TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mihkel Heinmaa 193939IADB

Veebi- ning mobiilirakendus kadunud ja leitud loomade info haldamiseks

Bakalaureusetöö

Juhendaja: Meelis Antoi
MSc

Tallinn 2022

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Mihkel Heinmaa

24.04.2022

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on luua töötav veebi- ja mobiilirakenduse prototüüp, millel oleksid olemas kõik primaarsed funktsioonid selle eesmärgipäraseks kasutamiseks. Lisaks kahele klientrakendusele luuakse ka neid teenindav tagarakendus.

Süsteemi eesmärk on võimaldada kasutajatel raporteerida oma kadunud lemmikloomi ning ka märkida üles leitud loomi. Raportid kuvatakse kaardil ning neid on võimalik kasutajal filtreerida. Samuti on lisatud lihtne sõnumirakendus, ning võimalus filtreerida välja raportid, mida süsteem peab valitud raportiga sarnaseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 6 peatükki, 13 joonist, 0 tabelit.

Abstract

Web and Mobile Application for Lost and Found Pets Information Management

The aim of this thesis is to create a system where people can report their lost pets or pets they have found on the streets. This system will consist of three parts: backend, web client and mobile client. System must have the following features: user can post a new report and edit it if needed; reports will be displayed on a map and user can do basic filtering; system will suggest connected reports to a specific report; system will provide a basic messaging functionality for the users.

First part details the underlaying problem and briefly portrays existing solutions to the problem.

Second part of the thesis focuses on choosing the frameworks to achieve the goals stated above. Author will provide pros and cons for every framework for each part of the system and describes what aspects were considered when the choice was made.

Third part describes the realization of the project starting from the database and ending with the mobile application. Here, key features will be outlined and briefly described, along with the development process.

The thesis is in Estonian and contains 27 pages of text, 6 chapters, 13 figures, 0 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
CSS	*Cascading Style Sheets*, küljendamisel kasutatav märgistuskeel
DOM	*Document Object Model*, dokumendi objektmudel
DTO	*Data Transfer Object*, andmeedastusobjekt
HTML	*HyperText Markup Language*, veebilehtede märgistuskeel
HTTP	*HyperText Markup Language*, teabeedastus protokoll
JS	*JavaScript*, programmeerimiskeel JavaScript
JSON	*JavaScript Object Notation*, andmevahetusvorming
JSX	JavaScripti süntaksi laiendus
LINQ	*Language Intergated Query*, C#-i kasutatav päringusüntaks
MTÜ	Mittetulundusühing
NPM	Üks paketihalduritest (akronüüm ei oma tänapäeval tähendust)
ORM	*Object–Relational Mapping*, objekt-relatsioonvastendus
PWA	*Progressive Web Application*, pseudoäpp
SDK	*Software Development Kit*, tarkvaraarenduse komplekt
SQL	*Structured Query Language*, struktuurpäringukeel
UI	*User Interface*, kasutajaliides
URL	*Uniform Resource Locator*, siin: võrguaadress

Sisukord

Jooniste loetelu

1 Sissejuhatus

Igal aastal satub ainuüksi Tallinna kodutute loomade varjupaika tuhande ringis lemmikloomi [1]. Kindlasti on veel suurem hulk neid loomi, kes ei satu varjupaikadesse või muude MTÜ-de hoole alla. Samas teab pea iga loomaomanik, kui tähtis on tema lemmikloom, seega on mõistetav teha kadunud lemmiku leidmine võimalikult lihtsaks. Niisamuti on ka omavalitsustel finantsiline huvi vähendada asukate hulka varjupaikades – näiteks Tallinna linna vastav eelarverida on 400 000 eurot (hõlmab ka muid loomakaitse tegevusi) [2].

Käesoleva töö eesmärk on luua tagarakendus ning kaks klientrakendust (veebi ja mobiili), milledel on prototüübina olemas kõik funktsioonid et oleks võimalikult hõlbus kadunud ja leitud loomad oma omanikega taas kokku viia. Lähtetingimusi ei ole – rakendus on iseseisev, enda andmebaasiga ning prototüübi tasemel ei sõltu ka muudest rakendustest välja arvatud Google' teenused.

Töös kirjeldatakse esmalt probleemi, seejärel asutakse analüüsima võimalike vahendeid, kuidas probleemi lahendada – tagarakenduse valik, veebirakenduse tarbeks JavaScripti raamistiku valik ning lõpuks levinumate hübriidmobiilirakenduste tehnoloogiate võrdlemine. Sellele järgneb realisatsiooni kirjeldus ning kokkuvõte.

2 Probleemi kirjeldus

Sissejuhatuses sai tõdetud, et loomi satub varjupaika suures hulgas. Kuigi koerte kiibistamine on juba aastast 2014 [3] Eestis kohutuslik, ei ole see ikkagi kuigi levinud ning kasside puhul on see sootuks vabatahtlik (kui omavalitsus pole otsustanud teisiti, näiteks Tallinnas on ka kasside kiibistamine kohustuslik [4]). Seepärast ei saa väga loota, et tänavalt leitud kassi võib kiibi alusel omanikule tagastada. Seda enam, et enamik inimesi ei haara kohe leitud looma sülle ning ei asu otsima aparaati ning selle operaatorit, kes kiibilt andmed sisse loeks. Selle asemel parimal juhul tehakse loomast foto ning postitatakse see sotsiaalmeediasse. Samuti ei ole Eestis registri pidaja konkreetselt määratletud – enamik omavalitsusi on liitunud küll suurima registriga (www.llr.ee), kuid leidub ka väiksemaid ning ka sootuks paberkandjal registreid [5].

Seega on 2 olulist probleemi, mis takistavad omaniku ja lemmiku taasühinemist: suhteliselt vähene kiibistatus ning tavainimese võimetus kiipe lugeda. Siit jällegi on lihtne lahendus vana hea kuulutuste süsteemi näol, mis küll on kaasaegses maailmas saanud veidi uue näo.

2.1 Olemasolevad lahendused

Järgnevalt vaadeldaksegi põgusalt olemasolevaid lahendusi grupeerides need laias laastus kaheks: ajalehe stiilis kuulutused ning kaardile visualiseeritud kuulutused.

2.1.1 Ajalehe stiilis kuulutused

Lihtsaim viis oma lemmiku kadumisest või tänavalooma leidmisest teada anda on teha sellekohane postitus vastavas sotsiaalmeedia grupis või siis sellistele olukordadele spetsialiseerunud veebilehel (näiteks loom24.ee). Tegevus on üpriski lihtne ning üsna piiramatu.

Enamasti on sellistel kuulutustel ja portaalidel ühised levinud probleemid – kuulutused pole hallatavad (Facebooki postitused) või pole kedagi neid haldamas (loom24). Nii kaovad uued postitused vana infomüra sisse ära. Samuti puudub tihiti filtreerimine ning

võimekus vaatlejale asjakohast infot kuvada – Tallinna elanikul pole kuigi oluline näha Võrus leitud loomi. Samas, suures plussiks on väga hõlbus postitamise viis ning tihti ei nõua see ka täiendavaid kasutajakontosid.

Seega on sellistel puhkudel osalus kuulutuste pidajal minimaalne, kuid kuulutajal või niisama vaatlejal tuleb teha tükk tööd, et hoida end adekvaatse infoga kursis.

2.1.2 Kaardile visualiseeritud kuulutused

Kasutajale tunduvalt informatiivsem on kuulutuste visualiseerimine kaardile, mis võimaldab kasutajal näha just temale huvipakkuva piirkonna kuulutusi. Kuigi tänapäeval oleks sellise rakenduse loomine võrdlemisi hõlbus, siis pole see siiski kuigi levinud. Peamiselt kasutavad seda võimalust vaid omavalituste poolt hallatavad registrid. See aga seab piiranguks omavalitsuse piirid või paremal juhul rakendust haldava riigi piiri (näiteks https://gis.maricopa.gov/ACC/Stray/index.html).

Sarnaseid rakendusi on loomulikult ka eraettevõtetelt/organisatsioonidelt, kuid ka nende puhul kaasub tihtilugu infomüra liiga vanade ja aegunud kuulutuste näol või siis püütakse teha ühe rakendusega kõike – rakendus kõikide kadunud/leitud asjade jaoks, mis tundub küll hea idee, kuid reaalsuses loob keeruka keskkonna, mida pole kuigi lihtne kasutada (näiteks traista.com).

Niisiis teevad sellised rakendused vaatleja jaoks töö oluliselt lihtsamaks, vähemalt info asjakohasuse osas. Samas nõuab kasutajalt detailsemat vormi täitmist.

2.2 Eesmärgi püstitus

Eelnevast lõigust selgus, et ühel või teisel olemasoleval rakendusel on ühed või teised vead. Käesoleva töö eesmärgiks on luua võimalikult lihtne kuid samas funktsionaalne rakendus, millel oleksid olemas järgmised funktsioonid (seda vähemalt veebikliendil, ideaalis ka mobiilikliendil):

- kasutaja peab saama sisestada raporti (kuulutuse);
- kasutaja peab saama sisestatud raportit redigeerida;
- kuulutus peab olema kuvatav kaardil;
- kasutaja peab saama kaardil olevaid tulemusi filtreerida;
- kasutajal peab olema võimalus kuulutajaga kontakteeruda;

- raportid (kuulutused) peavad olema ajakohased;

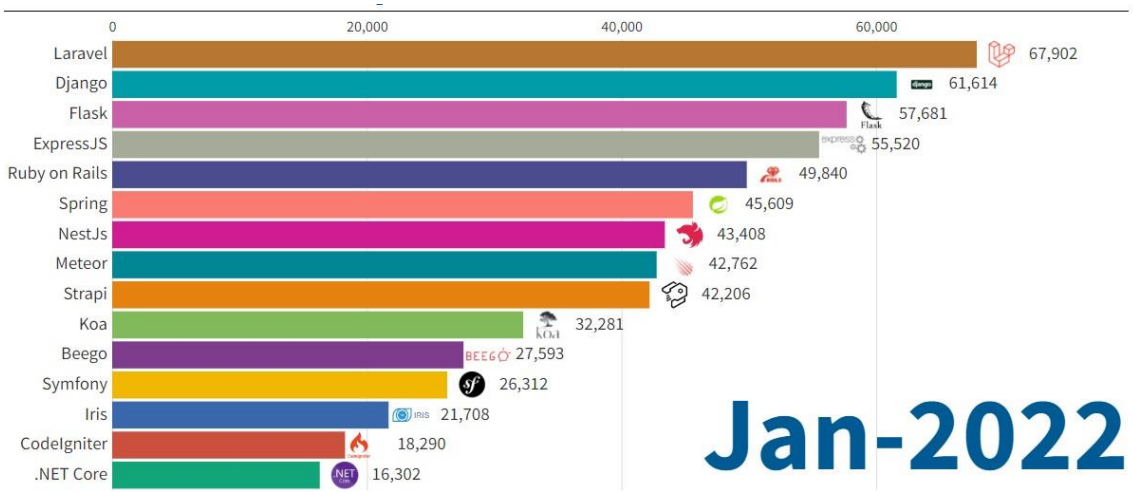
Antud töö eesmärk on prototüübi loomine ning see ei hõlma hinnangut projekti tasuvusele ega muudele kaasnevatele kulutustele ja tegevustele.

3 Tehnoloogiate valik

Vahendeid, kuidas püstitatud probleemi lahendada võib leida üüratult palju – juba ainult märkimist väärivaid programmeerimiskeeli on maailmas sadu [6]. Seega on valik üsna lai, kuid antud töö tegemiseks tuleb siiski valida vahendid populaarsemate seast, ning isiklike kogemusi kaalutledes, sest uue raamistiku rahuldaval tasemel selgeks tegemine võtab ligikaudu kuu [7] eeldades et vastav programmeerimiskeel on juba selge. Seega ajaline piirang kitsendab valikuid oluliselt, kuid valikud siiski jäävad. Järgnevalt kirjeldataksegi projekti mõistlike osade kaupa milliste tehnoloogiate vahel valida ning põhjendatakse, miks langes valik just valitule. Tasub meeles pidada, et iga tehnoloogia järel on eeliste ja puuduste väga subjektiivne loetelu ning see mis on ühe alla märgitud ei tähenda, et sama asi teistel puuduks lihtsalt selle pärast, et seda teise all kirjas pole. Need nimekirjad väljendavad üldist allikate autorite arvamust vaadeldavast tehnoloogiast.

3.1 Tagarakendus

Kümnest hetkel populaarseimast tagarakenduse arendamise raamistikust [8] tuleb antud projekti raames kohe välja jätta sellised, mille programmeerimiskeelega autoril puudub seni igasugune kokkupuude (Ruby on Rails (Ruby), Phoenix (Elixir)) ning ka sellised, kus küll keelega on autor tuttav, kuid raamistikuga mitte (Django (Python), Laravel (PHP), CakePHP (PHP), Flask (Python)) ning Koa (NodeJS)). Kahjuks eemaldusid sellega ka ühed populaarseimad (Joonis 1) [9]. Järgi jääb 3 – Spring (Java), Asp.NET (C#) ja ExpressJS (NodeJS) mida lähemalt analüüsitaksegi.



Joonis 1. Populaarseimad raamistikud tagarakenduse arendamiseks 2022. aasta jaanuari seisuga. Andmed põhinevad GitHubi tähtedel. Allikas: kuvatõmmis lehelt Statistics & Data [9].

3.1.1 ExpressJS

ExpressJS, nagu nimigi ütleb, on JavaScripti raamistik. Levinud arvamuse järgi pole JavaScript kuigi sobiv tagarakenduse kirjutamiseks, kuid sellegipoolest on just viimastel aastatel jõudnud ExpressJS populaarsemate raamistike sekka [10]. ExpressJS on kergekaaluline, platvormide ülene ning suure jõudlusega. Levinuim paketihaldur on NPM.

ExpressJS eelised [11]:

- Võimalus vähendada arenduspinu, kui nii taga- ja eesrakendus on mõlemad kirjutatud JavasScriptis.
- Algusest peale asünkroonne.
- Kiire ja lihtne arendamine, lihtne seadistada ning lihtne ühendad andmebaasidega.

ExpresseJS puudused [12]:

- Pole parim suurte projektide jaoks – selliste jaoks saab valida mõne muu NodeJS raamistiku.
- Teekidel tihtipeale kaheldav kvaliteet.
- Suur sõltuvus tagasikutsetest (nn *callback hell*).

3.1.2 .NET Core

.NET (loe: dot-net) on avatud lähtekoodiga, kuid aktiivselt Microsofti poolt arendatav raamistik. Raamistik kasutab C# keelt ning on endiselt üsnagi populaarne platvormiüleste veebirakenduste loomiseks. Erinevalt paljudest teistes, on andmete pärimine väga naturaalne, seda tänu C# sisseehitatud LINQ komponendile, mis muudab andmebaasi päringud loomulikult kirjutatavaks ja ka loetavaks [13]. Arendaja saab valida lausa 3 keele vahel: C#, F# ning Visual Basic .NET ning pakettide haldusega tegeleb NuGet [14].

.NET Core eelised:

- Suur võimekus. Erinevates võrdlusanalüüsides näitab .NET Core paremaid tulemusi nii erinevate kiiruste mõõtmiste kui ka peitaegades (*latency*), seda nii võrreldes ExpressJS-ga [15] kui ka Spring Booti-ga [16].
- Platvormide ülene – ühise C# koodibaasi pealt saadakse sujuv ja ennustatav kasutajakogemus erinevatel platvormidel [17].
- Asünkroonsus – arendajal on võimalus kasutada asünkroonse programmeerimise mustreid [18].

.NET Core puudused [19]:

- Kuigi raamistik üldiselt on avatud lähtekoodiga, on see siiski suuresti sõltuv Microsofti suvast.
- Keeruline üleminek muudelt raamistikelt, sh .NET Framework-lt.

3.1.3 Spring Boot

Java on ilmselt üks levinumaid ning siiani väga populaarne programmeerimiskeel [20]. Seega, vähemalt keele osas enamikele arendajatel üllatusi ei paku ning on alati turvaline valik. Olemuselt Spring Boot lihtsustab Spring rakenduste konfigureerimist ning on väga sobilik mikroteenuste arendamiseks. Väliste teekide haldajatest on populaarseima Maven ja Gradle.

Spring Booti eelised [21]:

- Lihtsaim viis alustada Spring rakenduse arendamist.
- Lihtsam arendus tänu *dependency injection*ile.

- Aitab vältida manuaalset trafaretse koodi kirjutamist.

Spring Booti puudused [21]:

- Teadmata Springi alustehnoloogiaid, võib hilisem arendus minna keerulisemaks.
- Ei sobi hästi suurte projektide ning monoliitide arendamiseks.

3.1.4 Raamistiku valimine

Nagu eelnevast lõigust selgus, siis on kõigil kolmel omad tugevad plussid ning tegelikult siin kontekstis üsna marginaalsed miinused – ilmselt sobiks ükskõik milline neist tagarakenduse tehnoloogiaks suurepäraselt.

Kuna Springiga on autoril kokkupuude vaid vastavalt kursuselt IT Kolledžist – kuigi raamistik on tuttav ja ka Javaga piisavalt kokku puutunud, kuid kuna pole kunagi ühtegi rakendust korralikult sellega ehitanud, siis pigem välistada see raamistik.

Erinevate JavaScripti tagarakenduse raamistikega on autor tuttav töölt, kuid ka siin pole autor ise päris nullist korralikku rakendust ehitanud. Samas on JavaScriptis (TypeScriptis) tagarakenduse tegemine köitev valik, just nimelt seetõttu, et see võimaldaks kirjutada projekti kõik kolm osa ühes programmeerimiskeeles. Samas tuleb tõdeda, et ajaline ressurss on väga piiratud, ning seetõttu pole mõistlik riskida sisuliselt tühjalt lehelt uue raamistikuga rakenduse ehitamist.

Seevastu C# ja .NET Core rakendusi on autor kirjutanud täies ulatuses nullist ning neid baasteadmisi on võimalik kiirelt rakendada ka selle projekti realiseerimisel. .NET-i kasuks räägib ka suurepärane toetus internetist saadaolevatel kasutajakogemustel ja juhistel ning väga mugav arenduskeskkond JetBrains-i Rider tarkvara näol ning ka keelena on C# intuitiivne ning hõlpsasti õpitav.

Seega kirjutatakse tagarakendus C# keeles, kasutades .NET Core raamistikku.

3.2 Andmebaas

Teades nüüd tagarakenduse raamistikku, saab edasi liikuda andmebaasi valimise juurde. Siinkohal tasuks kaaluda lisaks tänapäeval enimlevinud relatsioonilistele andmebaasidele ka NoSQL-i, kuna see on just mõeldud kiiremaks arendamiseks, võimaldab paindlikumaid struktuure ning on ideaalne suuremate andmemahtude juures, kus on

oluline kiirus ja lühike peitaeg. Skaleeritavuse ja kiirusega makstakse lõivu aga andmete järjekindluses [22].

Antud projektis vähemalt prototüübi tasemel pole suuri mahte ning hetkel pole oluline ka kiirus ning suhtlustihedus andmebaasiga on pigem tavaline kui intensiivne Seega esialgu on kindlasti parem valida traditsiooniliste relatsiooniliste andmebaaside vahel. Põhiolemuselt on nad kõik üsna sarnased ning enamiku neid .NET Core raamistik ka läbi Entity Framework-i (EF) toetab [23]. EF on objekt-relatsioon vastendamise raamistik .NET rakendustele, mis võimaldab arendajal töötada andmetega kui objektidega ning mitte fokuseerida aluseks olevale tabelitele ja veergudele kus need andmed salvestatud on [24]. Kui vaadata populaarsemaid andmebaasimootoreid aastast 2021 (Joonis 2) [25], siis on seal mitmeid andmebaase, millega autor rohkem või vähem kokku puutunud on.

Rank	Name	Type	May 2021	Chart May 2021
1.	Oracle	Relational, Multi-model	1269.94	
2.	MySQL	Relational, Multi-model	1236.38	
3.	Microsoft SQL Server	Relational, Multi-model	992.66	
4.	PostgreSQL	Relational, Multi-model	559.25	
5.	MongoDB	Document, Multi-model	481.01	
6.	IBM Db2	Relational, Multi-model	166.66	
7.	Redis	Key-value, Multi-model	162.17	
8.	Elasticsearch	Search engine, Multi-model	155.35	
9.	SQLite	Relational	126.69	
10.	Microsoft Access	Relational	115.40	

Joonis 2. Populaarseimad andmebaasi mootori 2021 aasta mai seisuga Arvuline väärtus tabelis on andmekoguja määratud skoor. Allikas: kuvatõmmis lehelt Statistics and Data [25]

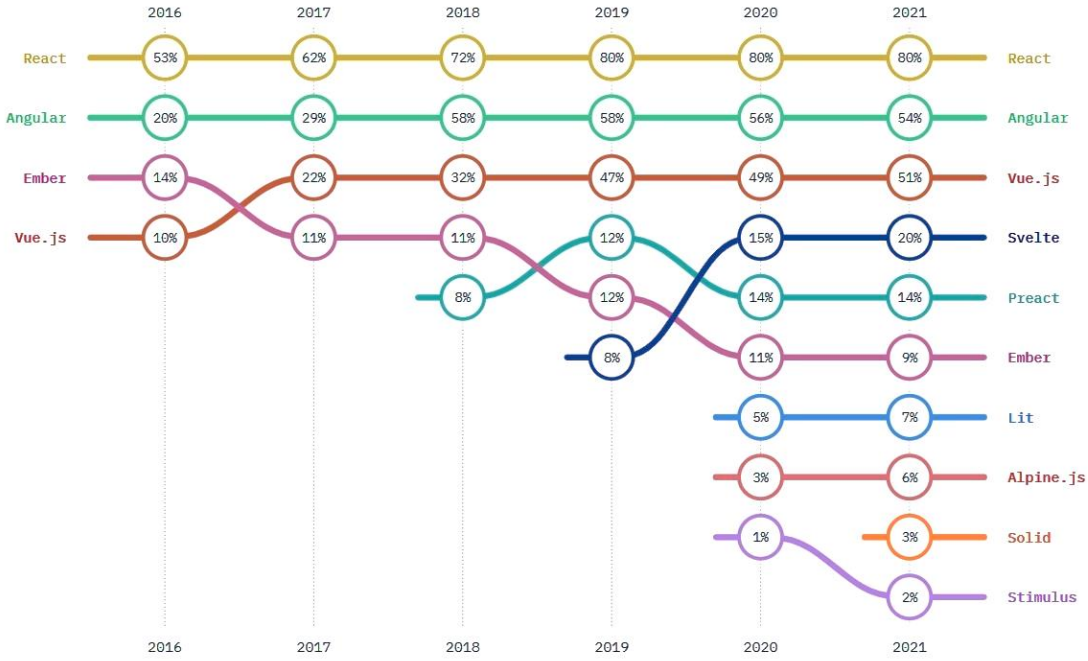
Populaarsete andmebaaside seast on autor kokku puutunud ise andmebaase luues enim SQLite ja Microsoft SQL Server-ga. SQLite sobib enam mobiilirakenduste arendamiseks. Kuna selles projektis pole mobiilirakendusel eraldi andmebaasil mõtet, siis puuduvad veenvad argumendid, miks peaks projekti andmebaasiks valima just selle. Kuna enamik populaarseid mootoreid on suhteliselt sarnase võimekusega ning need, kus erinevused esinevad pole antud projekti osas määravad, siis vähemalt arenduse jaoks on just tuttavuse poolest parim valik Microsoft SQL Server.

3.3 Veebi klientrakendus

Ilma põhjalikuma analüüsita otsustas autor, et veebiklient saab olema kirjutatud kasutades ühte JavaScripti raamistikku. Loomulikult on sellele mitmeid alternatiive, nagu ASP.NET

Web apps või suhteline uustulnuk ASP.NET Blazor, mis võimaldab samuti teha üheleherakendusi. JavaScript on veebi arendamisel siiski üldlevinud, ning ka selle projekti veebiklient saab olema JavaScriptis (TypeScriptis). Sellegipoolest jääb valida sobilik raamistik.

Seda, mis on hetkel (st aastal 2021, sest 2022 alles jookseb) populaarsemad JavaScripti raamistikud on võrdlemisi raske defineerida. Seda peamiselt seetõttu, et joon raamistiku (näiteks Vue) ja teegi (näiteks jQuery) vahel pole päris selge. Sellegi poolest võib ära tuua üsnagi adekvaatsed hinnangud ühe veebilehe järgi (küsitluses osalejate valim on küll üsna väike, kuid tulemused paistavad korreleeruvat muude allikatega) (Joonis 3) [26]. Selgelt on näha kolm teistest populaarsemat raamistikku, mida lähemalt vaadeldaksegi.



Joonis 3. Populaarsemad JavaScripti eesrakenduse raamistikud StateOfJS veebiküsitluse andmetel. Hinnang kujuneb valikute „kasutaksin veel“ või „rohkem ei kasutaks“ põhjal. Allikas: kuvatõmmis StateOfJS veebilehelt [26].

3.3.1 Angular

Angular on AngularJS järglane, ning too ilmus juba aastal 2009. Alates aastast 2016 on meil olemas selline Angular, mida tunneme. See on kirjutatud TypeScriptis ja seda arendab Google meeskond [27]. Põhiline üksus Angularis on NGMoodul, mis konfigureerivad süsteemi. Alati on olemas ka vähemalt üks Komponent (juur-komponent). Üldiselt on igal komponendil oma HTML mall, mida ta teenindab. *Router*

(suunaja) vahetab vastavalt kasutaja poolt seatud reeglitele URL aadressi või laeb vastavalt URL-le õige vaate ning logib tegevused veebisirvija ajalukku [28].

Angulari eelised [29], [30]:

- Pidev arendus ja tugi Google-lt.
- Kaksipidine andmete sidumine – muutused DOM-s kajastuvad ka rakenduses ning vastupidi.
- Kiirus ja sooritusvõime. Sisemised olemused optimeerivad ressursside kasutamist.
- Lihtne üles seada testimist.

Angulari puudused [29], [30]:

- Keeruline arhitektuur, mis teeb vea otsimise raskeks.
- Raskekaaluline. Angular on üsna mahukas võrreldes konkurentidega nagu React või Vue. Pole parim valik väiksemate rakenduste jaoks.
- Keeruline õppida.

3.3.2 Vue.js

Vue.js (loe: vjuu) arhitektuur võimaldab seda järk-järguliselt lisada juba olemasolevasse projekti mis teeb selle kasutuselevõtu väga lihtsaks. Seda saab kasutada nii töölaua kui ka mobiilirakenduste arendamiseks. Raamistiku baas toetab vaid vaate kihti – selleks, et saada kasutada keerulisemat funktsionaalsust nagu suunamised ja oleku haldamine, saab kasutada ametlikke toetavaid teeke ja pakette. Sarnaselt React-ga kasutab ka Vue virtuaalset DOM-i – luuakse DOM-st koopia, mida hoitakse JavaScripti andmestrukuuridena. Muudatused rakendatakse sellele koopiale ning hiljem võrreldakse originaalse struktuuriga ning laetakse DOM-i. Nii nagu Angularigagi, on ka siin võimalik kasutada kaksipidist andmete sidumist.

Vue.js-i eelised [31]:

- Kergekaaluline ja kiire. Erinevalt Angularist jääb Vue 2 projekti maht umbkaudu 30KB (gzipped formaadis) juurde.
- Lihtsus ja hästi dokumenteeritud.
- Kaksipidine andmete sidumine.

- Komponendi-põhine arhitektuur. Kõik komponendi jaoks olulised osad (HTML, CSS, JS) võib kirjutada ühte faili. Selline lähenemine teeb vigade leidmise ja testimise lihtsamaks.

Vue.js puudused [32], [33]:

- Kuigi Vue on kasutusel mitmete suurte firmade poolt, siis puudub kindel toetaja ja seega kindel finantstugi.
- Noor ja arengufaasis.
- Liiga palju valikuvõimalusi, mis võib raskendada tööd suurtes kollektiivides.

3.3.3 React

React sai alguse aastal 2013 ning on aktiivselt arendatav Meta poolt. Sellega saab luua keerulisi kasutajaliideseid kasutades väikesi isoleeritud komponente. Sarnaselt Vuega, kasutab ka React virtuaalset DOM-i, kuid andmete sidumine on ühepoolne. Vaadete väljakuvamine toimub JSX abiga. JSX on JavaScripti süntaksi laiendus – see näeb küll mõnevõrra välja nagu HTML-i mall, kuid omab JavaScripti võimekust [34]. React vastutab ise peamiselt visuaalse poole ja oleku haldamise eest, selleks et ehitada täielikku projekti, on enamasti vaja lisada täiendavaid teeke või mooduleid [35].

React-i eelised [35], [36]:

- Korduvkasutatavad komponendid. React sunnib arendajat mõtlema nii, et see looks väikeseid, mõtteliselt iseseisvaid üksusi, mida saab teiste komponentide sees korduvalt kasutada.
- Pigem lihtne omandada.
- Ühesuunaline andmete sidumine, mis tagab koodi stabiilsuse.

Reacti-i puudused [35], [36]:

- Kehv dokumentatsioon. Ametlik dokumentatsioon on üsna puudulik, kuid seevastu on laialdane kaas-arendajate tugi veebis.
- Ei sisalda esmainstallil kohe kõiki vajalike vahendeid täisväärtusliku rakenduse ehitamiseks.
- Järjendite kuvamine võib osutuda ressursimahukaks.

3.3.4 Raamistiku valimine

Autor on küll JavaScripti ja TypeScriptiga üsna tuttav, kuid täielikult uue raamistiku ja, nagu eelnevalt selgus, veel potentsiaalselt keeruka, õppimine ei mahu antud projekti skoopi. Seega peab valikust elimineerima Angulari. Jäävad Vue.js ja React. Mõlemaga on kokkupuude olemas, Reactiga küll tunduvalt suurem. Mõlemad jagavad ühiseid omadusi nagu virtuaalne DOM, TypeScripti tugi, PWA tugi, reaktiivsus ja komponendiline struktuur ning väga suur kasutajaskond, mis tagaks piisava toe. Kui Vue tuleb pakist välja täisväärtusliku raamistikuna või täiendavad vajalikud teegid ja komponendid on loodud enamasti Vue enda meeskonna poolt, siis Reacti tuleb hakata täiendama kolmandate osapoolte poolt loodud komponentidega [37].

Samuti tuleks etteruttavalt mõelda ka mobiilirakenduse peale, et arendustempot seal veidi kiirendada läbi võimaliku loogika või komponentide taaskasutuse. Mõlemad sõelale jäänud raamistikud on sobilikud PWA-de arendamiseks, kuid kuna autoril oli negatiivne kogemus PWA tegemisel kasutades nii Vue-d kui Reacti, siis PWA kui selline mobiilirakendusena seekord kaalumisele ei tule. Vue-l on üsna mitu satelliit raamistiku, mis on mõeldud mobiilirkanduste arendamiseks (näiteks Vue Native või Iconic), siis autor pole neist ühtegi kasutanud. Seega oleks mõistlik valida React, mille platvormiülese mobiilirakenduse arendamise raamistikuga React Native on arendaja juba tuttav ning Vue ja Reacti välja toodud erinevused ei oma määravat kaalu selle projekti realiseerimise mõttes.

3.4 Mobiilirakendus

Kuna enamasti nähakse hulkuvaid loomi väljas viibides, siis on mugav neid raporteerida just mobiilis, samas on kasutusmugavus veebirakenduses suuremal ekraanil tunduvalt parem. Seetõttu on veebirakendust mõeldud saatma ka mobiilirakendus.

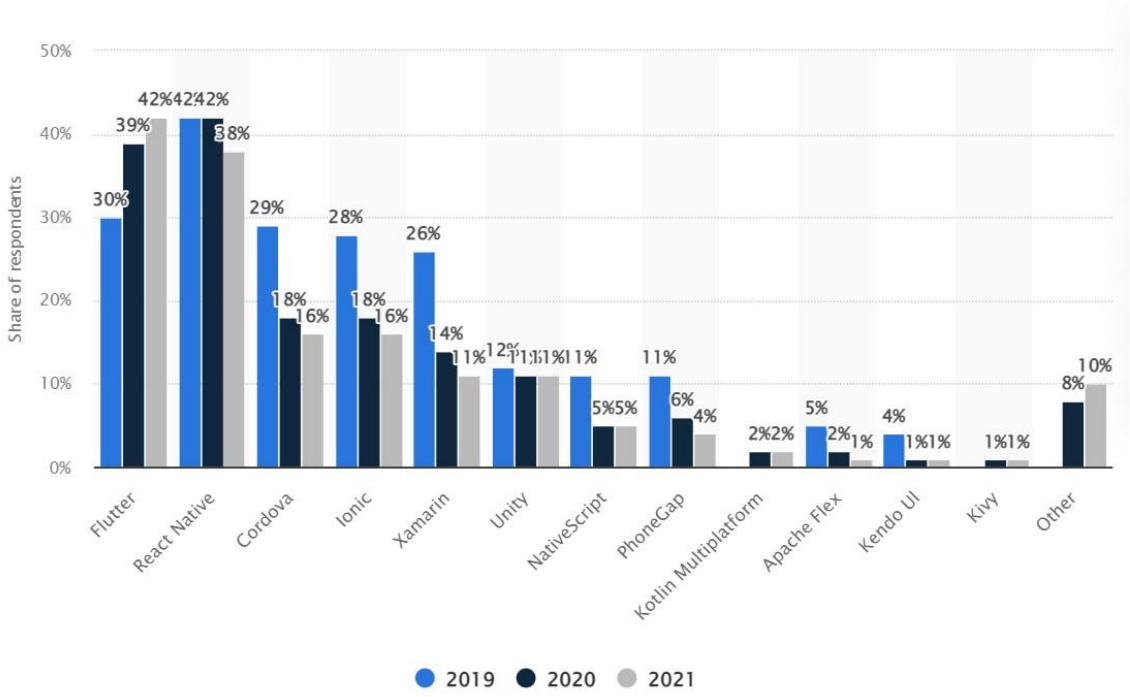
Mobiilirakenduste arendamise tehnoloogiad võib mõtteliselt jagada kolmeks:

1. Platvormipõhine arendamine (omarakendused ehk *native applications*). Arendus toimub igale mobiiliplatvormile (tänapäeval peamiselt Android ja iOS) eraldi, kasutades selleks ettenähtud vahendeid.
2. Platvormide ülene arendamine (platvormist sõltumatud rakendused ehk *cross-platform applications*, selles kontekstis on kasutusel ka termin

hübriidmobiilirakendused). Mõlemad rakendused jagavad ühist koodibaasi, mis siis pakitakse platvormispetsiifilistesse konteineritesse. Selline arendus kiirendab mobiilirakenduste ehitamist, kuid samas ei ole siin võimalik saada juurdepääsu kõikidele ressurssidele, mis on platvormipõhistel rakendustel. Samuti kaasneb sellega tihti veidi kehvem disain ning uuendamine ei saa toimuda nii kiiresti [38].

3. Progressiivsed veebirakendused (PWA). Need on väga tugevalt lihtsustades sisuliselt mobiiliekraanile mahutatud veebilehed, mida kasutaja saab oma avaekraanile paigutada. Üldiselt lihtne arendada, kuid ühilduvust ja samasugust kasutajakogemust ei saa garanteerida, sest erinevad veebilehitsejad toetavad PWA omadusi erineval tasemel [39].

Aja kokkuhoiu mõttes ei minda selle projekti raames arendama platvormipõhiselt vaid kaalutakse kas universaalset PWA-d või platvormide ülest arendust. Kuna autor on seni vaid ühe PWA rakenduse teinud ning seejuures ilma raamistikke kasutamata, siis pigem riskide maandamiseks mitte kulutada väärtuslikku ajalist ressurssi hetkel selle õppimiseks. Autor eelistab selles projektis arendada mobiilirakendust kasutades ühte hübriidtehnoloogiates, millega on varasem kogemus, ning mis on ka ühed populaarseimad selles valdkonnas (Joonis 4) [40]. Seda enam, et kahte neist on võimalik hõlpsasti siduda juba varem valitud tehnoloogiatega. Järgnevalt vaadeldakse kolme platvormiülest mobiilirakenduse arendamise tehnoloogiat.



Joonis 4. Populaarseimad platvormiüleste mobiilirakenduste rakenduste raamistikud aastatel 2019-2021. Allikas: kuvatõmmis Statista veebilehelt [40].

3.4.1 Flutter

Google toode Flutter ilmus aastal 2018 ning pakub suures valikus UI elemente, sellel on tugev kuvamismootor ning tuleb kuum-värskenduse (*Hot Reload*) (arendamise ajal värskendamine ilma taaskäivitamiseta) funktsiooniga [41]. Kasutatakse Google enda loodud keelt Dart. Flutter on konkurentidest kiirem ja võimekam, just tänu Darti kompilaatorile ja enda vidinatele (*widgets*) [42]. Lisaks mobiilirakendustele saab Flutterit kasutada ka töölaua rakenduste loomiseks [43].

Flutteri eelised [42]:

- Laialdane dokumentatsioon.
- UI kiire reageerimisvõime.
- Võimaldab luua kauneid kasutajaliideseid, mis ei jää maha platvormipõhistest.
- Kuum-värskenduse funktsioon tagab kiire arenduse.

Flutteri puudused [44]:

- Võrreldes teiste raamistikega võivad rakendused kujuneda mahukamaks.
- Dart pole kuigi populaarne programmeerimiskeel.

3.4.2 React Native

React Native on olnud pikalt antud segmendi liider ning alles viimasel aastal pidanud oma esikoha loovutama Flutterile [40]. Raamistik lasti välja aastal 2015 Facebooki poolt ning saavutas kiiresti populaarsuse. Nii nagu React, kasutab ka React Native JavaScripti, et ehitada üles komponendid nii, et tavakasutaja ei erista neid platvormipõhistest. Lisaks võimaldatakse kirjutada ka Javas või Swiftis omarakenduslike mooduleid. Ka React Native omab kuum-värskenduse funktsionaalsust [42].

React Native' eelised [45]:

- Jõudlus on lähedane omarakendustele.
- Rohkelt valmislahendusi ja kolmandate osapoolte loodud teeke.
- Suur kasutajaskond ning tugev eestvedaja Meta näol.

React Native' puudused [45]:

- Keeruline siluda.
- Ei sobi hästi, kui on vaja kasutajaliideses saavutada keerulisi viipeid.
- Uuendused on suured ning neid tuleb tihti.

3.4.3 Xamarin

Esitatud raamistikest vanim on Xamarin, sündides juba aastal 2011. Viis aastat hiljem ostis Microsoft selle ära ning nüüd jätkab see raamistik oma elu .NET maailmas. Xamarinis hakkama saamiseks tulebki hästi orienteeruda nii .NET-s kui ka C#-s. Raamistik jaguneb kaheks: Xamarin.Native ja Xamarin.Forms. Esimene prioritiseerib äriloogikat ning kasutades omarakenduslike (*native*) vahendeid tagab peaaegu omarakenduse lähedase võimekuse. Teise eesmärk on jagada kogu koodi, mitte ainult äriloogikat kuid sellega makstakse lõivu võimekuses. Xamarin.Forms kasutab üldiseid standardseid UI elemente, mis suurendab koodi taaskasutust, kuid UI tulemus võib jääda nähtavalt alla sellele, mis oleks omarakendustel [42].

Xamarini eelised [46]:

- Kiire arendustsükkel kuna enamiku koodi annab taaskasutada.
- Xamarin.Native-ga saavutab omarakendusega sarnase kasutajakogemuse.
- Xamarin.Native-ga saavutab omarakendusega lähedase jõudluse.

Xamarini puudused [46]:

- Suhteliselt keeruline.
- Vähese funktsionaalsusega. Xamarin pole sobilik, kui rakendus kasutab keerukat UI-d.
- Suhteliselt suur rakenduse suurus.

3.4.4 Raamistiku valimine

Kõigi kolme raamistikuga on autoril võrdväärne suhe, kuid ilmselt muid aspekte arvestamata kõige meeldivam neist on autorile Flutter. Kuid sellest subjektiivsest meeldivusest antud kitsas ajaraamistikus jääb väheseks – tuleb valida selline raamistik, millega arendamine antud kontekstis võtaks kõige lühema aja. Seega tuleb Flutter kõrvale jätta, sest kuigi nullist arendamine on sellega üsna kiire, siis tooks see ikkagi mängu kolmanda programmeerimiskeele, mis tähendaks, et midagi olemasolevast ei annaks eriti taaskasutada ja keele korralikuks omandamiseks kulub samuti omajagu aega.

Xamarini eeliseks on see, et sellisel juhul kasutaksid tagarakendus ja mobiiliklient mõlemad keelena C#-i. Kuigi tagarakenduse koodi ei anna kuidagi mobiilirakenduses taaskasutada, on ühine keel siiski suur aja kokkuhoid. Samas on arendamine Xamariniga pigem keerulisevõitu ning nõuaks ilmselt üsna palju dokumentatsiooniga tutvumist.

React Native kasutab seevastu TypeScripti ning ka veebiklient saab olema kirjutatud TypeScriptis kasutades Reacti. Seega on potentsiaalselt suur võimalus vähemalt mingit osa koodi nende kahe vahel taaskasutada. Sellest tulenevalt paistab React Native olevat parim valik antud projekti mobiilirakenduse loomiseks.

4 Realisatsioon

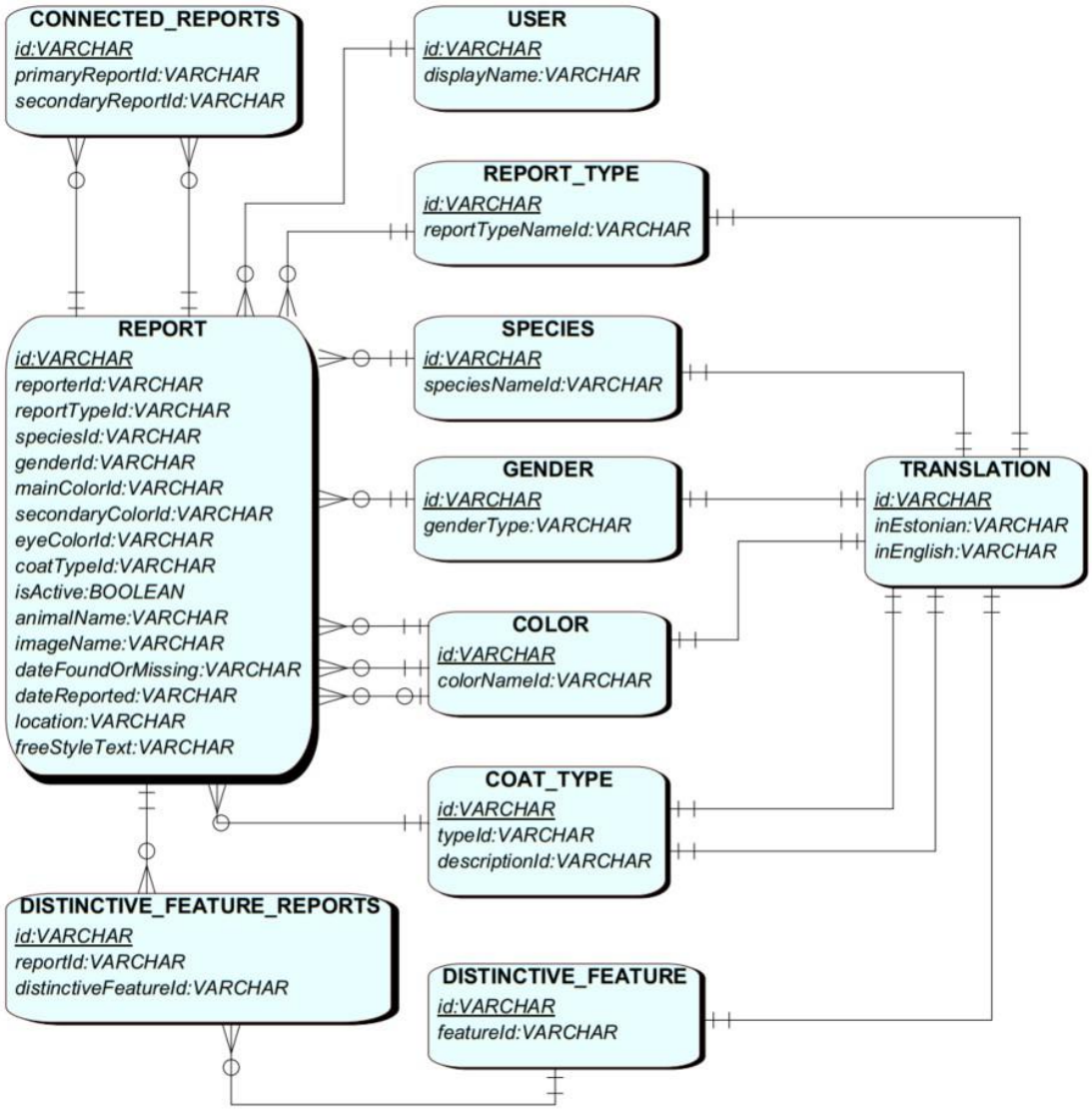
Järgnevalt kirjeldatakse kuidas projekt valmis ehitati. Alustatakse andmebaasist, seejärel tehakse põgus ülevaade loogikast nii kliendi kui ka süsteemi vaatest, järgneb põhifunktsioonide ja disainielementide tutvustus ning viimaks antakse ülevaade arendusprotsessist.

4.1 Andmebaas

Prototüübi loomiseks tehti üsna lihtne andmemudel (Joonis 5). Suur hulk tabeleid on selles lihtsalt nn sõnaraamatud (*dicitionaries*), kus hoitakse valikväärtusi. Selliseid tabeleid võiks asendada ka koodi sisse kirjutatud väärtustega, kuid see muudaks kogu projekti üsna paindumatuks, samuti raskendaks tõlgitavust. Peab kohe ütlema, et nii lihtne andmemudel toimib ainult prototüübi tasemel, kus on põhimõtteliselt eeldatud, et kaotatakse või leitakse mõni karvane loom (kass või koer). Reaalses elus pole nende sõnaraamatute sissekanded sobilikud iga loomaliigi jaoks ja seetõttu tuleb lisada näiteks väli, mis väljendab, mis liigi kohta see sisend sobida võiks ja loomulikult vahepealsed tabelid jms.

Tagarakendus jaguneb .NET traditsioonide kohaselt projektideks. Otseselt andmebaasiga on seotud projektid `Database` ja `Domain`. Viimases defineeritaksegi klassidena ära olemid ja läbi selle ka andmebaasi struktuur. Nagu eelnevalt mainitud, on .NET Core' sisseehitatud objekt-relatsioonivastendaja (ORM) Entity Framework, mis nende klasside alusel loob andmebaasi tabelid ning vastutab kogu järgnev andmete käitlemise eest.

Projektis `Database` asub klass `AppDbContext`, kus defineeritakse, millised klassid saavad olema andmebaasi tõlgitud. Samuti, kui vaja, siis saab siin ka täpsustada olemite suhteid ja ka näiteks, mis peaks juhtuma, kui mingi sissekanne ära kustutatakse. Siin asuvad ka repositooriumid andmebassiga suhtlemiseks ning ka nn Tööühiku klass (*Unit of Work*), mis võimaldab repositoorium-mustrit efektiivsemalt kasutada vahendades `DbContext`-i instantsi repositooriumitele [47].



Joonis 5. Andmebaasi olem-seos diagramm.

Kasutajal on võimalus ka lisada oma raportile pilt loomast. Pilte ei salvestata andmebaasi vaid staatiliselt serverisse.

Kuna olemasolevate rakenduste oluline puudus on nendes kuvatavate kuulutuste ajakohasus (kord sisestatud kuulutus jääbki sinna tihtipeale igaveseks), siis siin püütakse seda lahendada andmebaasi tasandil. Selleks saab kasutada SQL Server Agent-i ning lisada sinna „töö“ (*job*), mis kord nädalas käivitab käsu, mis on toodud Joonis 6. Sellega muudetakse sellised raportid, mida pole kuu aega uuendatud inaktiivseks.

```
UPDATE [PetApp].[dbo].[Reports] SET IsActive = 0 WHERE IsActive != 0
AND GETDATE() > DATEADD(month, 1, UpdatedAt);
```

Joonis 6. SQL Server Agent’i töö SQL käsk.

4.2 API otspunktid

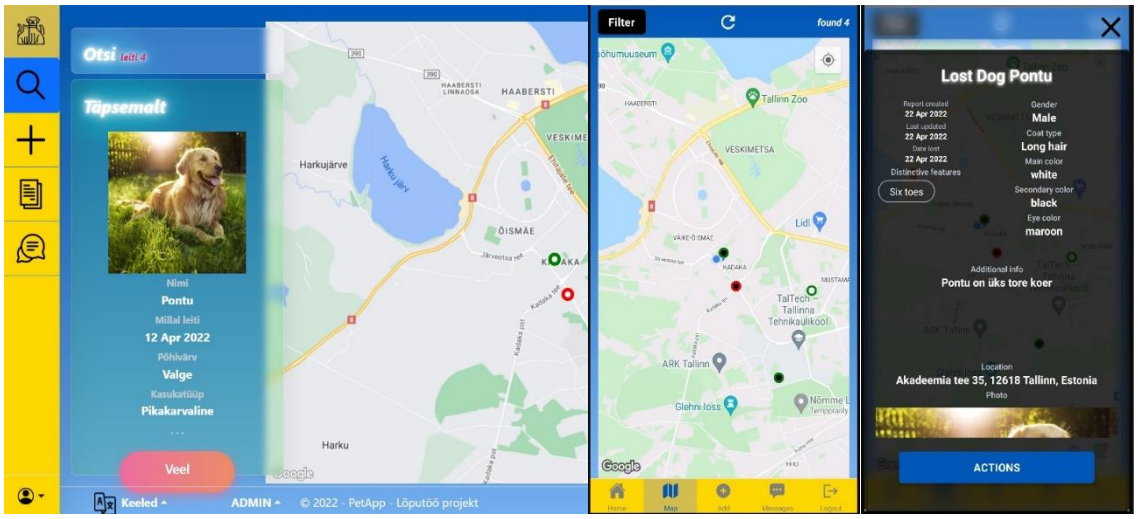
Eesrakendusi teenindavad kokku 21 API otspunkti. Neist 2 on identiteedi jaoks (sisse logimine ja registreerimine), 3 vajalikud kaardile info kuvamiseks, 3 kasutajatevahelise sõnumineerimise jaoks, 7 raportite haldamise tarbeks. Lisaks on 2 kultuurikeskkonna seadmiseks ning viimaks 4 otspunkti on demona admin-õigustes kasutaja jaoks ühe andmebaasi sõnaraamatu haldamiseks (sellest lähemalt peatükis 4.4). APIde struktuuriga saab tutvuda Swaggeri abil mille saab kätte tagarakenduse käivitamisel /swagger aadressi laiendusel.

Loodud struktuur pole ideaalne, kuid tagab antud prototüübi toimise. Ükski otspunkt pole eraldi mobiilirakenduse või eraldi veebirakenduse jaoks – kõik on ühine. Kaheldava väärtusega on otspunkt POST /Report/Image/{reportId}, mis teenindab piltide üles laadimist. Pilte saab üleslaadida veebivormi osana, raporti muu POST/PUT info läheb aga tavalise JSON-i kujul. Loomulikult võiks ka muu raporti saata veebivormina ja sinna lisada ka pilt, kuid testimise ja töökindluse huvides on need kaks otspunkti hetkel eraldatud. Tulevikus poleks ilmselt selline lahendus jätkusuutlik.

API otspunktid ei tagasta mitte tooreid andmebaasist saadud andmeid, vaid spetsiaalseid vastavalt kohaldatud andmeedastusobjekte (DTO), millega tagarakendus on püüdnud töödelda infot võimalikult palju, et eesrakendus saaks need võimalikult lihtsalt kasutusele võtta. Tagarakendus võtab ka vastu DTOsid ning vastandab need siis Domeeni objektideks.

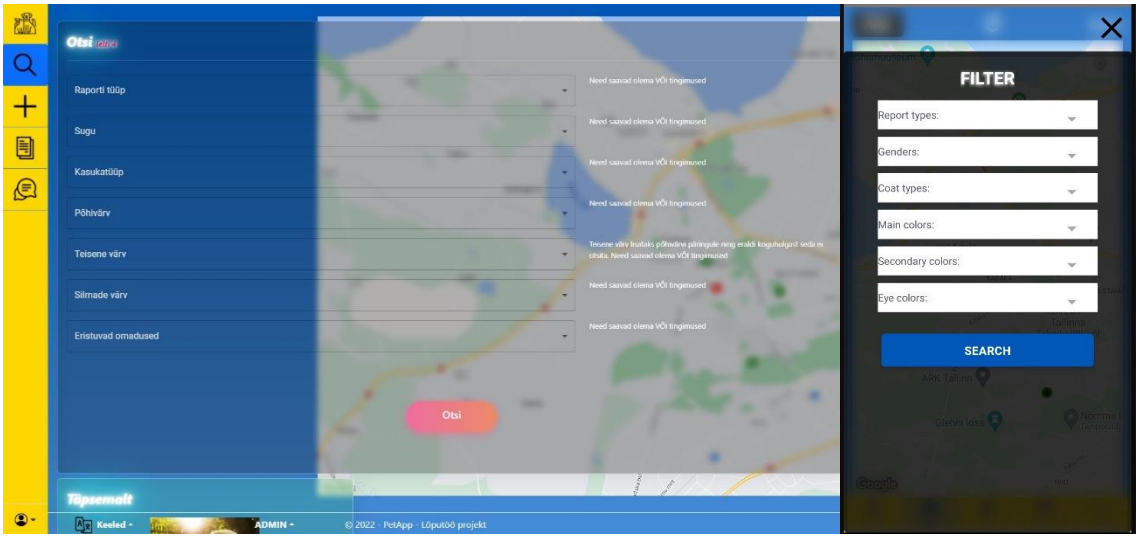
4.3 Disain ja kasutajateekond

Kasutajateekonnast on mõistlik rääkida koos disainiga. Mitte sisse loginud kasutajal on võimalik raporteid kaardivaates vaadata (Joonis 7), neid filtreerida (Joonis 8) ning detailse vaate avada. Detailses vaates, kus on kogu raport näha koos geokodeeritud aadressi ning pildiga, saab nii mitte sisse loginud kui ka sisse loginud kasutaja valida valiku „Näita seotud raporteid“. See viib tagasi kaardivaates, kus kuvatakse sellised raportid, mida süsteem peab valitud raportiga sarnasteks ning mis on 200 kilomeetri raadiuses. Tõsi, hetkel ei ole see süsteem kuigi tark ja filtreerib sisuliselt välja vaid need raportid, millega on üks ühele kattuvus mõningate raporti vormi valikutega.



Joonis 7. Kaardivaate detailide aknad kasutajaliideses. Vasakult: veebirakendus, mobiilirakenduse kaart, mobiilirakenduse detailide moodul kaardivaates. Siin ja edaspidi rakenduse kuvatõmmistel kujutatud koera pilt pärineb veebilehelt https://pawt4paws.com/cbd-for-dogs/how-to-keep-dog-cool-in-a-heatwave

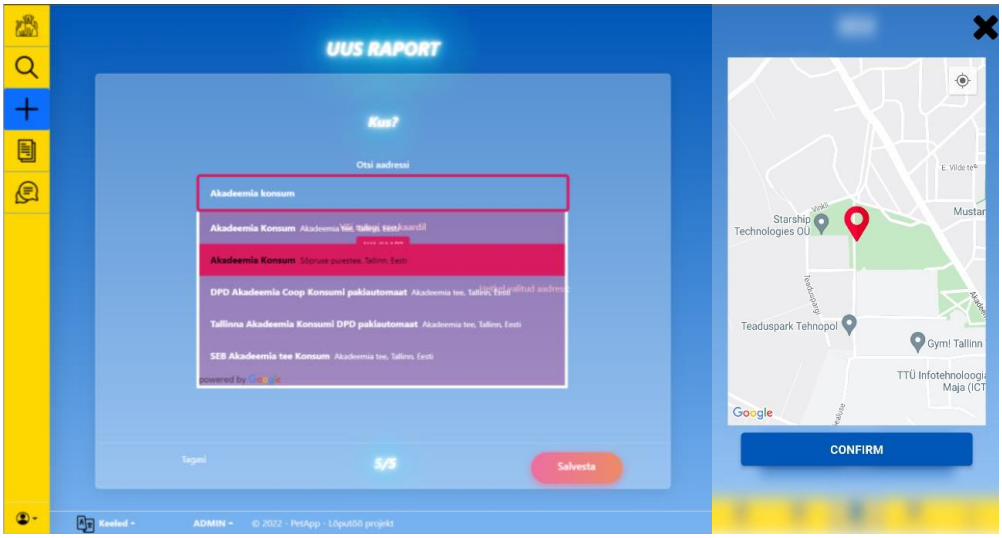
Sisse loginud kasutaja näeb detailide vaates vastavalt oma rollile valitud raporti suhtes täiendavaid valikuid. Olles raporti omanik saab ta seda siin muuta, aktiveerida või deaktiveerida ning ka kustutada. Kellegi teise raportil olles avaneb aga võimalus saata raporti omanikule privaatsõnum. Kõik privaatsõnumid on alati kindla raportiga seotud.



Joonis 8. Kaardivaate filter veebirakenduses kui ka mobiilirakenduses.

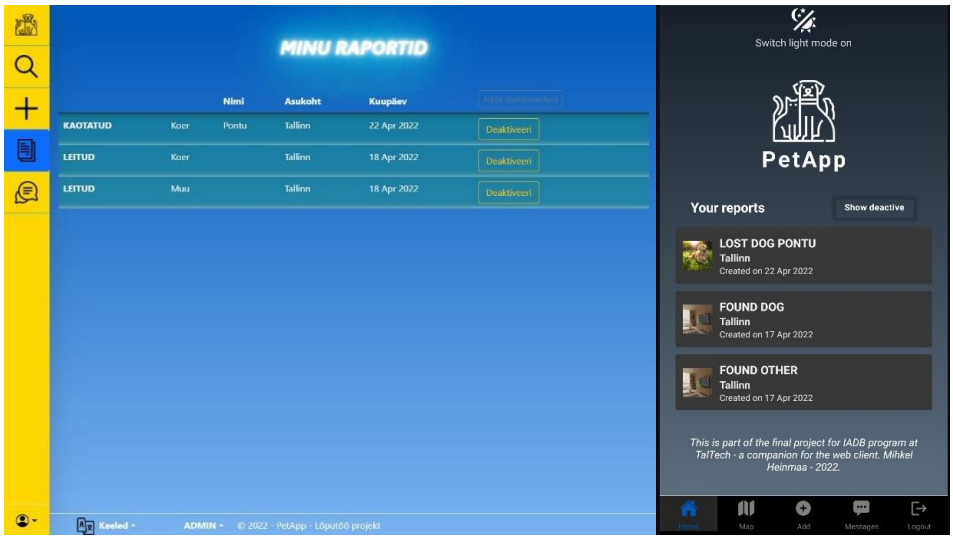
Uue raporti lisamine (ka olemasoleva muutmine) on veebi ja mobiilirakenduses veidi erinev. Mobiilirakenduses on kogu vorm üks pikk jada, veebirakenduses on vorm jagatud viieks ning iga kord enne uuele etapile suundumist toimub etapi valideerimine. Erinevus on ka asukoha määramisel. Veebirakenduses ei ole veel rakendatud asukoha markeriga kaardile märkimine, selle asemel toimub aadressi otsing ning süsteem konverteerib siis

saadud aadressi geograafilisteks pikkusteks ja laiusteks. Mobiilirakenduses on vastupidi – puudub aadressiotsing ning on vaid kaardile asukoha märkimine (Joonis 9).



Joonis 9. Raportil asukoha määramine. Vasakul veebiklient, paremal mobiilirakendus.

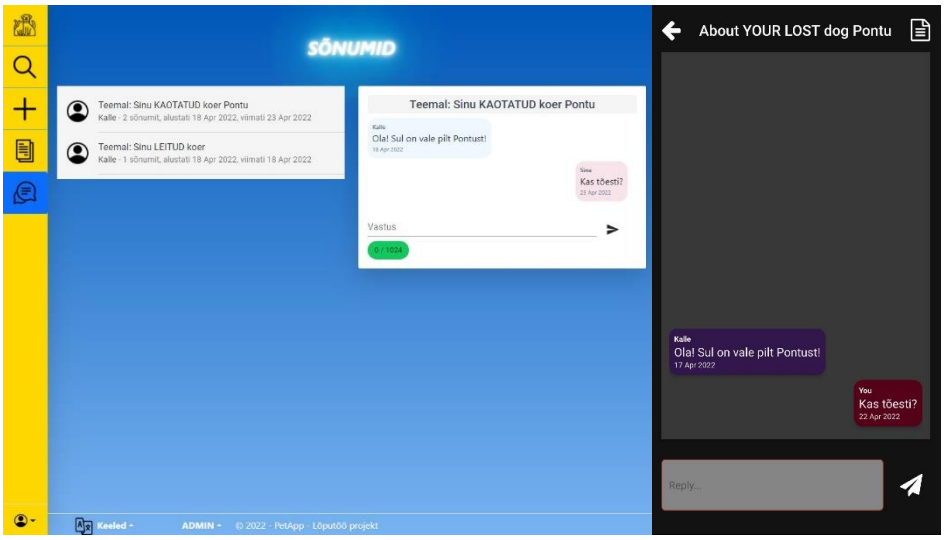
Ka koht, kus kasutaja näeb enda poolt loodud rakendusi on veidi erinev mobiili- ja veebirakenduses. Mobiilirakenduses kuvatakse kasutaja raportid kohe avalehel, veebis tuleb aga esmalt navigeerida „Minu raportid“ vaatesse (Joonis 10). Ühiselt võimaldavad mõlemad aga kuvast eemaldada mitteaktiivsed raportid (või siis vastupidi), veebivaates saab samas ka raportit aktiveerida või deaktiveerida.



Joonis 10. Kasutaja poolt loodud raportite kuvamine. Vasakul veebiklient, paremal mobiiliklient tumeda värviskeemiga.

Algne idee nägi ette, et kui raporteerija ei soovi oma isiklike kontaktandmeid raporti juures avaldada, siis ta seda ka tegema ei pea. Seepärast raporti koostamise vorm ei küsi

ka mingeid konkreetseid kontaktandmeid. Kui kasutaja siiski peaks tahtma näiteks oma telefoni numbri avaldada, siis saab ta seda teha vabatekstiliselt. Selleks, et kasutajad omavahel siiski suhelda saaksid, on loodud väga lihtne sõnumirakendus (Joonis 11). Mõlemal juhul on võimalik otse seotud raportini jõuda läbi teema päise.



Joonis 11. Sõnumivahetus kasutajate vahel. Vasakul veebiklient, paremal mobiiliklient tumedat värviskeemi kasutades.

Sellega tavakasutaja võimalused hetkel ka piirduvad. Lisaks on veebikliendis praegu vaid ühe mooduliga administratiivne vaade, mis demonstreerib kuidas toimub andmebaasi sisu tõlkimine. Veebikliendi puhul saab kasutaja valida ka keelte vahel (praegu eesti ja inglise keel), mobiilirakendus on lokaliseerimata.

Nagu eelnevat nähtub, siis on kasutajaliides üsna lihtsasti arusaadavalt ülesehitatud, puuduvad peidetud osad, mille kasutamiseks peab eelnevalt teadma kuhu vajutada või mida tõmmata. Seega võib pidada kasutajaliidest intuitiivseks. Puudustena tuleb aga välja tuua, et veebikliendis enamikes kohtades on ainuke võimalus valikute tegemiseks hiire klahvidega, disain üldiselt vajaks korralikumat läbimõtlemist ja ühtlustamist.

4.4 Internatsionaliseerimine

Antud projektis piirdub internatsionaliseerimine ja lokaliseerimine vaid tõlgetega. Tagarakendus toetab hetkel kahte keelt – eesti ja inglise ning veebiklient pakub kasutajale mõlemat valikut. .NET Core rakendustes määratakse lokalisatsiooni võimekus `Startup` klassis (Joonis 12). Ning see, kuidas lokalisatsioonipäring eesrakenduse poolt edasi

tagarakendusele antakse, märgitakse samuti sealsamas ära. Antud projektis saadetakse lokalisatsiooniinfo päringuparameetrina kuid seadistus lubab ka küpsistega seda teha.

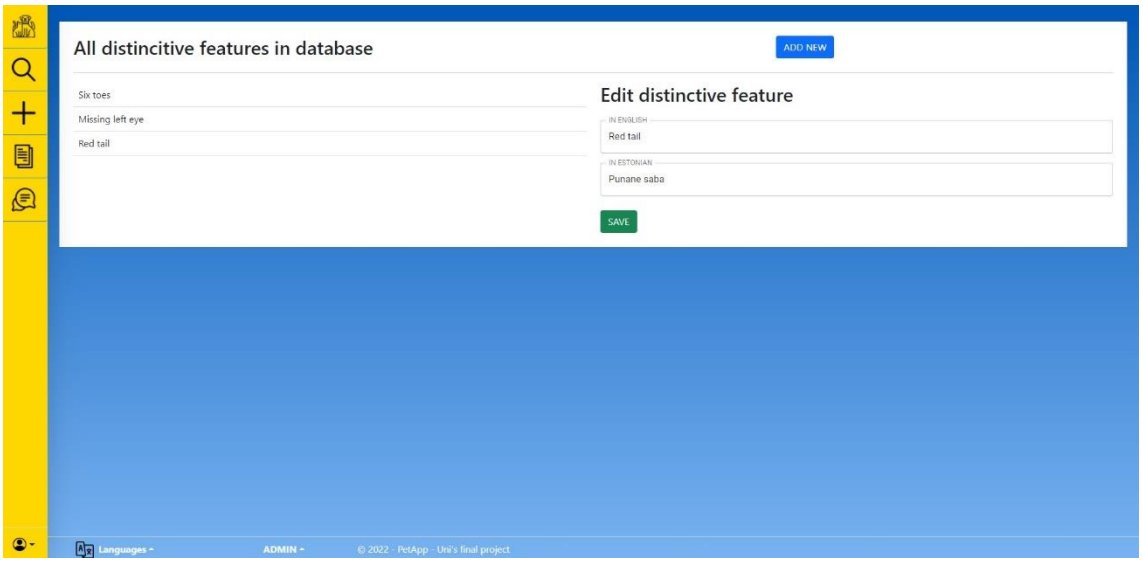
```
services.Configure<RequestLocalizationOptions>(options =>
{
    var appSupportedCultures = new[]
    {
        new CultureInfo("et"),
        new CultureInfo("en"),
    };

    options.SupportedCultures = appSupportedCultures;
    options.SupportedUICultures = appSupportedCultures;
    options.DefaultRequestCulture = new RequestCulture("en");
    options.SetDefaultCulture("en");
    options.RequestCultureProviders = new
List<IRequestCultureProvider>()
    {
        new QueryStringRequestCultureProvider(),
        new CookieRequestCultureProvider()
    };
});
```

Joonis 12. Lokalisatsiooni seadistamine .NET Core rakenduses.

Veebiklienti endasse ühtegi tõlget sisse kirjutatud pole – kõik tõlked küsitakse tagarakenduselt. Keele muutmisel saadetakse päring `GetLangResources` API otspunktile, mis siis tagastab esirakendusele kõik valitud keele staatilised tõlked. Tõlked ise asuvad `Resources` projektis .resx XML failides. Andmebaasi sõnaraamatute tõlked asuvad andmebaasis `Translations` tabelis. Siin on vastab iga suvalise andmebaasi sõnaraamatu (näiteks tabel `Species`) veeru sõne-tüübilisele atribuudi väärtusele (näiteks liigi nimetus) üks rida. Kontekstis olulised veerud tähistavad keeli, siin siis `InEnglish` ja `InEstonian`. Nüüd, kui saadetakse tagarakendusele päring (näiteks, tagasta kõik loomaliigid), siis DTO vastandamisel valitakse vastavalt lokalisatsioonile (mis, nagu eelnevalt kirjeldati, antakse edasi päringuparameetrina) õige sõne.

Sellisel lahendusel on vähemalt üks suur puudus: nimelt eesrakendus saab andmebaasi tõlke ainult siis kui see seda küsib. Seega kui näiteks kasutaja muudab eesrakenduses keelt, saadetakse päring tagarakendusele, et saada staatiliste sõnede tõlked samas andmebaasi tõlked jäävad endiseks. Selle vastu aitab korduspäring vastavale API otspunktile pärast keele vahetust või siis lihtsalt lehekülje värskendamine. Kumbki pole just kõige ökonoomsem tegevus. Positiivse poolepealt on sellised tõlked lihtsasti hallatavad ja väga ülevaatlikud. Joonis 13 kujutab ühte näidist, kuidas selliselt andmebaasi kirjeid ja tõlkeid kasutajaliideses hallata.



Joonis 13. Andmebaasi tõlgete haldamise näidis.

4.5 Arendusprotsessist

Projekti osade arendus viidi läbi väga kitsas ajaraamis, millest tingituna võib ka mitte asjatundlik vaatleja näha tulemuses konarlusi. Üldiselt võib arendusprotsessi pidada koskmudelil põhinevaks. Defineeriti nõuded, pandi paika disain, koodi kirjutamist alustati tagarakendus andmebaasist ning liiguti järkjärgult edasi kuni mobiilirakenduseni. Järgnevalt kirjeldatakse lühidalt osade kaupa, kuidas projekt valmis.

4.5.1 Tagarakendus

Andmekihtide mõttes otsustati tagarakendus ehitada kahele kihile – andmebaasi kiht (mida väljendab projekt `Domain`) ning andmete edastuskiht (mida väljendab projekt `DTO`). Keerukamate projektide korral on levinud kasutada ka nö ärikihti, kuid antud projekt on pigem lihtne ja kompaktne, ning täiendava andmekihi lisamine ei näinud olevat mõttekas.

Rakendus on jaotatud kaheksaks projektiks – seda saaks tahtmise korral õgvendada, kuid puudub otsene vajadus. Funktsionaalsete klasside olemused on enamasti määratud tema liideses. Autori arvates lihtsuse ja arusaadavuse huvides ei asu liidesed eraldi projektides vaid kõik mõtteliselt sama süsteemi osad asuvad ühes projektis. Osa projekte on juba eelnevalt tutvustatud – lisaks neile on veel `Brain`, mille ülesanne on hetkel ainult välja peilida nn seotud raportid, millest samuti juba juttu on olnud.

Rakendus ise käivitub aga projektist `WebApp`. Siin asuvad 6 API töötleja-klassi, mis vastutavad, et andmed eesrakenduse ja tagarakenduse vahel saaksid vahetatud. API dokumentatsiooni genereerib Swagger ning see võimaldab ka kasutajal proovida API päringuid otse veebilehitsejas.

Kokkuvõttes on tagarakendus üsnagi traditsiooniline, arendus kulges ilma suuremate tõrgeteta, vähesel määral kulus täiendavat aega olemi suhete täpsustamiseks. Tulevikus vajaks see aga kindlasti refaktoreerimist ning API töötlejate paremat struktureerimist-defineerimist.

4.5.2 Veebiklient

Veebikliendi struktuur on samuti jagatud mõttelisteks osadeks ning võimalusel on püütud jaotada vaated (nimetame neid konteineriteks) omakorda väiksemateks komponentideks. Sellest hoolimata on mõni konteiner selgelt ülekoormatud, ning tuleks refaktoreerida komponentideks, mis tuleks kindlasti hilisemas arenduses ära teha.

Tagarakendusega suhtlemine käib läbi teenuste-klasside „*services*“ kaustas. Enamus suhtlusest vahendab `baseService.ts`, aga spetsiaalsete olukordade jaoks on eraldi teenus-klassid sisselogimiseks ja registreerimiseks, tõlgete pärimiseks ning ka pildi üles laadimiseks. HTTP päringute tegemist vahendab Axios.

Autoriseemiseks kasutatakse JWT-d. See on rakendusepääsme standard, mis vahendab turvainfot kliendi ja serveri vahel. Pääse antakse edasi iga HTTP päringu päisega. Pääse ise saadakse sisselogimisel tagarakenduselt (pääsme võib ka salvestada ja seda mõnda aega taaskasutada, kuid antud juhul võetakse kasutusele igal sisselogimisel uus pääse). Pääsmesse on kodeeritud ka kasutaja ID (antud tagarakendus kasutab ID-na unikaalseid 128-bitilisi sõnesid) ning kasutaja rollid. Seda infot kasutab eesrakendus mitmel pool tingimuslikult info kuvamiseks.

Rakendus kasutab mitmeid visuaalseid komponente, mis on eelnevalt kellegi poolt loodud, näiteks ikoonid on alla laetud Flaticons veebilehelt. Autor on püüdnud kõik sellised elemendid koondada „*Credits*“ lehele. Lisaks on ohtralt kasutatud valmis Reacti UI komponente (peamiselt MUI (mui.com) pakutavat). Nende ja teiste kasutatud teekidega võib tutvuda `package.json` failis.

Kaarditeenuseid vahendab Google' Maps JavaScript API ning asukohta ja gekodeerimist vahendavad Google' Places API ja Geocoding API.

Veebikliendi ehitamisel oligi suuremad komistuskivid ja aega nõudvad tegevused seotud kaartidega ja Google' API-ga. Reacti jaoks on loodud mitmeid teeke, mis peaksid kaartidega tegelemist lihtsustama, kuid sellegipoolest on (vähemalt autori poolt valitud teegi puhul) kogu tegevus kuidagi kohmakas ja mitteintuitiivne. Tulevikus vajaks kindlasti kogu see osa mõningast üle kirjutamist.

4.5.3 Mobiiliklient

Mobiilirakenduse struktuur on väga sarnane sellele, mis on veebirakenduses. Mobiilirakendus ei ole lokaliseeritud, seega siin ei anta ka HTTP päringutega kaasa kultuuriinfot ja ka vastav teenus on siit puudu. See ei tähenda, et rakendust ei saa hiljem lokaliseerida – saab küll ja see näeks ilmselt väga sarnane välja sellega, mis juba veebikliendis loodud on.

Erinevalt veebist, on siin võimalik valida heleda ja tumeda värvirežiimi vahel. Kasutajaliides komponendid on samuti erinevad – kasutusel on spetsiaalsed React Native komponendid, et oleks võimalik UI ülekandmine kõikidele platvormidele. Ka siin oleks vaja suurendada koodi taaskasutust lahutades konteinerid (siin „*screen*" (ekraan) nime all) paremini väiksemateks taaskasutatavateks komponentideks. Kiirustamise tõttu on siin seis selles osas kehvem kui veebirakenduses. Ka siin kasutatakse Google rakendusliideseid (Maps JavaScript API asemel Maps SDK for Android).

Rakendust arendades ilmnesid päris mitmed tõrked, millest mõnedest lähemalt. Sarnaselt veebirakendusega, on ka siin kasutusel HTTP päringute tegemisel enamasti Axios, välja arvatud pildi saatmisel serverisse. Mingil kummalisel põhjusel ei saanud server Axiosega saadetud veebivormis aru, mis faili püütakse talle saata. Ei aidanud kõikvõimalikud moodused sisu tüüpi määrata ja muud soovitused, mida internetist leida võis. Selle asemel teeb pildi saatmisel Axiose eest nüüd töö ära Fetch API, millega sellist probleemi ei ilmunud.

Teise frustreeriva probleemina ilmnes see, et tehes päringud serverisse läbi Androidi emulaatori, annab see `HttpRequestException`-i, sest Android (ja iOS) ei usalda endasigneeritud sertifikaati. Selle kohta leidub veebis kõikvõimalike variante, kuidas

probleemi lahendada. Antud olukorras töötasid Microsofti dokumentatsioonis esitatud juhised [48].

5 Kokkuvõte

Töö eesmärk oli luua süsteem, mis aitaks kokku viia kaotatud ja leitud koduloomad nende omanikega. Selle saavutamiseks tuli luua 3 rakendust: tagarakendus, klientrakendus veebile ja klientrakendus mobiilile.

Tagarakenduse arendamiseks kasutati .NET Core raamistikku. Loodi andmebaas ning kirjutati rakendusliides, millega saaksid eesrakendused tagarakendusega suhelda. Tagarakendus võimaldab kuulutusi sisestada, neid muuta, tuvastab süsteemi arust sarnased kuulutused ning võimaldab kasutajatevahelist suhtlust.

Veebikliendi loomiseks kasutati React raamistikku. Rakendus toob tagarakenduse võimekuse kasutajani. Kuulutused kuvatakse kaardil ning kasutajal on võimalik neid filtreerida. Samuti saab kasutaja valida kahe keele vahel, seejuures on tõlgitud lisaks staatilistele sõnedele ka andmebaasi kirjed, mis ei ole kasutaja poolt modifitseeritavad.

Mobiilikliendi tegemiseks kasutati React Native raamistikku, mis võimaldab kirjutada korraga rakendust nii Androidile kui ka iOS-le. Mobiilirakendus sarnaneb üldjoontes veebikliendile. Kasutaja saab valida heleda ja tumeda värvirežiimi vahel.

Töö tulemusena loodi funktsioneeriv prototüüp. Edasises arenduses tuleb täiendada andmete pärimise loogikat tagarakendusest, et optimeerida mahte. Andmebaasi struktuur tuleb muuta loomaliigist sõltumatuks. Vaja on refaktoreerida ja puhastada koodibaas, et vähendada dubleerimist ning ühtlustada meetodeid. Parandada kasutjaliidest ning ei teeks kindlasti halba ka muuta sarnaste kuulutuste leidmise loogika targemaks kui see seni on.

Kasutatud kirjandus

[1] K. Mariste, „Pealinn,“ Tallinna linnavalitsus, 15 01 2021. [Võrgumaterjal]. Available: https://pealinn.ee/2021/01/15/video-aina-rohkem-loomi-jouab-tallinna-loomade-varjupaigast-tagasi-koju/. [Kasutatud 18 04 2022].

[2] Riigi Teataja, „Tallinna linna 2022. aasta eelarve,“ 28 12 2021. [Võrgumaterjal]. Available: https://www.riigiteataja.ee/akt/428122021027. [Kasutatud 18 04 2022].

[3] Riigi teataja, „Koerte ja kasside pidamise eeskiri,“ 19 12 2013. [Võrgumaterjal]. Available: https://www.riigiteataja.ee/akt/422122013011. [Kasutatud 18 04 2022].

[4] „Lemmikloomade kiibistamine,“ Tallinna linnavalitsus, 25 10 2018. [Võrgumaterjal]. Available: https://www.tallinn.ee/est/lemmikloom/Kiibistamine. [Kasutatud 21 04 2022].

[5] L. Moosaar, „Kiibista ja registreeri, et lemmik jõuaks tagasi koju,“ *Tapa valla ajaleht Sõnumed,* p. 4, 22 09 2016.

[6] TIOBE, „TIOBE Programming Community Index Definition,“ TIOBE, [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/programming-languages-definition/. [Kasutatud 18 04 2022].

[7] J. Gallagher, „Career Karma,“ 29 12 2020. [Võrgumaterjal]. Available: https://careerkarma.com/blog/how-long-does-it-take-to-learn-react-js/. [Kasutatud 18 04 2022].

[8] J. Clark, „Top 10 backend frameworks – Which is the best?,“ Back4App, [Võrgumaterjal]. Available: https://blog.back4app.com/backend-frameworks/. [Kasutatud 18 04 2022].

[9] „Most Popular Backend Frameworks – 2012/2022,“ Statistics and Data, [Võrgumaterjal]. Available: https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2022/. [Kasutatud 18 04 2022].

[10] „2021 Developer Survey,“ Stack, [Võrgumaterjal]. Available: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe. [Kasutatud 19 04 2022].

[11] S. CM, „What Are The Benefits Of Using Express.Js For Backend Development,“ Techomoro, 20 10 2019. [Võrgumaterjal]. Available:

https://www.techomoro.com/what-are-the-benefits-of-using-express-js-for-backend-development/. [Kasutatud 19 04 2022].

[12] M. Demchenko, „Review of Node.JS: Pros and Cons,“ ncube, 02 11 2020. [Võrgumaterjal]. Available: https://ncube.com/blog/review-of-node-js-pros-and-cons. [Kasutatud 19 04 2022].

[13] „LINQ overview,“ Microsoft, 04 02 2022. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/linq/. [Kasutatud 04 19 2022].

[14] „What is .NET? Introduction and overview,“ Microsoft, 11 03 2022. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/core/introduction. [Kasutatud 19 04 2022].

[15] „Node.js vs .NET Core the winner?,“ Medium, 09 06 2021. [Võrgumaterjal]. Available: https://levelup.gitconnected.com/node-js-vs-net-core-the-winner-5ba06efb4c35. [Kasutatud 19 04 2022].

[16] H. K. Dhalla, „A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core,“ *Journal of Physics: Conference Series,* kd. 1933, 2021.

[17] O. Glib, „Most Popular Backend Frameworks in 2021-2022 [Pros and Cons, What to Choose],“ Acropolium, 03 09 2022. [Võrgumaterjal]. Available: https://acropolium.com/blog/most-popular-backend-frameworks-in-2021-2022-pros-and-cons-what-to-choose/. [Kasutatud 19 04 2022].

[18] „Asynchronous programming patterns,“ Microsoft, 11 03 2022. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/. [Kasutatud 19 04 2022].

[19] S. Jain, „Here are Advantages and Disadvantages of Using .NET,“ Medium, 04 02 2020. [Võrgumaterjal]. Available: https://graffersid-sidharth.medium.com/here-are-advantages-and-disadvantages-of-using-net-252ee58590e4. [Kasutatud 19 04 2022].

[20] „Most used programming languages among developers worldwide, as of 2021,“ Statista, 08 2021. [Võrgumaterjal]. Available: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. [Kasutatud 19 04 2022].

[21] „Pros and Cons of Using Spring Boot,“ Scand, 26 06 2020. [Võrgumaterjal]. Available: https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/. [Kasutatud 19 04 2022].

[22] „What is NoSQL?,“ Amazon, [Võrgumaterjal]. Available: https://aws.amazon.com/nosql/. [Kasutatud 19 04 2022].

[23] „Database Providers,“ Microsoft, 09 03 2022. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli. [Kasutatud 19 04 2022].

[24] „What is Entity Framework?,“ Entity Framework Tutorial, [Võrgumaterjal]. Available: https://www.entityframeworktutorial.net/what-is-entityframework.aspx. [Kasutatud 19 04 2022].

[25] „The Most Popular Databases – 2006/2021 – Update May 2021,“ Statistics & Data, 05 2021. [Võrgumaterjal]. Available: https://statisticsanddata.org/data/the-most-popular-databases-2006-2021/. [Kasutatud 20 04 2022].

[26] „Front-end Frameworks,“ State of JavaScript, [Võrgumaterjal]. Available: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/. [Kasutatud 20 04 2022].

[27] D. Gavigan, „The History of Angular,“ Medium, 04 04 2018. [Võrgumaterjal]. Available: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7. [Kasutatud 20 04 2022].

[28] „Introduction to Angular concepts,“ Angular, 28 02 2022. [Võrgumaterjal]. Available: https://angular.io/guide/architecture. [Kasutatud 20 04 2022].

[29] A. Gazta, „Understanding the pros and cons of Angular Development,“ GreyCampus, 12 10 2021. [Võrgumaterjal]. Available: https://www.greycampus.com/blog/programming/good-and-bad-of-angular-development. [Kasutatud 20 04 2022].

[30] Siddharth, „Angular: Pros and Cons,“ Dev.to, 01 04 2021. [Võrgumaterjal]. Available: https://dev.to/siddharthshyniben/angular-pros-and-cons-m9l. [Kasutatud 20 04 2022].

[31] „Comparison with Other Frameworks,“ Vue.js, [Võrgumaterjal]. Available: https://v2.vuejs.org/v2/guide/comparison.html?redirect=true. [Kasutatud 20 04 2022].

[32] R. Patel, „What is Vue.js? Vue.js Advantages And Disadvantages,“ Space-O Technologies, 25 08 2021. [Võrgumaterjal]. Available: https://www.spaceo.ca/blog/vue-js-pros-and-cons/. [Kasutatud 20 04 2022].

[33] H. V. Vinuta Hutagikar, „Analysis of Front-end Frameworks for Web Applications,“ *International Research Journal of Engineering and Technology (IRJET),* kd. 7, nr 4, pp. 3317-3320, 2020.

[34] „ReactJS - Overview,“ TutorialsPoint, [Võrgumaterjal]. Available: https://www.tutorialspoint.com/reactjs/reactjs_overview.htm. [Kasutatud 20 04 2022].

[35] S. Barot, „React Advantages and Disadvantages - Why ReactJS,“ Aglowid, 14 03 2022. [Võrgumaterjal]. Available: https://aglowiditsolutions.com/blog/pros-and-cons-of-reactjs/. [Kasutatud 20 04 2022].

[36] W. Baranowski, „React JS: Advantages and Disadvantages,“ Massive Pixel Creation, 10 04 2021. [Võrgumaterjal]. Available: https://massivepixel.io/blog/react-advantages-disadvantages/. [Kasutatud 20 04 2022].

[37] Eugene, „Vue.js vs React: Comparison of Two Most Popular JS Frameworks,“ Codica, 10 11 2021. [Võrgumaterjal]. Available: https://www.codica.com/blog/react-vs-vue/. [Kasutatud 21 04 2022].

[38] N. Sakovich, „Cross-Platform Mobile Development: Five Best Frameworks,“ SaM Solutions, 25 10 2019. [Võrgumaterjal]. Available: https://www.sam-solutions.com/blog/cross-platform-mobile-development/. [Kasutatud 21 04 2022].

[39] M. Santoni, „Progressive Web Apps browser support & compatibility,“ Good Barber, 12 08 2021. [Võrgumaterjal]. Available: https://www.goodbarber.com/blog/progressive-web-apps-browser-support-compatibility-a883/. [Kasutatud 21 04 2022].

[40] „Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021,“ Statista, 07 2021. [Võrgumaterjal]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. [Kasutatud 21 04 2022].

[41] S. Martin, „7 Popular Cross-Platform App Development Tools That Will Rule in 2021,“ Medium, 14 04 2020. [Võrgumaterjal]. Available: https://medium.datadriveninvestor.com/7-popular-cross-platform-app-development-tools-that-will-rule-in-2020-349c80fb51. [Kasutatud 21 04 2022].

[42] „Cross-platform mobile development 2022: trends and frameworks,“ MerixStudio, 14 02 2022. [Võrgumaterjal]. Available: https://www.merixstudio.com/blog/cross-platform-mobile-development/. [Kasutatud 21 04 2022].

[43] „Flutter on Desktop,“ Flutter, [Võrgumaterjal]. Available: https://flutter.dev/multi-platform/desktop. [Kasutatud 21 04 2022].

[44] M. Berka, „Flutter Pros & Cons – Should You Use It In Your Project?,“ Invo, 2020. [Võrgumaterjal]. Available: https://invotech.co/blog/flutter-pros-cons-should-you-use-it-in-your-project/. [Kasutatud 21 04 2022].

[45] K. Shah, „Advantages and Disadvantages of React Native Development in 2022,“ Third Rock Techkno, 05 20 2021. [Võrgumaterjal]. Available: https://www.thirdrocktechkno.com/blog/pros-and-cons-of-react-native-development-in-2021/. [Kasutatud 21 04 2022].

[46] „Pros and Cons Xamarin Development: A Detailed Analysis of the Frameworks,“ Mindpool Technologies, 07 04 2022. [Võrgumaterjal]. Available: https://mindpooltech.com/pros-and-cons-xamarin-development/. [Kasutatud 21 04 2022].

[47] J. Singh, „Unit of Work in Repository Pattern,“ C# Corner, 30 11 2018. [Võrgumaterjal]. Available: https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/. [Kasutatud 22 04 2022].

[48] „Connect to local web services from iOS simulators and Android emulators,“ Microsoft, 12 07 2021. [Võrgumaterjal]. Available: https://docs.microsoft.com/et-ee/xamarin/cross-platform/deploy-test/connect-to-local-web-services#android-network-security-configuration. [Kasutatud 24 04 2022].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Mihkel Heinmaa

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebining mobiilirakendus kadunud ja leitud loomade info haldamiseks“, mille juhendaja on Meelis Antoi.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

25.04.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Seonduvate materjalide veebilingid

Projekti giti repositooriumi võib leida siit: https://gitlab.cs.ttu.ee/mihkel.heinmaa/petapp/

Rakendusega on võimalik tutvuda aadressil https://petapp.azurewebsites.net/ (vähemalt lõputööde kaitsmisperioodi lõpuni).