkou plánujete vrh na jaře roku 2022, kdy ji budete krýt s Vaším psem Jerrym. Zadejte tento plánovaný vrh do aplikace. K tomuto vrhu vytvořte dvě podmínky pro rezervaci.

7. Prohlédněte si přijaté rezervace pro Váš jarní vrh A. Alespoň jednu odmítněte a alespoň jednu přijměte.

8. Vašemu zimnímu vrhu B se narodilo jedno štěně. Je to fena a jmenuje se Bonnie, narodila se 18.12.2021 a má šedou barvu srsti. Čip ještě nemá.

9. Zvolte si jednu rezervaci zimního vrhu B a přiřaďte jí Bonnie.

10. S Bonnie jste byli na veterinární prohlídce, kde proběhlo odčervení. Evidujte tuto skutečnost v aplikaci.

11. Bonnie je již dostatečně stará a zájemce si ji převzal. Převeďte vlastnictví štěněte na zájemce.

4.2.2 Volba testovacích uživatelů

Testovací uživatele jsem volil tak, aby reprezentovali potencionální uživatele aplikace. Vyhledal jsem tedy lidi, kteří někdy v minulosti kupovali nebo plánují v budoucnosti koupit psa s PP. Rovněž jsem volil uživatele, kteří již někdy uskutečnili vrh svých psů, nebo to plánují v budoucnu. Jelikož testovací data jsou v angličtině je nutno, aby testeři uměli anglicky.

Dle Nielsena poměr benefitů k nákladům testování začne postupně s rostoucím počtem uživatelů klesat. Díky tomu lze určit počet uživatelů tak, aby byly optimalizovány náklady testů a jejich přínos. Graf tohoto poměru pro „typický projekt střední velikosti“ je k vidění na obrázku 4.1. Dle tohoto jsem určil počet testovacích uživatelů. Testy budu provádět pro zjednodušení se čtyřmi testery, ale počet pro nejlepší poměr cena-přínos by byl 6-9 testovacích uživatelů.

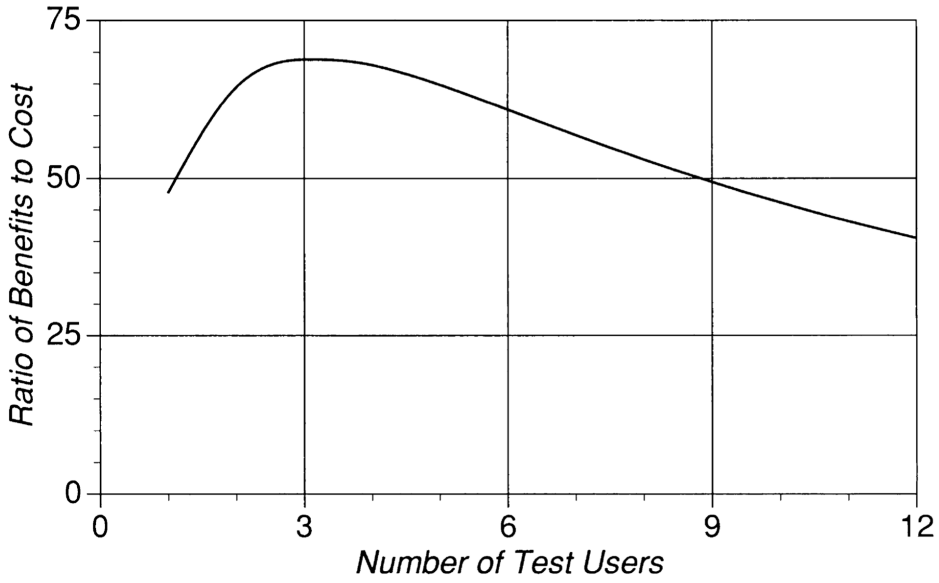
Testovací uživatele jsem zvolil následovně:

- Testovací uživatel A

  Muž, 27 let, vlastní psa bez PP. V budoucnosti si plánuje pořídit psa s PP. Manažer v bance.

- Testovací uživatel B

  Žena 22 let, vlastní fenu s PP. V budoucnu plánuje fenu uchovnit a uskutečnit alespoň jeden vrh. Studentka.



Obrázek 4.1: Graf poměru nákladů ku přínosům uživatelského testování pro „typický projekt střední velikosti“.

[18]

- Testovací uživatel C

  Muž 30 let, vlastní fenu s PP s níž úspěšně odchoval jeden vrh. UI/UX návrhář a grafik.

- Testovací uživatel D

  Žena 23 let, dříve chovala dva psy bez PP, nyní čeká v pořadníku na psa s PP. Studentka.

4.2.3 Průběh testování a nalezené nedostatky

Role moderátora v uživatelských testech jsem se zhostil já. Na začátku každého testu jsem stručně přiblížil aplikaci a vysvětlil používání aplikace Mailhog. Zdůraznil jsem, že se netestují schopnosti uživatele, ale použitelnost aplikace. Neúspěch při vykonání úkolu znamená selhání mne jako vývojáře a ne selhání testera. Poté jsem pomohl testerovi ve webovém prohlížeči v jeho telefonu otevřít spuštěnou aplikaci. Až poté začalo testování. Každý test jsem zaznamenával tak, aby šli vidět uživatelovi reakce a obrazovka telefonu. Videa z testování jsou obsažena na přiložené SD kartě.

Je možné, že aplikace obsahuje chyby, které by snížily efektivitu testování, nebo například úkoly scénáře nejsou dostatečně srozumitelné. Proto jsem se rozhodl nejdříve vykonat jeden pilotní test. Jedná se o úplně první test. Na

základě jeho výsledků je možné aplikaci a testovací scénář upravit. Ostatní testy jsem provedl až na základě vyhodnocení a zpracování pilotního testu.

Každý test zde popíšu včetně nalezených nedostatků. Tyto nedostatky poté ohodnotím dle jejich závažnosti na stupnici 1-5, kde 1 znamená zanedbatelná závažnost a 5 znamená kritická závažnost. K nedostatkům také navrhnu jejich řešení.

4.2.3.1 Testovací uživatel A

S testovacím uživatelem A jsem provedl výše zmíněný pilotní test. Výsledky testu byly katastrofální. Problémem se ukázala být převážně má špatná připravenost na testování. V průběhu testu jsem špatně zvolil testovací místo, takže nás při testování nás rušili okolní vlivy a další obyvatelé domu, kde byl test prováděn. Rovněž jsem před testováním ještě prováděl drobné úpravy zdrojového kódu, u kterých jsem dostatečně neověřil funkčnost, takže aplikace nebyla zcela funkční a v některých případech jsem byl nucen zasáhnout do testování, nebo upravit testovací scénář. Dalším nedostatkem bylo nedostatečné vysvětlení funkcionality MailHog, který simuloval e-mailového klienta a zároveň seznámení s používáním notebooku, na kterém byl spuštěn.

Na druhou stranu, když pominu fatální chyby způsobené mou špatnou připraveností, uživatel úkoly plnil vcelku intuitivně a s výjimkou posledního úkolu je dokázal vykonat. V hodnocení tedy pominu následky mé nepřipravenosti a proberu pouze jimi neovlivněné nedostatky.

Nalezené nedostatky:

- Do pole s datem se snažil napsat datum ve špatném formátu, nakonec datum zvolil přes datepicker.

  Závažnost: 1

  Řešení: Přidat placeholder s formátem data.

- Pokusil se založit vrh z detailu psa, když nenašel vhodné tlačítko, správně hledal v sekci mé vrhy.

  Závažnost: 1

  Řešení: Přidat tlačítko na založení vrhu do detailu mého psa.

- Převedení vlastnictví štěněte hledal v detailu psa. Nakonec využil nejbližší podobnou akci a založil štěněti životní záznam.

  Závažnost: 4

  Řešení: Přidat akci převedení vlastnictví štěněte i do detailu psa.

4.2.3.2 Testovací uživatel B

S testovacím uživatelem B jsem měl testování již dostatečně dobře připravené a mohli jsme se plně věnovat testování. Nalezené nedostatky:

- Snažila se registrovat s nedostatečně dlouhým heslem, jakmile ji aplikace upozornila na chybu, napravila ji. Tento zbytečný krok navíc ji rozhodil a způsobil chybu ve správnosti zopakování hesla.

  Závažnost: 2

  Řešení: Přidat informaci o minimální délce hesla ještě před odesláním formuláře.

- Do pole s datem se snažila napsat datum ve špatném formátu, nakonec po několika pokusech zjistila jaký je správný formát pro datum a zadávala datum ve správném formátu.

  Závažnost: 1

  Řešení: Přidat placeholder s formátem data.

- Do pole pro výběr matky psa omylem zadala hodnotu, hodnota poté nešla smazat.

  Závažnost: 5

  Řešení: Umožnit odstranit hodnotu z pole.

- Informaci o veterinární prohlídce zadala do popisu psa, namísto vytvoření životního záznamu. Nejspíše uživatele nenapadlo, že aplikace umožňuje evidovat životní záznamy.

  Závažnost: 3

  Řešení: Vytvořit dokumentaci, nebo návod, který představí veškeré funkcionality aplikace.

- Nedokázala přiřadit štěně k rezervaci, hledala přiřazení štěněte k rezervaci v detailu psa. Dokonce i po nápovědě, že rezervace musí být nejdříve přijata, si po přijetí rezervace nevšimla změny stavu rezervace a nové možnosti přiřazení štěněte.

  Závažnost: 4

  Řešení: Vytvořit dokumentaci, nebo návod, který představí veškeré funkcionality aplikace. Případně funkcionalitu přidat i do detailu psa.

- Při zobrazení klávesnice se obrazovka přiblížila a zůstala přiblížená.

  Závažnost: 2

  Řešení: Zakázat automatické přiblížení obrazovky

4.2.3.3 Testovací uživatel C

Testovací uživatel C má za sebou již zkušenosti s odchovem vrhu a následném nabízení štěňat. Živí se jako UI/UX návrhář a grafik, takže je s rozhraními

aplikací každodenně v kontaktu. Těmto důvodům přisuzuji fakt, že se v aplikaci poměrně rychle zorientoval a bez větších obtíží dokončil všechny úkoly. Nicméně i tak se projevilo několik nedostatků:

- Snažil se registrovat s nedostatečně dlouhým heslem, jakmile ho aplikace upozornila na chybu, napravil ji.

  Závažnost: 2

  Řešení: Přidat informaci o minimální délce hesla ještě před odesláním formuláře.

- Do pole s datem se snažil napsat datum ve špatném formátu, ale nakonec zjistil jaký je správný formát.

  Závažnost: 1

  Řešení: Přidat placeholder s formátem data.

- Při plnění úkolu 7 si nevšiml, že je žádost již přijata. Místo přijetí jí přiřadil štěně. Domníval se, že je takto žádost přijata.

  Závažnost: 3

  Řešení: Zvýraznit texty stavu rezervace. Vytvořit dokumentaci nebo návod k životnímu cyklu rezervací.

4.2.3.4 Testovací uživatel D

- Hledala vytvoření žádosti o rezervaci štěněte v detailu štěněte, jakmile nenašla odpovídající tlačítko správně vytvořila rezervaci k vrhu

  Závažnost: 1

  Řešení: Návrh na budoucí rozšíření: žádosti o rezervaci konkrétních štěňat.

- Při prvním zadávání data si po zadání data ve špatném vůbec nevšimla, že se do pole propsalo špatné datum a vytvořila psa se špatným datem narození. Při druhém zadání si chyby všimla a po několika neúspěšných pokusech využila pro zadání data přidružený datepicker.

  Závažnost: 3

  Řešení: Přidat placeholder s formátem data

- Při plnění úkolu 7 si nevšimla, že je žádost již přijata. Místo přijetí jí přiřadila štěně. Domnívala se, že je takto žádost přijata

  Závažnost: 3

  Řešení: Zvýraznit texty stavu rezervace. Vytvořit dokumentaci nebo návod k životnímu cyklu rezervací.

- Při vytváření štěněte se jej snažila vytvořit v editaci vrhu, jakmile zjistila že zadaná data nedávají smysl hledala jinde a úspěšně úkol dokončila.

  Závažnost: 2

  Řešení: Vytvořit dokumentaci, nebo návod, který představí veškeré funkcionality aplikace.

- Informaci o veterinární prohlídce zadala do popisu psa, namísto vytvoření životního záznamu. Zřejmě uživatelku nenapadlo, že aplikace umožňuje evidovat životní záznamy.

  Závažnost: 3

  Řešení: Vytvořit dokumentaci nebo návod, který představí veškeré funkcionality aplikace.

- Při zobrazení klávesnice se obrazovka přiblížila a zůstala přiblížená.

  Závažnost: 2

  Řešení: Zakázat automatické přiblížení obrazovky

4.2.4 Vyhodnocení testů

Uživatelské testování odhalilo mnohé nedostatky aplikace. Některé pouze znepříjemnily používání aplikace, jiné byly chyby, které narušovaly funkcionalitu aplikace. Zajímavým poznatkem je, že každý testovací uživatel nedokázal správně použít alespoň jednu funkcionalitu aplikace, avšak tyto nepochopené funkcionality se u každého uživatele lišily. Dle mého názoru jsou některé problémy s plněním úkolů způsobeny převážně z důvodu, že uživatelé viděli aplikaci poprvé a neznali veškeré její funkcionality. To by šlo snadno vyřešit publikováním přehledu funkcionalit aplikace, což navrhuji jako námět na další rozvoj aplikace. Další nedostatky jsem zjistil v nepochopení editací přidružených záznamů na jiných místech, než jsou editační stánky hlavních záznamů (životní záznamy psa v editaci psa, atd.). Zde by mohlo pomocí dříve zmíněné popsání funkcionalit aplikace, ale doporučil bych zvážit úpravu uživatelského rozhraní tak, aby editační formuláře byly více sjednocené.

Sledování reakcí jednotlivých uživatelů mě velmi překvapilo a poskytlo mi zajímavé poznatky. Na vlastní oči jsem se přesvědčil, že každý uživatel k používání aplikace přistupuje odlišně. Velkým překvapením pro mě bylo zjištění, že uživatelé namísto předpřipraveného datepickeru preferovali zadávat datum na klávesnici telefonu. Další překvapivé zjištění bylo to, že prohlížeče Safari automaticky přibližovali stránku při zobrazení klávesnice a tím narušovali responzivní z

Jako často se opakující a dle mého pohledu závažné a nutné k odstranění jsem identifikoval tyto nedostatky:

- Chybějící informace o minimální délce hesla

  Vykonaná oprava: K poli pro heslo jsem přidal informaci o minimální délce.

- Špatný formát datepickeru

  Vykonaná oprava: Do polí pro zadání data jsem přidal placeholder se správným formátem.

- Nepochopení životního cyklu rezervací štěňat

  Vykonaná oprava: Jednotlivé stavy rezervací jsem od sebe graficky odlišil, upravil jsem text notifikací po úspěšné změně stavu rezervace, tlačítka pro jednotlivé změny stavů rezervace jsem od sebe graficky odlišil.

- Nemožnost smazání hodnoty v nepovinném selectboxu

  Vykonaná oprava: Umožnění mazání nepovinných hodnot.

- Prohlížeč Safari při zobrazení klávesnice přibližoval stránku.

  Vykonaná oprava: Zakázal jsem automatické přiblížení stránky.

Kapitola 5

Vyhodnocení a návrh uvedení do provozu

Výstupem této práce je funkční prototyp aplikace, který podporuje propojení poptávky a nabídky štěňat a také usnadňuje evidenci chovatelových psů. Navržená funkcionalita byla otestována tak, že podporuje vykonání všech navržených akcí, dle uživatelského testování byli některé funkcionality poněkud méně intuitivní. Některé tyto problémy jsem opravil, ty jiné by mohlo vyřešit sepsání dokumentace obsahující výpis všech funkcionalit. Pevně věřím, že při opětovném použití by se uživatelé s rozhraním sžili, a po čase by aplikaci dokázali plnohodnotně využívat.

Nicméně se zkušenostmi, které jsem při této práci nabyl, bych aplikaci implementoval jinak. Rozhodně bych zvolil jiné rozložení zdrojového kódu a funkcionalit klientské aplikace. Rovněž bych zvolil jiné komponenty v uživatelském rozhraní, kde bych dbal více na sjednocení jednotlivých komponent, které spolu souvisí a jejich pečlivé provázání.

Stále se jedná o prototyp, který prozatím není vhodný pro spuštění do provozu. K jeho spuštění je potřeba dokončit ještě několik následujících akcí:

1. Testování implementace klientské aplikace

   V sekci testování jsem se zaměřil převážně na serverovou část, uživatelské rozhraní a funkcionalitu aplikace jako celku. Nicméně klientská část aplikace je pouze Hi-fi prototyp, který nebyl testován na správnost implementace. Bylo by tedy vhodné před spuštěním implementovat automatizované testy i pro klientskou aplikaci.

2. Zátěžové testování

   Aplikace byla testována pouze v základních objemech dat. V produkčním prostředí by ale měla zvládat zpracovávat mnohonásobně větší objemy dat, než na jaké byla prozatím testována. Bylo by tedy vhodné vykonat ještě zátěžové testování serverové části aplikace.

3. Doplnit databázi chovných klubů a jejich podmínek pro registraci

   V databázi jsou pouze podmínky pro registraci severských plemen psů. Pro spuštění a plného využití potenciálu této funkcionality je nutné databázi doplnit.

4. Vytvořit souhrn funkcionalit a jejich návod

   Uživatelské testování ukázalo, že uživatelé si často nejsou vědomi toho, že aplikace disponuje některými funkcemi. Bylo by vhodné pro ně vytvořit jejich souhrn a návod, jak je co nejefektivněji používat.

5. Grafické vylepšení uživatelského rozhraní

   V této práci jsem prototyp uživatelského rozhraní řešil pouze z pohledu intuitivnosti a jednoduchosti použití. Jeho vylepšení po designové stránce by zpříjemnilo používání aplikace.

6. SEO optimalizace

   V této práci jsem se problémem SEO optimalizace vůbec nezabýval, nicméně před spuštěním by bylo nutné ji provést, aby byla aplikace co nejlépe dohledatelná.

5.0.1 Uvedení do provozu

Aplikace je momentálně spustitelná v několika dockerových kontejnerech, které obsahují vše, co je potřeba ke spuštění prototypu. Pro spuštění v produkčním prostředí by bylo vhodné pokračovat v tomto principu s tím rozdílem, že už by nebyly potřebné pomocné image. Finální image aplikace by mohl vycházet z toho současného a jednotlivé verze aplikace by byly uchovávány v jednotlivých kontejnerech. To by umožnilo jejich rychlé přepínání v případě výpadků.

Závěr

Cílem této práce bylo navrhnout a realizovat funkční prototyp webové aplikace pro propojení nabídky a poptávky po štěňatech. Funkcionalita měla umožnit evidenci psů, vrhů a jejich prezentaci, evidenci zájemců, rezervaci štěňat a interaktivního průvodce uchovněním psů do vybraných chovatelských klubů. Návrh a implementace prototypu měla být provedena na základě analýzy uživatelských požadavků a rešerše existujících aplikací. Prototyp bylo potřeba podrobit vhodným testům.

Dle mého názoru jsem cíle splnil. Pro přesné vytyčení funkcionality jsem provedl analýzu uživatelských požadavků cílové skupiny uživatelů a rešerši podobných aplikací. Rešerše ukázala, že ještě neexistuje aplikace, která by propojovala podporu chovu psů a zároveň umožňovala prezentovat a nabízet vrhy chovatele, a že je o takovou aplikaci zájem. Na základě provedené rešerše a analýzy jsem navrhl konkrétní funkcionality aplikace. Následně jsem navrhl a implementoval funkční prototyp webové aplikace. Navrhl jsem vhodné otestování správnosti a dle toho aplikaci podrobil testování. Závažné nedostatky jsem odstranil a pro méně závažné jsem navrhl řešení, jak je odstranit. Součástí implementace je i sada automatizovaných testů. Prototyp aplikace je spustitelný pomocí Dockeru. Zdrojové kódy včetně návodu ke spuštění jsou k dispozici na přiložené SD kartě.

Práce popisuje proces vývoje konkrétního prototypu webové aplikace od rešerše a analýzy, přes návrh, implementaci až po závěrečné ověření správnosti. Může sloužit jako inspirace nebo předloha pro vývoj jiných webových aplikací. Pro mě měla tato práce velký přínos v získaných zkušenostech, kdy jsem samostatně vykonal všechny fáze vývoje webové aplikace a vypořádal se s problémy, které nastaly.

Součástí práce je i seznam potřebných akcí pro spuštění do provozu a také návrh spuštění. Další rozvoj této práce souvisí v jejím připravení a následném spuštění. Rovněž existuje spousta dalších funkcionalit pro usnadnění chovu psů, které mohou být implementovány jako pokračování této práce.

Literatura

[1] Podíl domácích mazlíčků v českých domácnostech mírně roste. únor 2018, [cit. 2021-03-03]. Dostupné z: https://www.focus-agency.cz/z-nasich-vyzkumu/podil-domacich-mazlicku-v-ceskych-domacnostech-mirne-roste

[2] Sedgwick, P.: Convenience sampling. *Bmj*, ročník 347, 2013.

[3] *Webfordog.cz*. [cit. 2021-03-03]. Dostupné z: https://www.webfordog.cz

[4] *Puppyfind.com*. [cit. 2021-03-03]. Dostupné z: https://www.puppyfind.com

[5] *Puppyfatapp.com*. [cit. 2021-03-03]. Dostupné z: https://www.puppyfatapp.com

[6] *Google play*. [cit. 2021-03-03]. Dostupné z: https://play.google.com

[7] *Appstore*. [cit. 2021-03-03]. Dostupné z: https://www.apple.com/cz/ios/app-store

[8] *Dogbreederpro.com*. [cit. 2021-03-03]. Dostupné z: https://dogbreederpro.com

[9] Wiegers, K. E.; Beatty, J.: *Software Requirements 3*. USA: Microsoft Press, 2013, ISBN 0735679665.

[10] Samarasinghe, N.; Som´e, S. S.: Generating a Domain Model from a Use Case Model. *SITE*.

[11] Symfony. [cit. 2021-03-18]. Dostupné z: https://symfony.com/what-is-symfony

[12] Symfony - Architecture. [cit. 2021-03-18]. Dostupné z: https://www.tutorialspoint.com/symfony/symfony_architecture.htm

[13] MariaDB. [cit. 2021-03-18]. Dostupné z: https://mariadb.org

[14] Das, L.: The best way to architect your Redux app. *freeCodeCamp [online]*, červenec 2018, [cit. 2021-03-21]. Dostupné z: https://www.freecodecamp.org/news/the-best-way-to-architect-your-redux-app-ad9bd16c8e2d

[15] Fielding, R. T.; Taylor, R. N.: Principled Design of the Modern Web Architecture. *ACM Trans. Internet Technol.*, ročník 2, č. 2, Květen 2002: str. 115–150, ISSN 1533-5399, doi:10.1145/514183.514185. Dostupné z: https://doi.org/10.1145/514183.514185

[16] Úvod do JSON. [cit. 2021-03-21]. Dostupné z: https://www.json.org/json-cz.html

[17] How to draw curved paths. [cit. 2021-11-13]. Dostupné z: https://tex.stackexchange.com/questions/180303/how-to-draw-curved-paths

[18] Nielsen, J.: *Usability Engineering*. Interactive Technologies, Elsevier Science, 1994, ISBN 9780125184069. Dostupné z: https://books.google.cz/books?id=95As2OF67f0C

[19] Doctrine 2. [cit. 2021-12-10]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/2.10/index.html

[20] Working with JPA Entity Objects. [cit. 2021-12-10]. Dostupné z: https://www.objectdb.com/java/jpa/persistence/managed

[21] Shards React. [cit. 2021-12-11]. Dostupné z: https://designrevision.com/downloads/shards-react/

[22] Grofek, T.: ElateMe - QA v multiplatformních aplikacích. 2019.

[23] Olan, M.: Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, ročník 19, č. 2, 2003: s. 319–328.

[24] PHPStan - PHP Static Analysis Tool. [cit. 2021-12-23]. Dostupné z: https://github.com/phpstan/phpstan

[25] PHPUnit Manual. [cit. 2021-12-23]. Dostupné z: https://phpunit.readthedocs.io/en/9.5/

[26] PHP Code sniffer. [cit. 2021-12-23]. Dostupné z: https://github.com/squizlabs/PHP_CodeSniffer

Příloha A

Seznam použitých zkratek

PP Průkaz původu

FR Functional requirement

NFR Nonfunctional requirement

UC Use case

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

REST Representational state transfer

API Application programing interface

ORM Object-relational mapping

URI Uniform resource identifier

URL Uniform resource locator

VMS Virtual managed server

PHP Hypertex preprocesor

SoC Separation of concerns

JSON JavaScript Object Notation

SEO Search engine optimalization

Příloha B

Dotazník

1. Kde jste si vybíral/a nebo plánujete vybrat štěně? (0 - n odpovědí)
   - Internetové inzeráty
   - Sociální sítě
   - Poptání u chovatelů
   - Útulek
   - Jiné (otevřená odpověď)
2. Jste ochotni čekat na štěně dle vašich představ? (Maximálně 1 odpověď)
   - Ano - raději si počkám na vhodný vrh, tak aby štěně odpovídalo mým představám
   - Spíše ano
   - Spíše ne
   - Ne - štěně chci mít co nejdříve, i pokud by nemělo být dle mých představ
3. U výběru štěněte je pro mě důležité (U každé položky volba jedné hodnoty z: „Důležité“, „Spíše důležité“, „Spíše nedůležité“, „Nedůležité“)
   - Průkaz původu
   - Plemeno
   - Pohlaví
   - Pořizovací cena
   - Vzhled
   - Rodiče
   - Od koho štěně pořizuji (chovné podmínky, reputace)
   - Vzdálenost chovatele od mého bydliště

4. Pokud jsou pro Vás důležité ještě jiné parametry, které nejsou v předchozí otázce, prosím, uveďte je. (Otevřená odpověď)
5. Uvítal/a byste aplikaci, která by Vám usnadnila najít si štěně dle Vašich představ? (Maximálně 1 odpověď)
   - Ano
   - Ne
6. Víte jak uchovnit psa? (Maximálně 1 odpověď)
   - Ano
   - Ne
7. Jakou formou nabízíte štěňata? (0 - n odpovědí)
   - Internetové inzeráty
   - Sociální sítě
   - Nemusím nabízet, zájemci mě kontaktují sami
   - Jiné (otevřená odpověď)
8. Je pro Vás důležité, komu štěňata prodáte? (Maximálně 1 odpověď)
   - Ano
   - Ne
9. Pokud jste na předchozí otázku odpověděl/a ano, jaká kritéria u zájemců jsou pro Vás důležitá? (U každé položky volba jedné hodnoty z: „Důležité“, „Spíše důležité“, „Spíše nedůležité“, „Nedůležité“)
   - Zkušenosti s chovem
   - Věk
   - Prostředí, ve kterém by štěně vyrůstalo
   - Osobní setkání před závaznou rezervací
   - Ochota informovat o dospívání pejska
   - Finanční zabezpečení
   - Sympatie
10. Pokud máte ještě jiná kritéria na zájemce o štěňata, která nejsou v předchozí otázce, prosím, uveďte je. (Otevřená odpověď)
11. Jak si evidujete zájemce o štěňata?
    - Na papír

- Do elektronického dokumentu
- Specializovaná aplikace
- Pamatuji si to
- Jiné (otevřená odpověď)

12. Měl/a byste zájem o aplikaci, která by Vám usnadnila prodej štěňat? (Maximálně 1 odpověď)
    - Ano
    - Ne
13. Pokud si přejete být informován/a o průběhu vývoje aplikace, zanechte mi, prosím, kontakt. (Otevřená odpověď)

Příloha C

Detailní popis uživatelských požadavků

- **UC01 - Registrace**

  Popis: Nepřihlášený uživatel vyplní potřebné údaje a registruje se do aplikace.

  Aktéři: Nepřihlášený uživatel.

  Změna stavu: Aplikace bude evidovat nový uživatelský účet.

- **UC02 - Přihlášení**

  Popis: Uživatel přijde na stránku přihlášení, vyplní své přihlašovací údaje a přihlásí se.

  Aktéři: Nepřihlášený uživatel.

  Změna stavu: Nepřihlášený uživatel se změní v přihlášeného uživatele.

- **UC03 - Odhlášení**

  Popis: Uživatel klikne na tlačítko odhlásit se.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Z přihlášeného uživatele se stane nepřihlášený uživatel.

- **UC04 - Změna uživatelských údajů**

  Popis: Uživatel vyplní své nové uživatelské údaje a uloží je.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude u uživatele evidovat nové uživatelské údaje.

- **UC05 - Změna jazyka aplikace**

Popis: Uživatel klikne na příslušné tlačítko reprezentující chtěnou jazykovou mutaci.

Aktéři: Přihlášený uživatel, nepřihlášený uživatel.

Změna stavu: Aplikace se bude zobrazovat v jiné jazykové mutaci.

- **UC06 - Vytvoření psa**

  Popis: Uživatel klikne na tlačítko vytvořit nového psa. Aplikace mu zobrazí formulář, kde uživatel vyplní údaje o psovi a uloží je.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude evidovat nového psa.

- **UC07 - Editace psa**

  Popis: Uživatel si zvolí psa, kterého chce editovat kliknutím na příslušné tlačítko. Aplikace mu zobrazí formulář, kde uživatel vyplní nové údaje o psovi a uloží je.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude u daného psa evidovat nové údaje.

- **UC08 - Odstranění psa**

  Popis: Uživatel si zvolí psa, kterého již nechce evidovat a klikne na tlačítko odstranit. Aplikace jej vyzve k potvrzení a po úspěšném potvrzení přestane psa zobrazovat.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace nebude evidovat psa.

- **UC09 - Zobrazení psa**

  Popis: Uživatel si zvolí psa a klikne na příslušný odkaz. Aplikace mu zobrazí detailní výpis údajů, které o daném psovi eviduje.

  Aktéři: Přihlášený uživatel.

- **UC10 - Převedení vlastnictví psa**

  Popis: Přihlášený uživatel si zvolí psa přiřazeného k rezervaci a kterého si již majitel odvedl. Klikne na příslušné tlačítko,systém mu zobrazí formulář pro převedení vlastnictví psa, uživatel jej vyplní, potvrdí a odešle.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude evidovat psa pod jiným majitelem.

- **UC11 - Vytvoření životního záznamu**

  Popis: Uživatel si zvolí psa, kterého chce editovat a klikne na tlačítko přidat životní záznam. Aplikace zobrazí formulář pro přidání nového životního záznamu, uživatel jej vyplní a odešle.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude u psa evidovat nový životní záznam.

- **UC12 - Zobrazení životních záznamů**

  Popis: Uživatel si zvolí psa, u kterého chce zobrazit životní záznamy a klikne na příslušné tlačítko. Aplikace mu zobrazí veškeré životní záznamy, které eviduje.

  Aktéři: Přihlášený uživatel.

- **UC13 - Odebrání životního záznamu**

  Popis: Uživatel si ve výpisu životních záznamů zvolí životní záznam u psa, který již nechce evidovat a klikne na příslušné tlačítko. Aplikace jej vyzve k potvrzení a po úspěšném potvrzení přestane záznam zobrazovat.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace nebude dále evidovat životní záznam.

- **UC14 - Editace životního záznamu**

  Popis: Uživatel si ve výpisu životních záznamů daného psa zvolí životní záznam, který chce editovat a klikne na příslušné tlačítko. Aplikace zobrazí příslušný formulář, uživatel jej vyplní a odešle.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude u životního záznamu evidovat nová data

- **UC15 - Vytvoření vrhu**

  Popis: Uživatel klikne na tlačítko vytvořit nový vrh. Aplikace mu zobrazí příslušný formulář, uživatel vyplní veškeré údaje o vrhu a odešle jej.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude evidovat nový vrh.

- **UC16 - Editace vrhu**

Popis: Uživatel si ve výpisu vrhů zvolí vrh, který chce evidovat a klikne na příslušné tlačítko. Aplikace zobrazí příslušný formulář, uživatel vyplní nová data vrhu a odešle jej.

Aktéři: Přihlášený uživatel.

Změna stavu: Aplikace bude u vrhu evidovat nová data.

- **UC17 - Odstranění vrhu**

  Popis: Uživatel si ve výpisu vrhů zvolí vrh, který již nechce evidovat a klikne na příslušné tlačítko. Aplikace jej vyzve k potvrzení a po úspěšném potvrzení přestane vrh zobrazovat.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace nebude dále vrh evidovat.

- **UC18 - Zobrazení vrhu**

  Popis: Uživatel si zvolí vrh, který jej zajímá a klikne na příslušné tlačítko. Nepřihlášení i přihlášení uživatelé si mohou prohlédnout zveřejněné vrhy. Přihlášený uživatel si navíc může prohlédnout údaje o neveřejném vrhu, který vlastní nebo ke kterému má přiřazenou rezervaci.

  Aktéři: Přihlášený uživatel, nepřihlášený uživatel.

- **UC19 - Žádost o rezervaci**

  Popis: Uživatel si zvolí vrh, ve kterém má zájem o štěně a klikne na příslušné tlačítko. Systém mu zobrazí formulář pro prokázání splnění podmínek rezervace, uživatel jej vyplní a odešle.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude evidovat novou žádost o rezervaci psa.

- **UC20 - Potvrzení rezervace**

  Popis: Majitel psa si zobrazí žádost o rezervaci, zkontroluje data splnění podmínek rezervace a klikne na příslušné tlačítko.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace převede žádost o rezervaci na rezervaci.

- **UC21 - Definice podmínek rezervace**

  Popis: Uživatel zvolí vrh, pro který chce podmínky definovat a klikne na příslušné tlačítko. Systém zobrazí příslušný formulář, uživatel jej vyplní a odešle jej.

Aktéři: Přihlášený uživatel.

Změna stavu: Aplikace bude evidovat podmínky pro rezervaci.

- **UC22 - Stornování rezervace**

  Popis: Tenhle UC může provést majitel vrhu i zájemce o rezervaci. Majitel vrhu se rozhodne nepřijmout rezervaci a klikne na příslušné tlačítko. Zájemce o rezervaci se rozhodne, že již nemá zájem a klikne na příslušné tlačítko. Aplikace zobrazí rezervaci jako stornovanou.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude rezervaci evidovat jako stornovanou.

- **UC23 - Zobrazení rezervace**

  Popis: Uživatel si zvolí rezervaci a klikne na příslušné tlačítko. Aplikace zobrazí veškeré údaje o rezervaci.

  Aktéři: Přihlášený uživatel.

- **UC24 - Přiřazení psa k rezervaci**

  Popis: Majitel vrhu k rezervaci přiřadí psa. Po provedení UC bude systém evidovat přiřazeného psa k rezervaci. Vlastník rezervace si bude moci zobrazit údaje o psovi.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude evidovat přiřazení psa k rezervaci.

- **UC25 - Vyhledání vrhu**

  Popis: Uživatel zadá kritéria, dle kterých chce vyhledávat. Aplikace mu zobrazí všechny vrhy, které splňují daná kritéria.

  Aktéři: Nepřihlášený uživatel, přihlášený uživatel.

- **UC26 - Zobrazení chovatelských klubů**

  Popis: Uživatel si zobrazí chovatelské kluby, které umožňují registraci zvoleného plemene.

  Aktéři: Přihlášený uživatel.

- **UC27 - Zobrazení podmínek registrace**

  Popis: Uživatel pří prohlížení chovatelských klubů klikne na příslušné tlačítko. Aplikace zobrazí seznam podmínek pro rezervaci do zvoleného chovatelského klubu.

Aktéři: Přihlášený uživatel.

- **UC28 - Volba chovatelského klubu pro registraci**

  Popis: Uživatel si zvolí chovatelský klub, do kterého chce psa registrovat.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude u psa evidovat chovatelský klub pro registraci.

- **UC29 - Označení podmínky registrace do chovatelského klubu za splněnou**

  Popis: Uživatel označí podmínku registrace do chovatelského klubu za splněnou.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude evidovat podmínku pro psa jako splněnou.

- **UC30 - Označení podmínky registrace do chovatelského klubu za nesplněnou**

  Popis: Uživatel označí podmínku registrace do chovatelského klubu za nesplněnou.

  Aktéři: Přihlášený uživatel.

  Změna stavu: Aplikace bude evidovat podmínku pro psa jako nesplněnou.

- **UC31 - Zobrazení splněných a nesplněných podmínek pro registraci**

  Popis: Uživatel si pro zvoleného psa zobrazí splněné a nesplněné podmínky pro registraci do chovatelského klubu.

  Aktéři: Přihlášený uživatel.

Příloha D

Seznam úkolů aplikace

D.1 Uživatelské účty

T01 Zadat přihlašovací údaje

Typ: Akce; **Důležitost**: 5; **Frekvence**: 2; **D x F**: 10

T02 Přihlásit se

Typ: Akce; **Důležitost**: 5; **Frekvence**: 2; **D x F**: 10

T03 Zobrazit informaci, zda je uživatel přihlášený

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 3; **D x F**: 12

T04 Odhlásit se

Typ: Akce; **Důležitost**: 5; **Frekvence**: 2; **D x F**: 10

T05 Zadat uživatelské údaje pro registraci

Typ: Akce; **Důležitost**: 5; **Frekvence**: 1; **D x F**: 5

T06 Registrovat se

Typ: Akce; **Důležitost**: 5; **Frekvence**: 1; **D x F**: 5

T07 Potvrdit registraci

Typ: Akce; **Důležitost**: 5; **Frekvence**: 1; **D x F**: 5

T08 Zobrazit své uživatelské údaje

Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 1; **D x F**: 3

T09 Zadat nové uživatelské údaje

Typ: Akce; **Důležitost**: 2; **Frekvence**: 1; **D x F**: 2

T10 Změnit uživatelské údaje
Typ: Akce; **Důležitost**: 2; **Frekvence**: 1; **D x F**: 2

T11 Zadat nové heslo
Typ: Akce; **Důležitost**: 3; **Frekvence**: 1; **D x F**: 3

T12 Změnit heslo
Typ: Akce; **Důležitost**: 3; **Frekvence**: 1; **D x F**: 3

T13 Zvolit účet pro resetování hesla
Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T14 Iniciovat resetování hesla
Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T15 Resetovat heslo
Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

D.2 Lokalizace

T16 Zobrazit aktuální jazykovou mutaci aplikace
Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**:

T17 Zobrazit dostupné jazykové mutace aplikace
Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T18 Změnit aktuální jazykovou mutaci aplikace
Typ: Akce; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

D.3 Základní úkoly aplikace

T19 Zobrazit výsledek provedené operace
Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T20 Informovat o chybě
Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T21 Zobrazit dvoufázové potvrzení kritických operací
Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T22 Potvrdit dvoufázové potvrzení kritických operací
Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T23 Zamítnout dvoufázové potvrzení kritických operací
Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

D.4 Psi

T24 Zobrazit své psy

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 3; **D x F**: 12

T25 Zadat údaje psa

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T26 Vytvořit nového psa

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T27 Upravit údaje psa

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T28 Odstranit psa

Typ: Akce; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T29 Zobrazit údaje psa

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 3; **D x F**: 12

T30 Zadat nového majitele psa

Typ: Akce; **Důležitost**: 3 ; **Frekvence**: 1; **D x F**: 3

T31 Převést vlastnictví psa

Typ: Akce; **Důležitost**: 3; **Frekvence**: 1; **D x F**: 3

T32 Zobrazit životní záznamy psa

Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T33 Zadat údaje životního záznamu psa

Typ: Akce; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T34 Vytvořit životní záznam psa

Typ: Akce; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T35 Upravit životní záznam psa

Typ: Akce; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T36 Odstranit životní záznam psa

Typ: Akce; **Důležitost**: 3; **Frekvence**: 1; **D x F**: 3

D.5 Vrhy

T37 Zobrazit vrhy

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 3; **D x F**: 12

T38 Zadat údaje vrhu

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T39 Vytvořit nový vrh

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T40 Zobrazit podmínky rezervace vrhu

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T41 Zadat údaje podmínky rezervace vrhu

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T42 Přidat podmínku rezervace vrhu

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T43 Upravit podmínku rezervace vrhu

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T44 Odstranit podmínku rezervace vrhu

Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T45 Upravit vrh

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T46 Přiřadit štěně k vrhu

Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T47 Odstranit vrh

Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T48 Zobrazit údaje vrhu

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 3; **D x F**: 12

T49 Zadat kritéria vyhledávání vrhů

Typ: Akce; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T50 Zobrazit zadané kritéria vyhledávání vrhů

Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T51 Zobrazit dostupné hodnoty kritérií vyhledávání vrhů

Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T52 Vyhledat dostupné vrhy

Typ: Akce; **Důležitost**: 4; **Frekvence**: 3; **D x F**: 12

D.6 Rezervace

T53 Zobrazit odeslané rezervace

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T54 Zobrazit obdržené rezervace

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T55 Zobrazit detail rezervace

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T56 Vytvořit rezervaci

Typ: Akce; **Důležitost**: 4; **Frekvence**: 2; **D x F**: 8

T57 Stornovat rezervaci

Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T58 Přijmout rezervaci

Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T59 Zadat údaje splnění podmínky rezervace

Typ: Akce; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T60 Zobrazit dostupné psy pro rezervaci

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

T61 Přiřadit psa rezervaci

Typ: Zobrazení; **Důležitost**: 4; **Frekvence**: 1; **D x F**: 4

D.7 Chovatelské kluby

T62 Zobrazit dostupné chovatelské kluby

Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T63 Zobrazit podmínky registrace chovatelského klubu

Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T64 Přihlásit psa k registraci do chovatelského klubu

Typ: Akce; **Důležitost**: 3; **Frekvence**: 1; **D x F**: 3

T65 Zobrazit stav podmínek registrace chovatelského klubu pro konkrétního psa

Typ: Zobrazení; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 6

T66 Označit podmínku registrace za splněnou

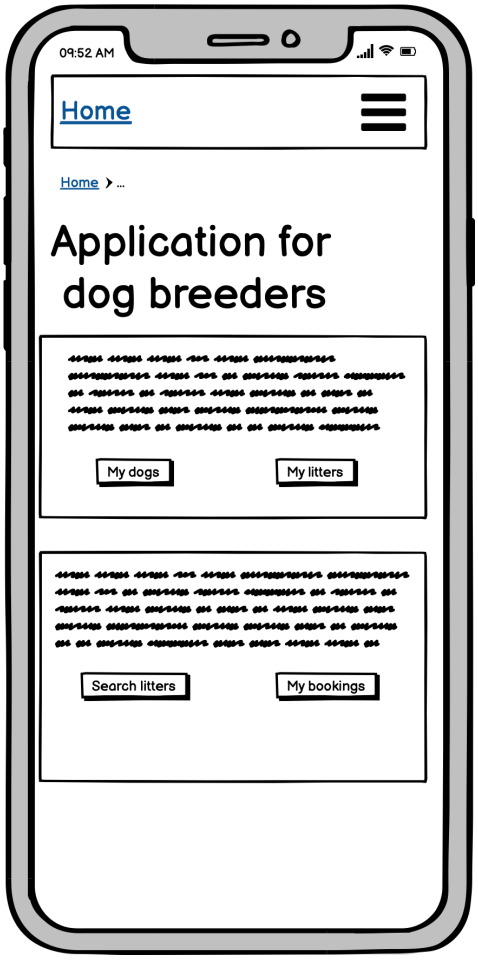
Typ: Akce; **Důležitost**: 3; **Frekvence**: 1; **D x F**: 3

T67 Označit podmínku registrace za nesplněnou

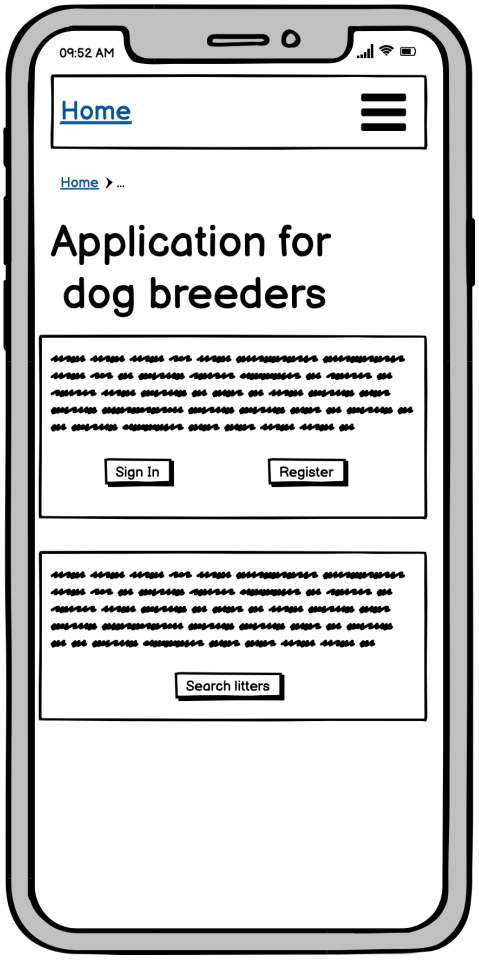
Typ: Akce; **Důležitost**: 3; **Frekvence**: 2; **D x F**: 3

Příloha E

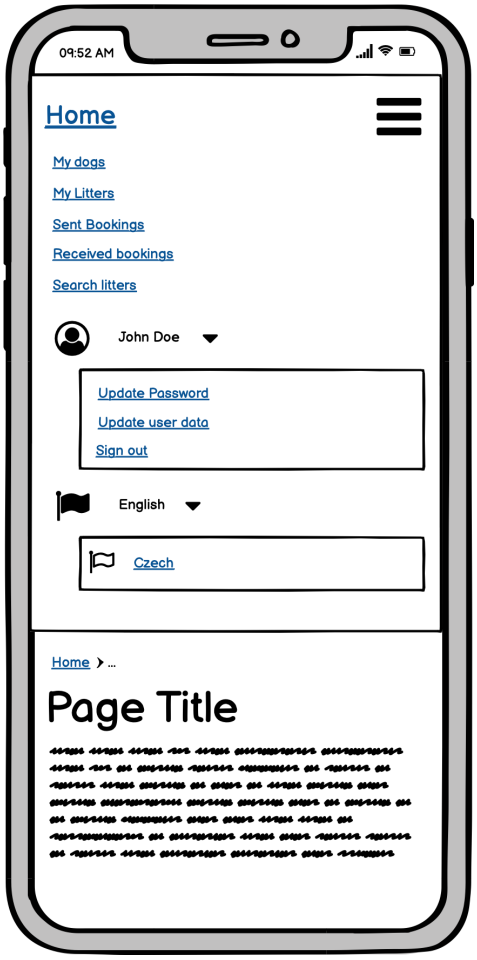
Wireframes aplikace



Obrázek E.1: Wireframe - Homepage přihlášeného uživatele

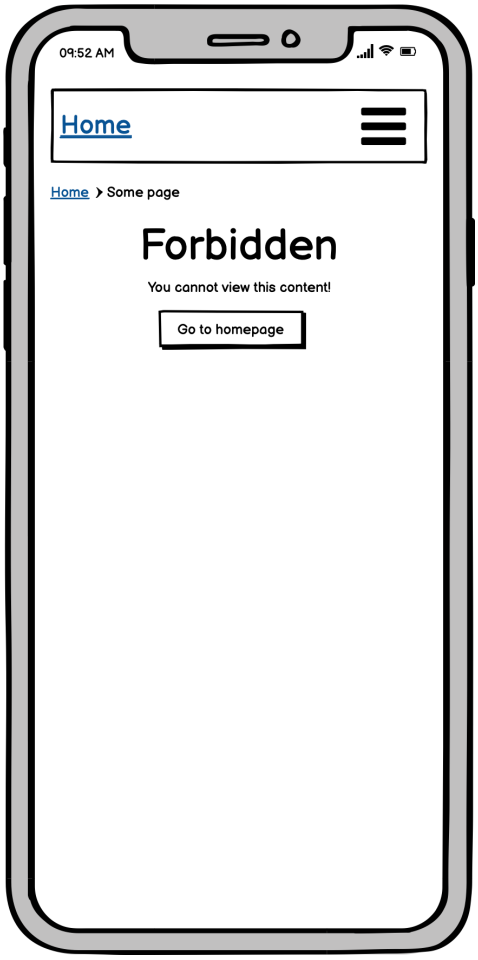


Obrázek E.2: Wireframe - Homepage nepřihlášeného uživatele

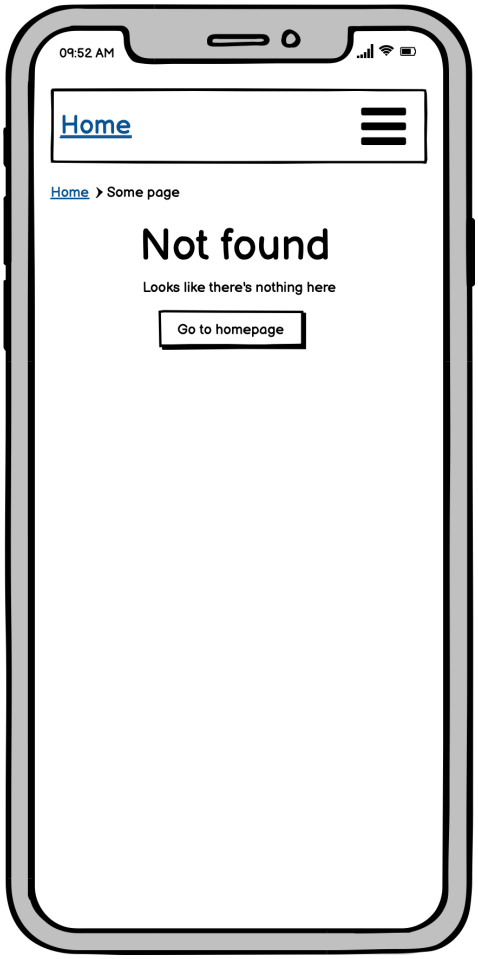


Obrázek E.3: Wireframe - Menu přihlášeného uživatele

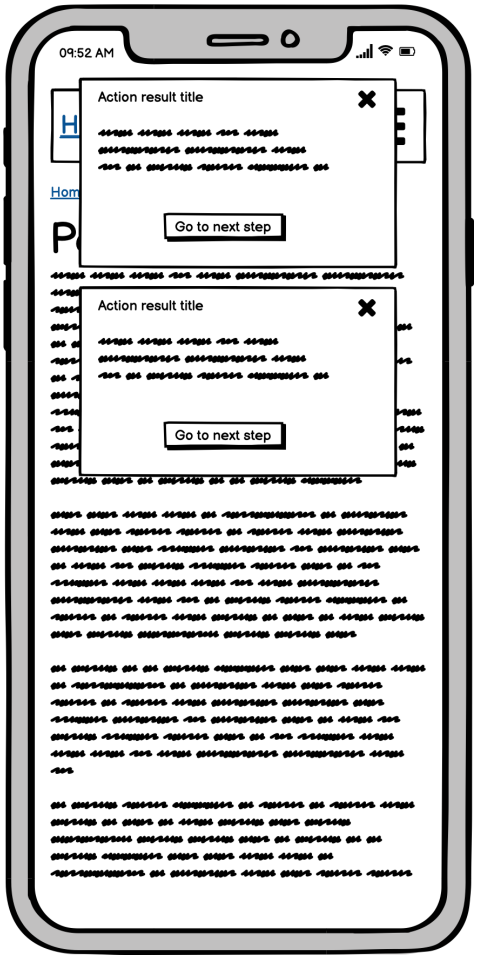
Obrázek E.4: Wireframe - Menu nepřihlášeného uživatele



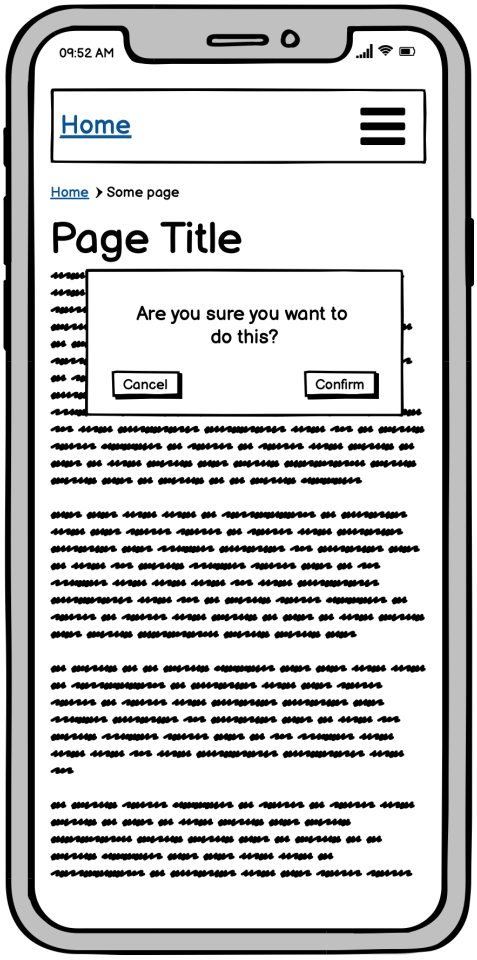
Obrázek E.5: Wireframe - Stránka, pro kterou nemá uživatel oprávnění



Obrázek E.6: Wireframe - Nenalezená stránka

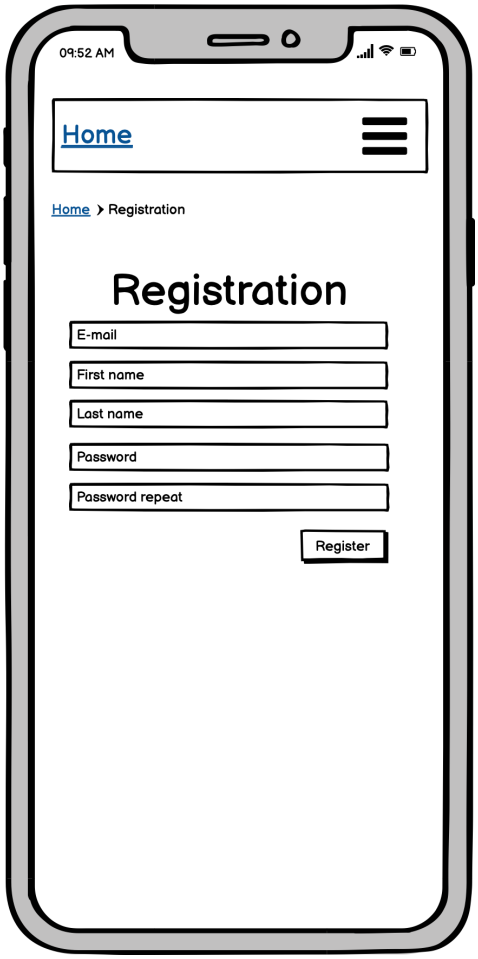


Obrázek E.7: Wireframe - Oznámení výsledku operace

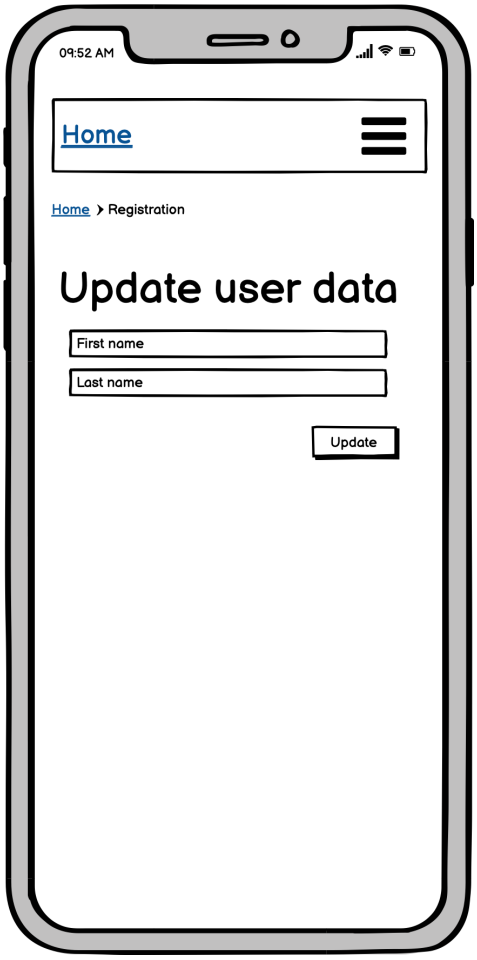


Obrázek E.8: Wireframe - Dvoufázové potvrzení akce

Obrázek E.9: Wireframe - Přihlášení

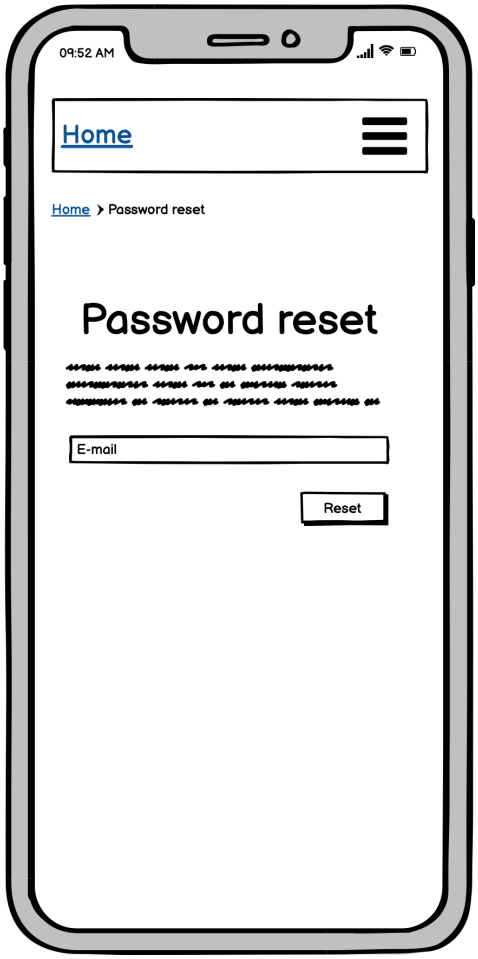


Obrázek E.10: Wireframe - Registrace



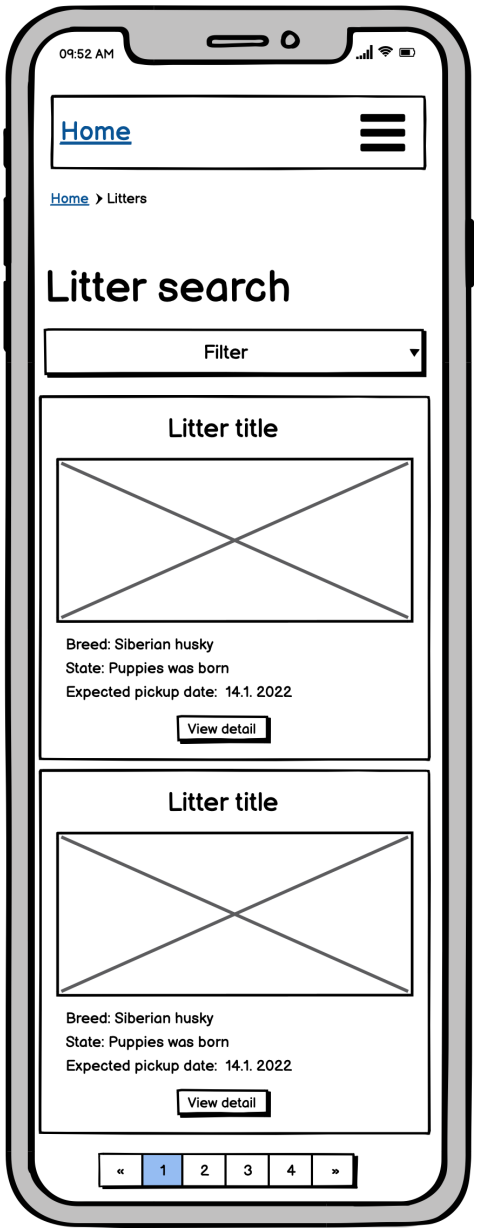
Obrázek E.11: Wireframe - Změna uživatelských dat

Obrázek E.12: Wireframe - Změna hesla

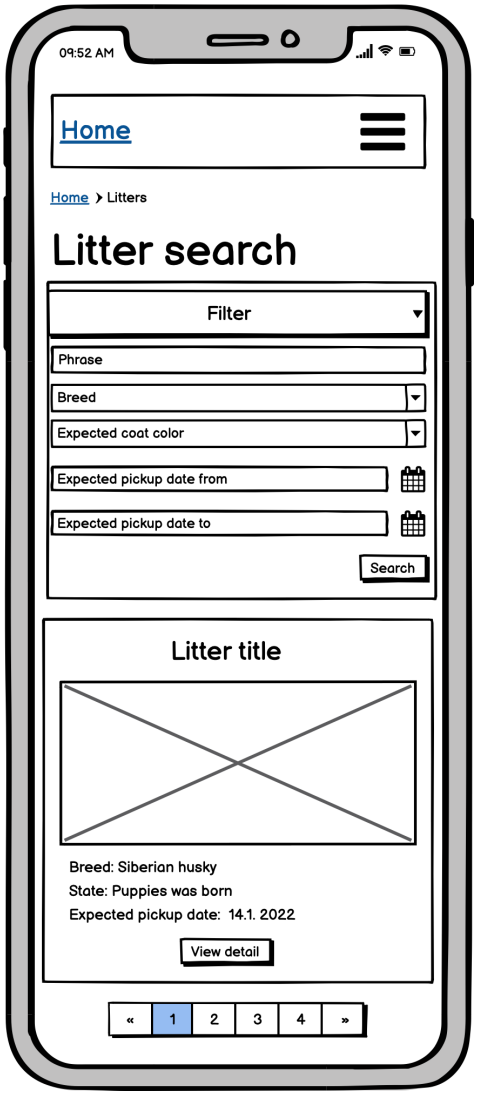


Obrázek E.13: Wireframe - Resetování hesla

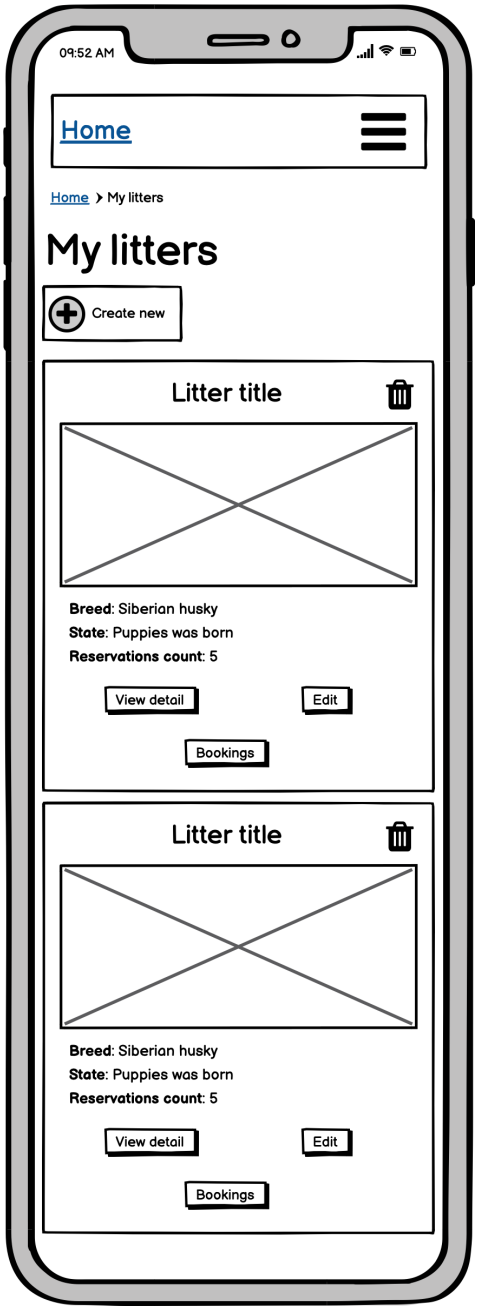
Obrázek E.14: Wireframe - Dokončení resetování hesla



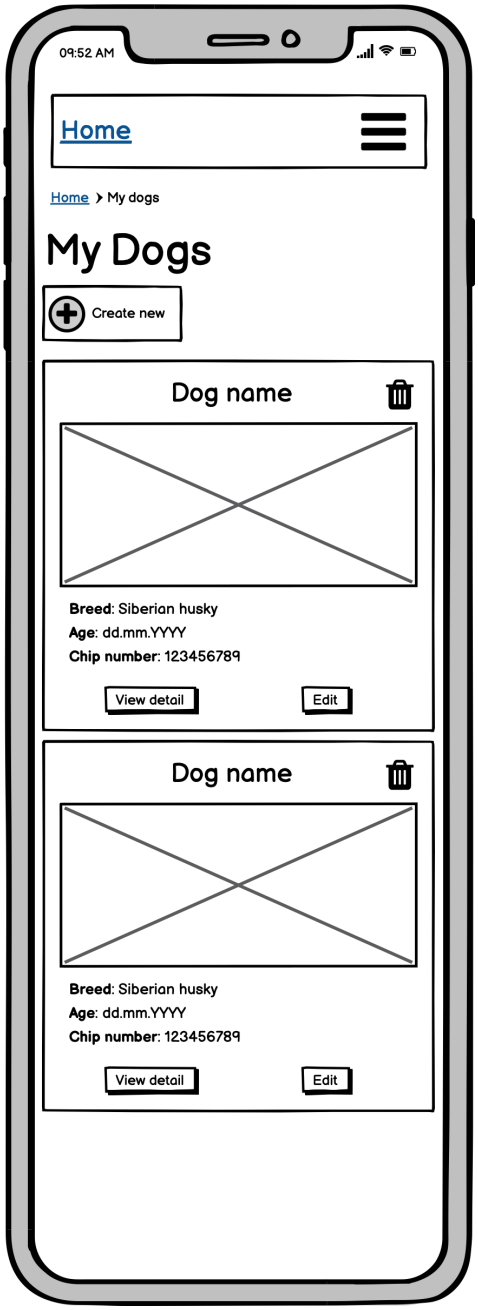
Obrázek E.15: Wireframe - Výsledky vyhledávání vrhů



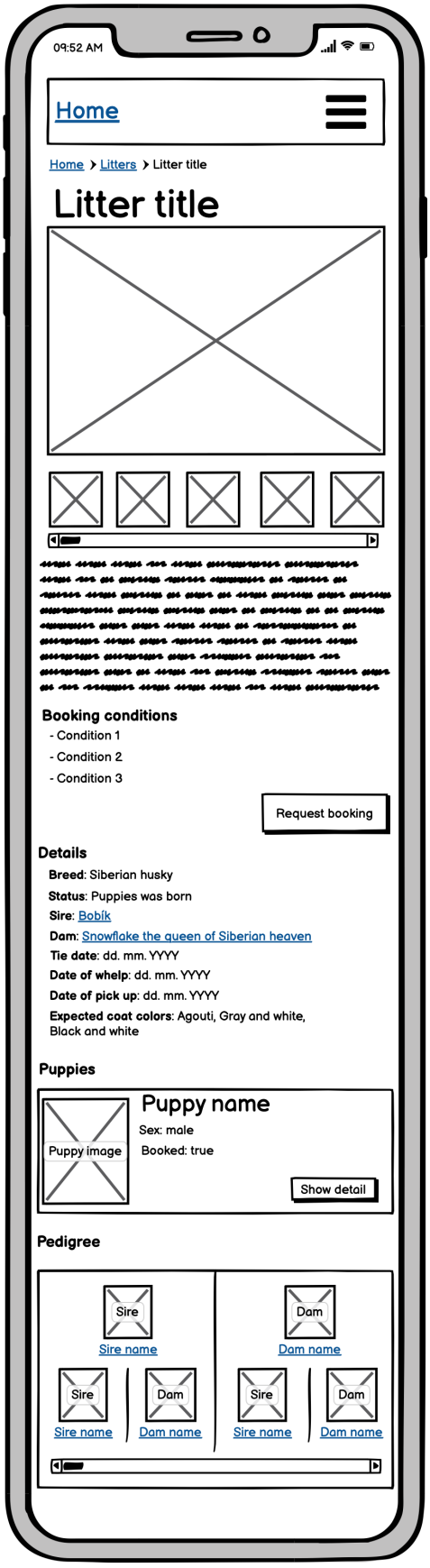
Obrázek E.16: Wireframe - Filtrování vyhledávání vrhů



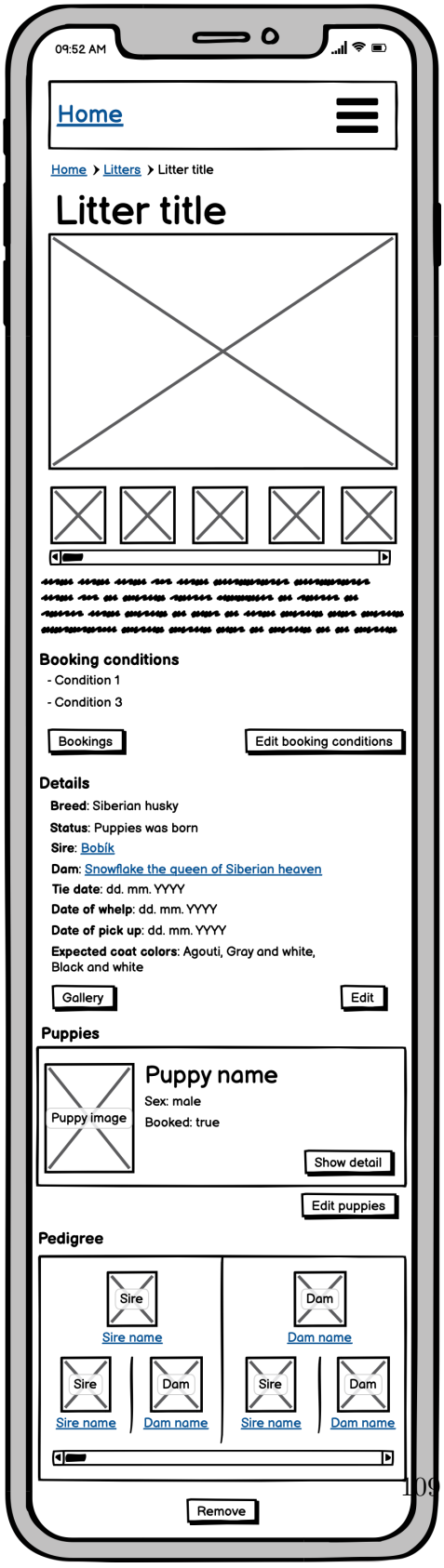
Obrázek E.17: Wireframe - Seznam vrhů uživatele



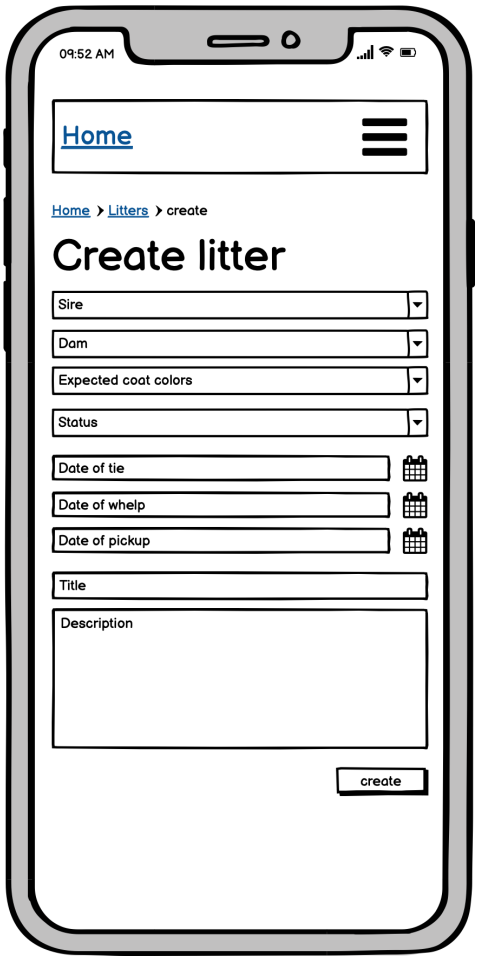
Obrázek E.18: Wireframe - Seznam psů uživatele



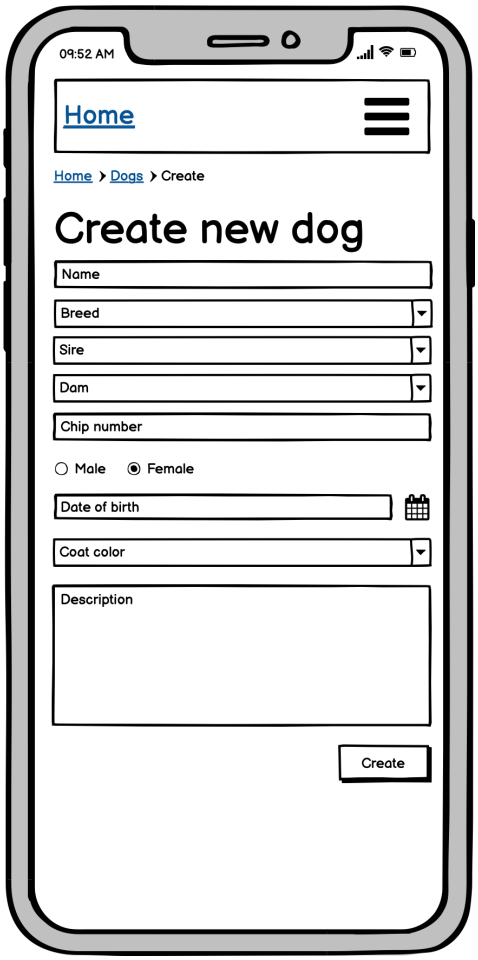
Obrázek E.19: Wireframe - Detail vrhu z pohledu zájemce



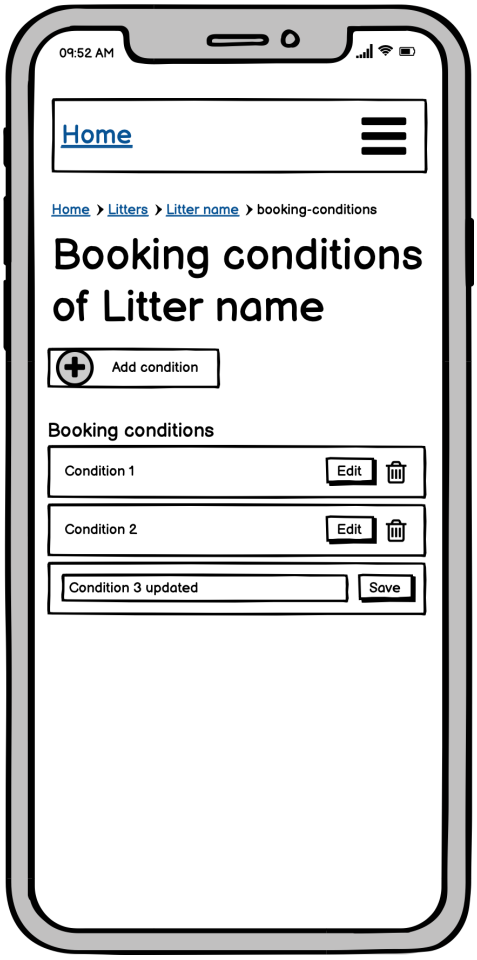
Obrázek E.20: Wireframe - Detail vrhu z pohledu majitele



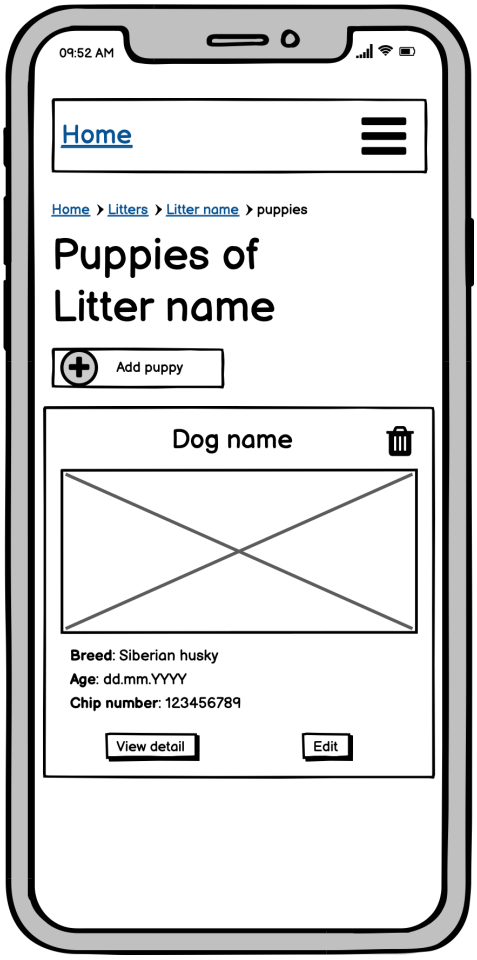
Obrázek E.21: Wireframe - Vytvoření / úprava vrhu



Obrázek E.22: Wireframe - Vytvoření / úprava psa



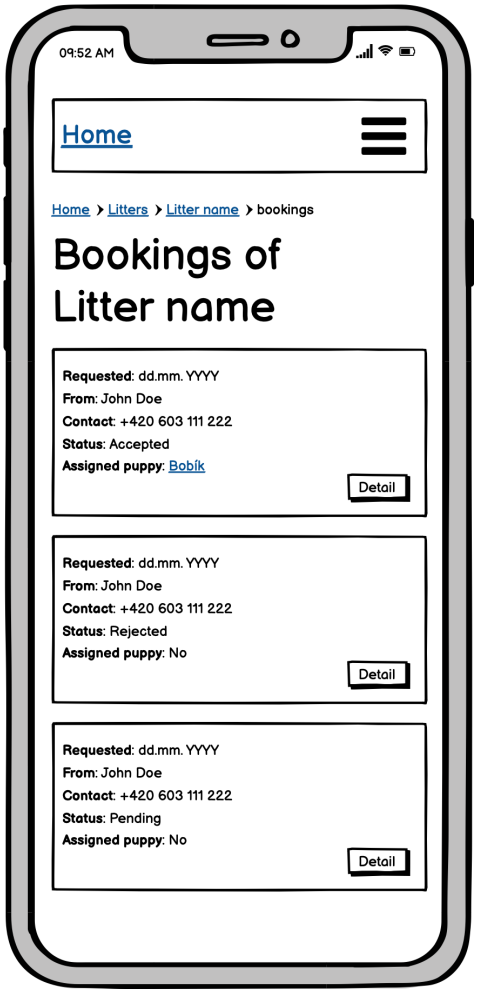
Obrázek E.23: Wireframe - Úprava podmínek registrace vrhu



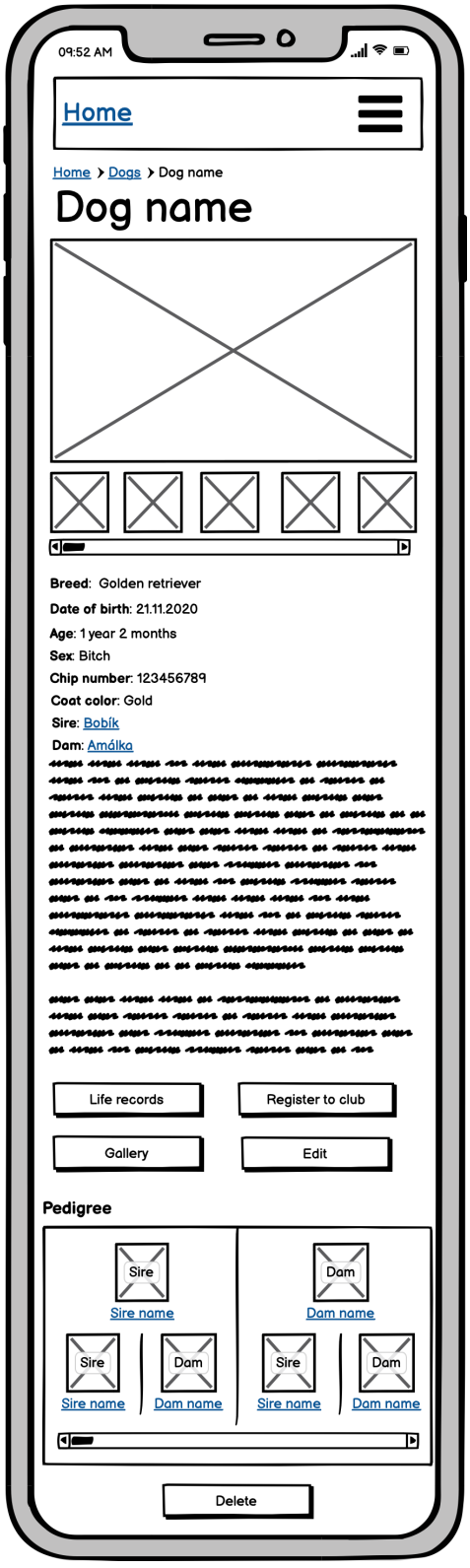
Obrázek E.24: Wireframe - Úprava štěňat vrhu

Obrázek E.25: Wireframe - Žádost o rezervaci

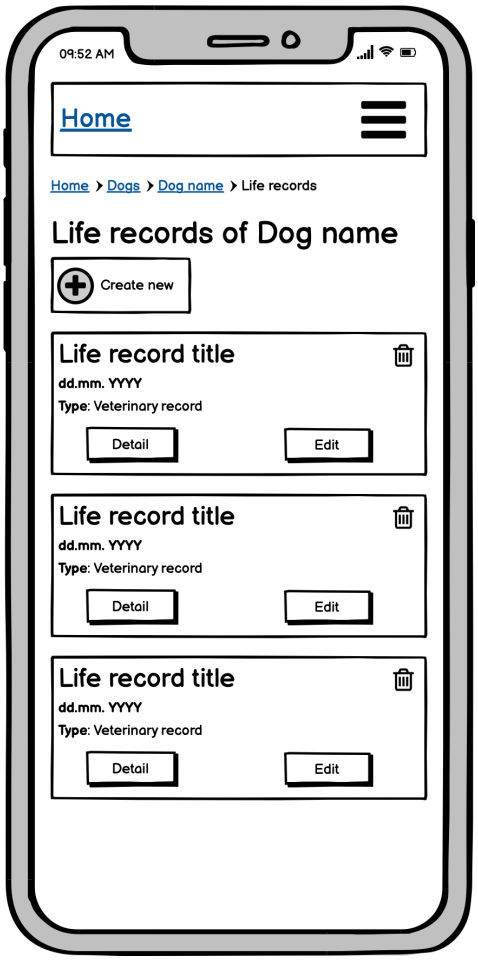
Obrázek E.26: Wireframe - Úprava žádosti o rezervaci



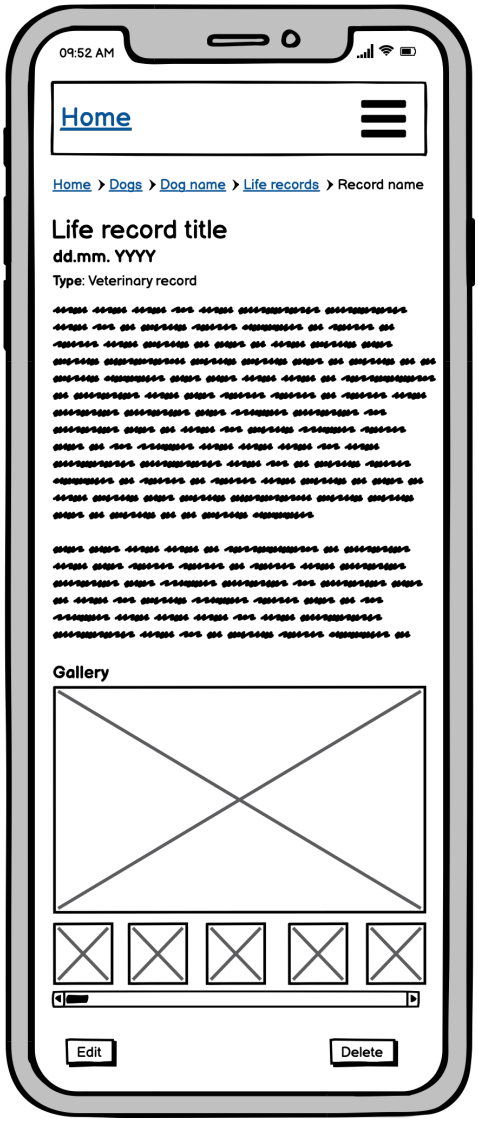
Obrázek E.27: Wireframe - Seznam žádostí o rezervaci



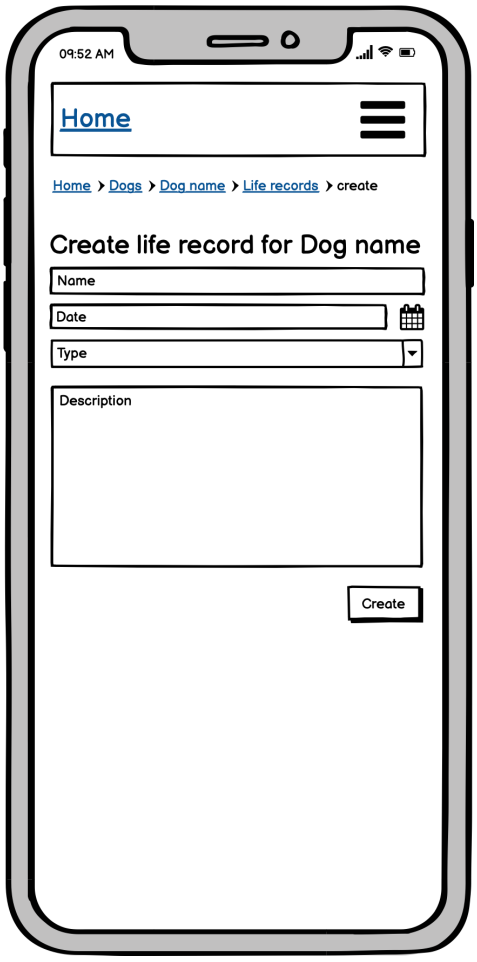
Obrázek E.28: Wireframe - Detail psa



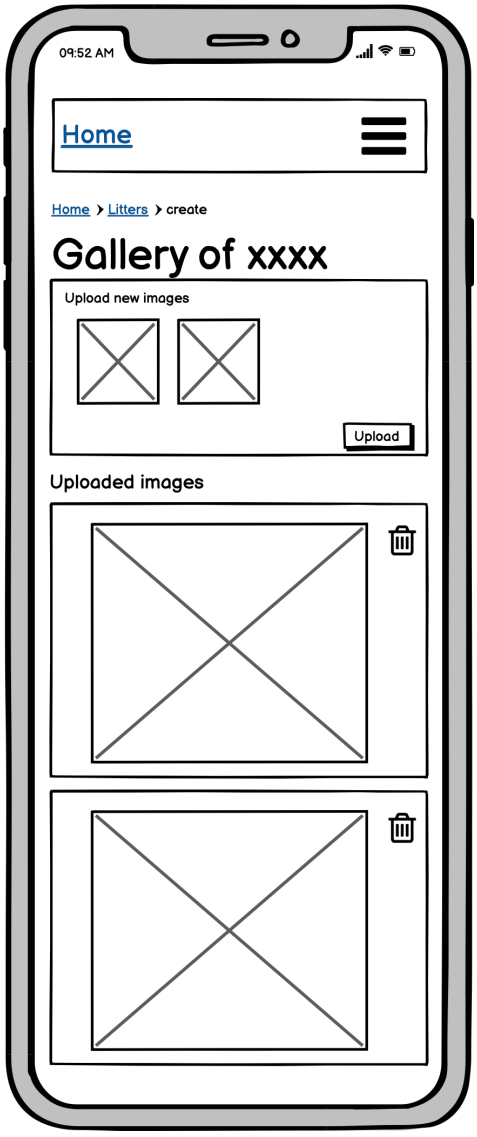
Obrázek E.29: Wireframe - Seznam životních záznamů psa



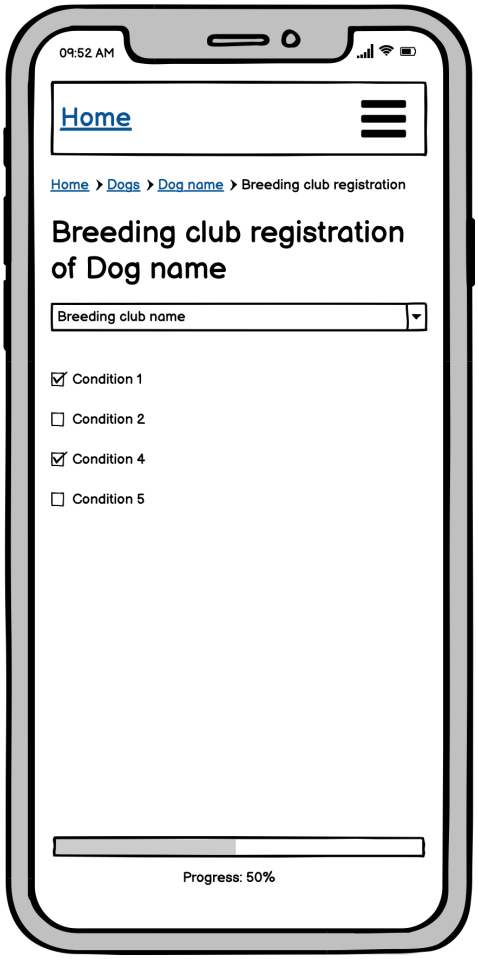
Obrázek E.30: Wireframe - Detail životního záznamu psa



Obrázek E.31: Wireframe - Vytvoření / úprava životního záznamu psa



Obrázek E.32: Wireframe - Správa galerie



Obrázek E.33: Wireframe - Správa podmínek pro registraci do chovatelského klubu

Příloha F

Obsah přiložené SD karty

- `src`
  - `app` ..... zdrojové kódy aplikace včetně návodu na spuštění
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `tests` ..... záznamy uživatelského testování