TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Robert Laursoo, 193798IADB

Veebirakenduse prototüübi loomine rehvitöökodade reklaamimiseks ja leidmiseks

Bakalaureusetöö

Juhendaja: Meelis Antoi

Magistrikraad

Tallinn 2023

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: Robert Laursoo

03.01.2023

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua veebirakenduse prototüüp rehvitöökodade leidmiseks ja reklaamimiseks, mis aitaks teenust vajaval inimesel leida sobiv teenusepakkuja. Hetkel puudub hea ülevaade Eesti rehvitöökodadest, nende asukohtadest ja osutatavatest teenustest. Sellest tulenevalt on spetsiifilist teenust vajaval inimesel keerukas leida sobivat teenusepakkujat.

Rakenduse arendamisel on silmas peetud projekti aegpüsivust ja paindlikkust edasisel arendamisel. Rakenduse kasutaja poolel on fookus võimalikult lihtsal teekonnal sobiva teenuseosutaja leidmiseks läbi kasutajasõbraliku otsingufiltri.

Arendusprotsessi käigus luuakse veebirakenduse prototüüp, mis võimaldab ettevõtjal luua enda töökoda ja teenuseid tutvustav profiil ning kliendil leida asukoha ja osutatavate teenuste põhjal endale sobiv teenusepakkuja. Rakenduse äriloogika põhikomponendiks ja ühtlasi kõige keerukamaks osaks on töökodade tõhusa ja skaleeruva filtreerimise võimaluse loomine.

Töö keskendub olemasolevate alternatiivide tugevusi koondava ning kaardistatud puuduseid katva IT-lahenduse loomisele, kasutades ajakohaseid tehnilisi vahendeid ning arenduspraktikaid. Töös määratletakse nõuded kavandatavale rakendusele, selgitatakse olemasolevate lahenduste puuduseid, põhjendatakse tehniliste lahenduste valikuid ning antakse ülevaade arendusprotessist. Töö tulemuseks on toimiv veebirakenduse prototüüp.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 6 peatükki, 7 joonist, 8 tabelit.

Abstract

Development of Web Application Prototype for
Advertising and Finding Tire Workshops

The purpose of the thesis is to create a web application prototype for advertising and finding tire workshops. Currently, an overview of workshops in Estonia, their locations and their services is not available. Therefore finding a suitable service provider is complicated.

Development of the application aims for the project to be time-proof and convenient to develop in the future. On the user side, the emphasis is on a simple user experience and a convenient search filter for finding a service provider.

In the development process, a web application prototype is created that allows the entrepreneur to create a business profile and the customer to find a suitable service provider based on the location and the services. The main component of the prototype is a scalable and convenient search filter.

The emphasis of the thesis is on creating an IT solution that combines the strengths of the existing alternatives and covers the identified shortcomings, using up-to-date technical tools and development practices. The work defines the requirements for the planned application, explains the shortcomings of the existing solutions, justifies the choices of technical solutions and provides an overview of the development process. The result of the work is a functional web application prototype.

The thesis is in Estonian and contains 37 pages of text, 6 chapters, 7 figures, 8 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
API token	API pääse, rakendusepääse, ühene identifikaator juurdepääsu taotleva rakenduse autentimiseks, parooli analoog
artisan	Laravel raamistiku käsurealiides (CLI), mille eesmärgiks on lihtsustada rakenduse arendamist ja kasutamist.
CI	*Continuous Intergration* – pidev integratsioon
CLI	*Command-line interface* – käsurea liides
CRUD	*Create, Read, Update, Delete* – akronüüm, mis tähistab viise, kuidas andmeid käidelda (loo, loe, uuenda, kustuta).
DBMS	*Database Management System*, andmebaasisüsteem
ERD	*Entity Relationship Diagram* – andmebaasi disaini struktuuri kujutav diagramm, olemi-suhte diagramm
Eesrakendus	Klientrakendus, veebirakenduse (programmi) osa, mida kasutaja näeb, võimaldab rakendust kasutada ja ligipääsu andmetele
Git	Versioonihaldustarkvara
HTTP	*HyperText Transfer Protocol*, protokoll teabe edastamiseks arvutivõrkudes
Klientrakendus	Veebirakenduse (programmi) osa, mida kasutaja näeb, võimaldab rakendust kasutada.
JSON	*JavaScript Object Notation*, andmevahetusvorming
MoSCoW	Nõuete prioriseerimise meetod
MVC	*Model-view-controller* ehk mudel-vaade-kontroller – arhitektuurimuster veebiteenuste kujundamiseks, mis jaotab rakenduse kolmeks loogiliseks osaks
NPM	*Node Package Manager*, keskkond JavaScript pakettidele
ORM	*Object-relational mapper*, võimaldab olemeid käsitleda objektorienteeritud keeltes
REST	*Representational State Transfer* – veebiteenusega suhtlemise liidese arhitektuur
SEO	*Search Engine Optimization*, rakenduse optimeerimine otsimootorile, indekseeritavuse ja leitavuse parandamiseks
SQL	*Structured Query Language*, struktureeritud päringute keel relatsiooniliste andmebaaside halduseks ja kasutamiseks
Tagarakendus	Veebirakenduse (programmi) osa, mida kasutaja ei näe, võimaldab ligipääsu andmetele (ilma kasutajaliideseta) liidese vahendusel

VPS	Virtuaalne privaatserver. Virtuaalmasin, mis kasutab osa füüsilise serveri ressurssidest ja mille puhul saab valida meelepärase operatsioonisüsteemi ning seda ise juurkasutaja õigustes hallata, uuendada ja turvata.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva töö raames luuakse veebirakenduse prototüüp rehvitöökodade leidmiseks ja reklaamimiseks.

Eestis on hinnanguliselt tuhandeid rehvitöökodasid, kuid vajalikul hetkel sobiva teenuseosutaja leidmine on keerukas, kuna puudub hea ülevaade Eesti rehvitöökodadest, nende asukohtadest ja osutatavatest teenustest. Vajadus teenuse järele tekib enamasti ebamugaval ajal, ootamatult ja harva. Nendeks ootamatuteks ebamugavusteks võivad olla esimene lumi (millega kaasnevad töökodades järjekorrad), rehvi purunemine väljaspool tavapärast tööaega või võõras asukohas. Sellisel hetkel oleks abi rakendusest, mis aitaks leida sobiva teenuse pakkuja.

Töö eesmärgiks on luua kasutusvalmis rakenduse prototüüp, mis võimaldaks töökodadel (ettevõtetel) läbi profiili loomise oma teenuseid reklaamida ning aitaks spetsiifilist teenust (rehvivahetus, -hotell, -parandus, -müük) vajaval inimesel sobivat töökoda leida.

Vajadus uudse veebirakenduse järele tulenes olemasolevate alternatiivide sobimatusest nii spetsiifilise teenuse reklaamimiseks kui ka sobiva teenuseosutaja leidmiseks. Töös välja toodud olemasolevad alternatiivid (käsitletud peatükis 2.3) on oma funktsionaalsuselt piiratud, Eestis väikese kasutajaskonnaga või tasulised. Lahendatava probleemi tausta selgitatakse põhjalikumalt peatükis 2.1.

Rakendus teostatakse kohanduva kasutajaliidesega veebirakendusena, kuna see IT-lahendus sobib kõige paremini tõenäolise kasutajateekonnaga, mis algab probleemi tekkimisele järgnevalt Google otsinguga nutiseadmest. Rakenduse moodustavad serveripoolne rakendus (raamistik Laravel) ja klientrakendus (React), mis omavahel suhtlevad üle REST API.

Töö esimeses osas analüüsitakse probleemi, seatakse lõputöö eesmärgid ja skoop ning antakse ülevaade olemasolevatest lahendustest. Töö teises osas analüüsitakse loodavat lahendust, määratakse nõuded rakendusele ning selgitatakse tehnoloogiate valikut. Töö kolmandas osas antakse ülevaade klient- ja serverrakenduse arendamise protsessist ning esinenud keerukustest. Viimases osas kirjeldatakse tehtud töö tulemusi ning edasiarenduse võimalusi.

2 Probleemi taust ja rakenduse eesmärk

Käesolevas peatükis annab töö autor ülevaate lahendatava probleemi taustast, lõputöö metoodikast ning loodava rakenduse eesmärgist. Arvestades püstitatud eesmärke, analüüsitakse olemasolevaid lahendusi, nende tugevusi ja puuduseid ning seatakse tööle kitsendused.

2.1 Probleemi taust

Rehvitöökodade (edaspidi ka lihtsalt „töökoda“) arv Eestis on suur. Tähise „rehvitöökoda“ all peab töö autor silmas nii spetsiaalselt rehvitöödele spetsialiseerunud töökodasid kui ka autoremonditöökodasid üldiselt, mis teiste teenuste hulgas osutavad ka valitud rehvitöid. Selliselt määratletuna võib töökodade arvu hinnata mitmele tuhandele.

Hetkel puudub hea ülevaade Eesti rehvitöökodadest, nende asukohtadest ja osutatavatest teenustest. Sellest tulenevalt on spetsiifilist teenust vajaval inimesel keerukas leida sobivat teenusepakkujat.

Mõned kasutajalood, mis aitavad probleemi illustreerida, on välja toodud alljärgnevalt.

- Rehvide vahetamise hooajal (kevadel ja sügisel) tekivad osades töökodades, mis on tuntumad ja sellest tulenevalt ka populaarsemad, pikad järjekorrad. Järjekorra pikkus võib olla kohati kuni mitu nädalat samas kui mõnes teises (vähem tuntud) töökojas, mis osutab vajalikku teenust (rehvihotell, rehvivahetus) saaks rehvid koheselt vahetatud.
- Rehv läheb katki tundmatud asukohas ning oleks vaja leida asukoha lähiümbrusest sobivale teenusele (rehvi parandus, müük) pakkuja.
- Rehv läheb katki ebamugaval ajal (reede õhtu, pühapäev) ning oleks vaja leida asukoha lähiümbrusest sobivale teenusele (rehvi parandus, müük) pakkuja.

Käesolevas lõputöö eesmärgiks on luua veebirakenduse prototüüp, mis võimaldaks:

- saada hea ülevaate Eesti töökodadest;
- teenust osutavatel ettevõtetel enda teenust reklaamida (töökoja profiili loomine rakenduses);
- teenust vajavatel inimestel leida (rakenduse abil) endale sobiv teenuseosutaja.

Prototüübi all mõeldakse antud töö kontekstis kasutusvalmis veebirakendust, mis vastaks analüüsis määratletud kõige kõrgema prioriteediga nõuetele.

2.2 Metoodika

Esmalt selgitatakse eksisteerivat probleemi ning seatakse töö jaoks täpsemad eesmärgid. Tuginedes töö eesmärkidele luuakse ülevaade olemasolevatest sarnastest lahendustest, nende headest külgedest ning puudustest. Seejärel pakutakse välja probleemi lahendamiseks (tõenäoliselt sobiv) IT-lahendus, mille prototüübi loomine aitaks (suhteliselt kulutõhusalt) luua selgust, kas pakutud suund on probleemi lahendamiseks sobiv.

Seejärel analüüsitakse planeeritavat lahendust, kaardistatakse funktsionaalsed ja mittefunktsionaalsed nõuded rakendusele ning pannakse need tähtsuse järjekorda (prioritiseeritakse). Nõuete määramise aluseks on kavandatava rakenduse äriloogika, mis ühtlasi arvestab olemasolevate lahenduste tugevuste ja puudustega. Samuti antakse selgitus rakenduse arenduseks valitud tehnoloogiatele klientrakenduse, serveri-poolse rakenduse, nende omavahelise liidese ja andmebaasi disaini osas.

Analüüsile järgneb rakenduse arendust puudutav osa, kus antakse ülevaade rakenduse erinevate osade arendamise protsessist ning rakenduse testimisest.

Töö viimases osas hinnatakse valminud rakenduse vastavust seatud eesmärkidele, töö analüüsi osas kirjeldatud nõuetele ning tuuakse välja edasiarenduse võimalused.

2.3 Olemasolevad lahendused

Käesolevas peatükis antakse ülevaade olemasolevatest lahendustest, millel on sarnasusi loodava veebirakendusega. Loodava rakenduse puhul on eesmärgiks koondada olemasolevate lahenduste head omadused ja samas katta puudused.

2.3.1 Auto24 kataloog

Käesoleva lõputöö idee üheks ajendiks oli autokaupade ja autodega seotud teenuseid osutavate ettevõtete kataloog Auto24 portaali alamlehel „Firmad ja teenused“.

Alamlehe näol on tegemist võrdlemisi vanamoodsa ilme ja kasutusmugavusega veebikataloogiga. Ettevõtte lisamine Auto24 kataloogi on tasuline (hind 68.96 eurot

aastas, seisuga 11.11.2022). See võib olla põhjuseks, miks „Rehvid ja veljed“ rubriigis on vaid 50 ettevõtet, kuigi rehvitöökodasid on Eestis hinnanguliselt tuhandeid. Nõuetele vastava ettevõtte kataloogist leidmine on keerukas, sest puudub filtreerimine spetsiifilise teenuse järgi ning asukohafilter on olemas vaid linna täpsusega.

Kuna Auto24.ee domeeni autoriteetsus (*domain authority*) on suur, siis on võimalik, et SEO seisukohast võib olla mõistlik end sinna (viite saamiseks) ka raha eest lisada.

2.3.2 Google Business Profile

Google Business Profile on (kliendi vaatest) väga hea kasutusmugavusega, kuna see on integreeritud nii Google Search kui ka Google Maps teenustesse. Lahenduse puudusena võib välja tuua asjaolu, et rakendus ei sisalda kõiki ettevõtteid, vaid ainult neid, mis on end sinna ise lisanud või mille info Google on veebist leidnud (ja osanud selle tegevusvaldkonnaga seostada).

Enda ettevõtte äriprofiili loomine ja haldamine eeldab sotsiaalmeedia haldamise oskuseid ja aega. Spetsiifilisemate teenuste ja kategooriate valik on rakenduses piiratud nii ettevõtte leidmise kui lisamise poolel. Lisaks kerkib rakenduses aeg-ajalt esile arvustuste modereerimise võimaluse puudumise ja pahatahtlike negatiivsete arvustuste probleem.

2.3.3 Foursquare City Guide

Foursquare City Guide on väga hea kasutusmugavusega rakendus erinevate teenuste leidmiseks. Kõigile rakenduse kasutajatele on antud võimalus panustada sisu toimetamisse läbi muudatuste soovitamise. Enim väärtust loova sisendi ettevõtte profiilile (näiteks reiting ja hinnatase) annavadki Fourquare’i puhul just kliendid. Rakendusel on Eestis vähe kasutajaid ning sellest tulenevalt ei anna enda ettevõtte reklaamimine selles suurt ärilist efekti. Töö kirjutamise ajal on märksõnaga „tire“ Tallinnas leitavad 5 rehvitöökoda.

Kuna rakendus ei ole valdkonnaspetsiifiline ja püüab katta kõiki võimalikke otsitavaid valdkondi, siis väga konkreetse teenuse (näiteks rehvihotell, rehviparandus, müük) leidmine on keerukas.

2.4 Töö kitsendused

Käesoleva lõputöö raames loodava prototüübi skoobi määrab nõuete kirjeldus. Kogutud nõuded prioritiseeritakse MoSCOW meetodil [1] ning lõputöö praktilise osana teostatakse esmajärjekorras vaid kõige kõrgema tähtsusega (*Must have*, M) funktsionaalsused. *Must have* prioriteediga nõudeid iseloomustab see, et ilma nende täitmiseta ei ole võimalik kavandatavat tarkvaralahendust kasutada.

Kõiki mõeldavaid funktsionaalsuseid lõputöö raames arendada ei ole võimalik, sest töö skoop läheks liiga laiaks ja tehnoloogiad (ajaraami arvestades) liiga keerukaks.

3 Loodava veebirakenduse analüüs

Analüüsi esimeses pooles selgitab autor nõuded kavandatavale rakendusele, prioritiseerib nõuded MoSCoW meetodil ning toob välja võimalikud tulevikus teostatavad nõuded. Analüüsi teises pooles annab autor põhjendused lahenduse teostamiseks valitud tehnoloogiatele.

3.1 Nõuete määramine

Rakendusele seatud nõuded põhinevad kasutajalugudel. Kasutajalood on grupeeritud eepikute ja kasutajarollide (persona) järgi loogilisteks tööühikuteks, mis kirjeldavad kuidas arenduse käigus tehtav iga väike ühik tööd loob väärtust kasutajale [2].

Kasutajarollideks käesolevas töös on:

- teenusepakkujat otsiv registreerimata kasutaja (edaspidi “klient”);
- teenuseid pakkuv registreerunud kasutaja (edaspidi „teenusepakkuja“);
- veebirakendust haldav administraator.

Kõige tavapärasemaks nõuete prioritiseerimise viisiks peetakse nõuetele arvulise hinde andmist tulenevalt nende olulisusest. Üheks selliseks tähtsuse järjekorda seadmise meetodiks on MoSCoW, mille põhjal jagatakse kavandatava lahenduse I etapi nõuded nelja gruppi vastavalt sellele, kui oluline on nende nõuete teostamine [1], [3]. Gruppide iseloomustus on järgmine:

- ***Must have*** (M) (peab olema) – nõuded, ilma milleta ei ole võimalik kavandatavat lahendust kasutada.
- ***Should have*** (S) (peaks olema) – nõuded, mis on kavandatava lahenduse kontekstis olulised, kuid milleta on saab kavandatavat lahendust kasutada. S prioriteediga nõuded võib teostada näiteks järgmises iteratsioonis.
- ***Could have*** (C) (võiks olla) – nõuded, mis ei kuulu kavandatava lahenduse põhifunktsionaalsusesse ning mille puudumine ei mõjuta oluliselt lahendust.
- ***Won’t have*** (W) (ei pea olema) – nõuded, mis ei ole kokkulepitud ajaraamis prioriteediks ning neid üldiselt ei realiseerita, kuid välistatud ei ole nende prioriteedi hilisem muutmine.

Klassifitseerimise põhjal selgub iga nõude prioriteetsus. Olulisuse hindamisel peab autor silmas nõude olulisust rakenduse kasutajale ja arendamisele kuluvat ressurssi (lõputöö konteksti tööaeg).

Must have kategoorias olevaid nõudeid on vajalik rakendada esimeses etapis, et rakendust oleks võimalik kasutada (esialgu piiratud funktsionaalsuses). *Should have* kategooria nõuded teostatakse järgmistes iteratsioonides. Nõudeid võib edaspidi uuesti tähtsuse järjekorda seada (ehk prioritiseerida), mille tulemusena võib nõude prioriteet muutuda.

3.1.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded kirjeldavad seda, mida süsteem (rakendus) tegema peab. Järgnevalt on kirjeldatud autori poolt oluliseks peetud ehk kõrgeima prioriteediga (*Must have*) funktsionaalsed nõuded.

Nõuded on vormistatud eepikutena ja vastavast eepikust tulenevate kasutajalugudena (tabelid 1, 2 ja 3) ehk väärtust loovate (töö)ühikutena, mille keerukus on piisavalt väike [2].

Kliendi (*client*) funktsionaalsed nõuded:

Eepik: „Ülevaade teenusepakkujatest“

Tabel 1. Eepiku „Ülevaade teenusepakkujatest“ kasutajalood.

ID	Roll	Tegevus	Eesmärk
FN-C-1	Kliendina	soovin saada ülevaate teenusepakkujatest (loeteluna)	et leida endale sobiv teenusepakkuja
FN-C-1.1	Kliendina	soovin filtreerida loetelust töökojad, mis asuvad soovitud piirkonnas (sh kõigis alampiirkondades)	et leida teenusepakkuja endale sobivad asukohas
FN-C-1.2	Kliendina	Soovin filtreerida loetelust töökojad, mis osutavad vajalikku teenust	et leida vajalikule teenusele pakkuja

Teenusepakkuja (*user*) funktsionaalsed nõuded:

Eepik: „Kasutaja profiili haldus“

Tabel 2. Eepiku „Kasutaja profiili haldus“ kasutajalood.

ID	Roll	Tegevus	Eesmärk
FN-U-1	Teenuse-pakkujana	soovin reklaamida oma rehvitöökoda	et saada rohkem kliente ja ärikontakte
FN-U-1.1	Teenuse-pakkujana	soovin end rakenduses kasutajaks registreerida	et saada ligipääs rakenduse funktsionaalsustele
FN-U-1.2		soovin luua rakenduses töökoja profiili	et reklaamida enda rehvitöökoda
FN-U-1.3		soovin muuta töökoja profiili avalikuks või privaatseks (peidetuks)	et vajadusel piirata ligipääsu enda töökoda puudutavale infole (näiteks info muutmise ajaks)
FN-U-1.4		soovin muuta enda kontoga seotud töökoja profiilil olevat infot	et tagada info korrektsus ning profiili vastavus ärilistele eesmärkidele
FN-U-1.5		soovin kustutada enda töökoja profiili	et töökodade haldamise vaates ei esineks mittevajalikke kandeid

Administraatori (*admin*) funktsionaalsed nõuded:

Eepik: „Rakenduse administreerimine kasutajaliideses“

Tabel 3. Eepiku „Rakenduse administreerimine kasutajaliideses“ kasutajalood.

ID	Roll	Tegevus	Eesmärk
FN-A-1	Admin rollis kasutajana	soovin muuta teenusepakkuja profiilis sisalduvat infot	et saaksin leheküljel olevat sisu hoida korrektsena ning vajadusel parandada info sisestamisel tehtud vigu
FN-A-1.1	Admin rollis kasutajana	soovin luua töökodade profiile	et külastajatel oleks võimalik leida töökodasid, mida ettevõtte esindaja ei ole veel lehele ise lisanud
FN-A-1.2		soovin muuta töökodade profiilides sisalduvat infot (isegi kui profiil ei ole seotud minu kasutajakontoga)	et saaksin leheküljel olevat sisu hoida korrektsena ning vajadusel parandada info sisestamisel tehtud vigu
FN-A-1.2		soovin muuta profiili omanikku	et töökoja esindajal oleks võimalik profiil enda haldusesse võtta
FN-A-1.3		soovin hallata teenuste, kontakti tüüpide ja haldusüksuste loetelusid	et saaksin lisada puuduolevaid väärtuseid, kandeid täiendada või muuta ja vajadusel parandada sisestamisel tehtud vigu
FN-A-1.4		soovin muuta profiili avalikuks või privaatseks	et hoida rakenduse sisu asjakohasena

3.1.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded kirjeldavad, kuidas süsteem (rakendus) toimib või käitub. Käesolevas punktis on välja toodud *Must have* (M) prioriteediga mittefunktsionaalsed nõuded. Nõuete aluseks on vajadus tagada rakenduse kasutatavus erinevates võimalikes olukordades.

Eepik: „Rakenduse kasutatavus“

Tabel 4. Eepiku „Rakenduse kasutatavus“ kasutajalood.

ID	Nõue
MF-1	Rakendust peab saama kasutada väikeste ekraanidega seadmetel (nutitelefon)
MF-2	Ligipääs rakenduse funktsionaalsustele toimib vastavuses juurdepääsupiirangutega
MF-3	Rakendus peab olema ligipääsetav enamlevinud veebibrauseritest

3.1.3 Tulevikus loodavate funktsionaalsuste nõuded

Ajalise piirangu ja kohatise tehnilise keerukuse tõttu jäävad madalama prioriteediga (*Should have, Could have, Won't have*) käesoleva lõputöö skoobist välja, kuid rakenduse potentsiaali tutvustamise huvides on järgnevalt mõned siiski välja toodud.

Kliendi funktsionaalsed nõuded:

- Kliendina soovin töökodasid sorteerida hinnataseme järgi (S)
- Kliendina soovin jätta töökoja profiilile tagasisidet (arvustused) ilma kasutajakontot loomata (anonüümsete kommentaaridega seotud probleemide vältimiseks tugeval autentimisel põhinev isikutuvastuse lisamine lehele) (C)
- Kliendina soovin filtreerida hetkel avatud olevaid töökodasid (lahtiolekuaegade lisamise teostamine) (S)
- Kliendina soovin saada soovitusi sobivaima teenusepakkuja leidmiseks (parim hinne, parim hind, hetkel avatud jne) (C)
- Kliendina soovin sorteerida (järjestada) töökodasid (nime järgi, kauguse järgi, uudsuse, hinna või hinnangu järgi) (C)
- Kliendina soovin ülevaadet viimati lisatud või uuendatud töökodadest, hinnangutest, teadaannetest (C)
- Kliendina soovin töökoja profiilil näha, millal infot on viimati uuendatud ning saada teavitatud (näiteks hoiatusega), kui info võib olla aegunud (C)

- Kliendina soovin teha muudatusettepanekuid profiilile, kui olen teadlik, et info ei ole korrektne (S)

Teenusepakkuja funktsionaalsed nõuded:

- Teenusepakkujana soovin, et saaksin enne konto loomist (registreerumist) tutvuda kasutajatingimustega (S)
- Teenusepakkujana soovin, et saaksin enda profiili teiste hulgast esile tõsta (näiteks tasulise teenusena) (C)
- Teenusepakkujana soovin, et saaksin muuta enda profiilide URLi (aadressi), et saaksin viidet lihtsasti jagada (C)
- Teenusepakkujana soovin lisada enda töökoja lehele profiilipildi, et potentsiaalsed kliendid näeksid, missugune töökoda välja näeb ja oskaksid seda kergema vaevaga leida (S)

Administraatori funktsionaalsed nõuded:

- Administraatorina soovin ülevaadet kasutajate aktiivsusest (C)
- Administraatorina soovin hallata kasutajakontosid, et saaksin (mugavalt) eemaldada inaktiivsed või mittevajalikud kasutajakontod koos nende loodud sisuga (S)

Mittefunktsionaalsed nõuded:

- Kasutajale tuleb päringu ebaõnnestumisel kasutajaliideses kuvada asjakohane ja mõistetav veateade (S)
- Rakenduse avalik sisu peaks olema otsimootoritele indekseeritav ja vastama tehnilise SEO headele tavadele (S)

3.2 Kasutatavad tehnoloogiad

Käesolevas peatükis annab töö autor ülevaate rakenduse prototüübi loomiseks valitud tehnoloogiatest.

Tehnoloogiate valiku lähtekohaks on autori eelistus ehitada rakendus hajusa süsteemina (*distributed system*). See tähendab, et rakenduse komponendid – antud juhul klientrakendus, tagarakendus ja andmebaas – eksisteerivad eraldiseisvalt ja teineteisest

sõltumatult [4], [5]. Sellise lahenduse realiseerimiseks on sobiv rakendusliidese (näiteks RESTful API) kasutamisel põhinev arhitektuur.

Alternatiivseks valikuks oleks MVC arhitektuurimuster, kus rakenduses teostatud vaatekiht suhtleb andmekihiga läbi kontrollerite [6].

3.2.1 Serveripoolse programmeerimiskeele valik

Töö autor valis serveripoolseks programmeerimiskeeleks PHP (versioon 8). PHP on 1994. aastal algselt veebilehtede programmeerimiseks loodud keel [7].

Tänaseks on PHP keelena oluliselt edasi arenenud (jõudnud versioonini 8), väga populaarne ja sobilik ka suuremate veebirakenduste arenduseks. W3Techs andmetel on enam kui 77% veebilehtede serveripoolne rakendus kirjutatud PHP keeles [8].

Erinevalt teistest valikus olnud keeltest (C#, Java) on PHP interpreteeritav keel, mis tähendab, et kõrgtaseme koodi ei teisendata täitmiseks masinkoodi (kompileerita), vaid programmi loeb ja täidab interpretaator. Interpreteeritavates keeltes programmid on aeglasemad aga neid on kergem siluda (*debug*) [9].

Tallinna Tehnikaülikooli lõputööde hulgas on PHP programmeerimiskeelt võrreldes Java ja C#'ga kasutatud vähem. Samuti ei kasutatud IT-arenduse õppekava kohustuslikes ainetes (C# ja Javaga võrreldavas mahus) PHP raamistikke, millest tulenevalt on autoril selle kasutamine huvitav ja võimaldab õppida midagi uut.

Kuna autor kasutab ka enda teiste arendusprojektide puhul PHP keelt, siis on lõputöö kirjutamine võimalus end selles keeles arendada.

PHP keeles kirjutatud rakenduse paigaldamiseks (*deployment*) on võimalik kasutada jagatud virtuaalserveri teenust (*shared hosting*), mis teeb serveripoolse rakenduse majutuse soodsaks (eriti arvestades paindlikku ligipääsu arvutusressursile) ning rahalise kulu prognoositavaks. Samuti vastutab sellise teenuse puhul serveri turvalisuse ja käideldavuse eest serveriteenuse pakkuja [10], mis võimaldab autoril keskenduda rakenduse arendamisele.

3.2.2 Raamistiku valik

Serveripoolse rakenduse arendusraamistikuks valiti Laravel (versioon 9), mis on väga levinud (tabel 5) ning mõistliku õpikõveraga raamistik veebirakenduse arendamiseks programmeerimiskeeles PHP. Laravel sobib kasutamiseks täispinu raamistikuna (sisaldab mallide teeki Blade) kui ka lihtsalt API liidesel põhineva tagarakenduse ehitamiseks [11].

JetBrains DevEco 2021 arendajate küsitluses [12] selgus, et 67% PHP arendajatest kasutab regulaarselt Laravel raamistikku, millele järgnes Symfony (24%) raamistik. Sarnane on kahe kõige populaarsema PHP raamistiku kasutajaskonna suhe ka Stack Overflow 2022 arendajate küsitluses [13].

Packagist repositooriumi kaudu raamistike installeerimist puudutavate andmete põhjal on enim kasutatav Laraveli versioon 9 (välja antud 01.2022, kõige uuem versioon) samas kui Symfony puhul on enim kasutatavad versioonid 3 (12.2015) ja 4 (10.2017) ning kõige uuemat versiooni sisuliselt keegi ei kasuta.

Alljärgnevalt on tabelina välja toodud võrdlev ülevaade levinumate PHP raamistike kasutusest seisuga 11.11.22.

Tabel 5. PHP raamistike võrdlus.

PHP raamistik	**DevEco '21[1] [12]**	**Stack-Overflow '22[2] [13]**	**Installeerimisi[3]**	**Viimane põhi-versioon**
laravel/framework	67%	9.45%	5 431 657	01.2022
symfony/symfony	24%	3.58%	576 032	02.2022
codeigniter/framework	9%	-	24 178	03.2015
yiisoft/yii2	6%	-	270 319	12.2013
slim/slim	5%	-	462 931	04.2019
cakephp/cakephp	4%	-	174 644	11.2019

1 Regulaarseid kasutajaid PHP arendajate hulgast

2 Viimase aasta jooksul kasutanud (kõigi veebiraamistike hulgast)

3 Installeerimisi viimase 30 päeva jooksul Packagist repositooriumi andmetel (seisuga 11.11.22)

Sagedamini tarkvaralahendustes esinevate funktsionaalsuste teostamise jaoks leidub Laravelile tõenäoliselt valmis pakett. Aktiivselt arenduses olevate ja suure kasutajaskonnaga avatud lähtekoodiga pakettide kasutamine vähendab (vähese kogemusega arendaja puhul) kehva lahenduse loomise võimalust ja esineda võivaid vigu ning tagab juurdepääsu võimalikele (turva)vigade avastamisele ja kiiretele (turva)paikadele.

Laravel raamistikuga on vaikimisi kaasas paketid laravel/sanctum tokenil põhineva autentimise teostamiseks, laravel/tinker PHP ja Laravel rakenduse kasutamiseks käsureal ning guzzlehttp/guzzle, mis on HTTP klient PHP rakendustele.

Rakendust kavandades uuris autor põhjalikumalt kolme paketti Laravel raamistikus kasutamiseks:

- nuwave/lighthouse (Lighthouse) – GraphQL spetsifikatsioonil põhineva API liidese tegemiseks;
- laravel/passport (Passport) – OAuth2 toega autentimise tegemiseks;
- spatie/laravel-query-builder – API päringute teisendamine andmebaasipäringuteks, sisaldab filtrite ehitamise ja päringu modifitseerimise funktsionaalsust.

Prototüübi loomisel siiski väliseid pakette ei kasutatud, kuid tagarakenduse tulevikukindlust ja võimalikke edasiarendusi silmas pidades on kasulik teada, et valitud raamistikul on ka GraphQL liidese ja OAuth2 toega autentimise võimalused olemas. Filtreerimise funktsionaalsuse ehitamisel võttis autor eeskuju spatie/laravel-query-builder paketist, kuid paketti ennast siiski ei kasutanud. Spatie on Belgia ettevõte, ühtlasi üks suuremaid ja tuntumaid avatud lähtekoodiga pakettide arendaja Laravel ökosüsteemi jaoks.

Kui eelnevalt kirjeldatud arenduskeele ja raamistiku valikule läheneda majandusliku kasu perspektiivist, siis PHP ja Laravel on kõige halvem võimalik kombinatsioon. StackOverflow 2022 arendajate uuringu põhjal [13] saavad kõige vähem palka Laraveli arendajad (veebiraamistike arvestuses) ja PHP arendajad (programmeerimiskeelte arvestuses).

3.2.3 API spetsifikatsiooni valik

Planeeritava rakenduse puhul otsustati, et klientrakendus ja serverrakendus suhtlevad omavahel üle liidese (API). Kaasaegsete veebirakenduste puhul on peamisteks liidese tüüpideks RESTful ja GraphQL API liidesed.

API on nõuete kogum (leping), mis määratleb, kuidas rakenduse osapooled saavad omavahel suhelda ehk pääsevad ligi teineteise ressurssidele [14].

RESTful API on rakendusliidese arhitektuuriline stiil. Iga ressursi jaoks on oma URL, mille tulemusena REST API moodustab sisuliselt URLide loetelu. Ühtlasi kirjeldab iga URL (oma struktuuriga), missuguseid ressursse on võimalik APIst pärida [15]. Kasutusel on peamiselt 4 verbi, mis kirjeldavad tegevust, mida päringuga teha soovitakse. REST API on põhjalikumalt kirjeldatud peatükis 4.2.4.

GraphQL on Facebooki poolt 2012. aastal loodud [16], spetsiaalselt rakendusliideste jaoks mõeldud päringukeel, mille süntaks on inspireeritud JSONist [17] ja rakendusliidese spetsifikatsioon [18], millele vastava liidese puhul toimuvad kõik päringud ühe *endpointi* kaudu. URLide loetelu asemel annab ülevaate ressurssidest *schema* (sisuliselt API dokumentatsioon), mille põhjal päringuid koostatakse [15], [17]. Kasutusel on peamiselt üks verb – POST – ning ühe päringuga on võimalik kombineerida erinevaid ressursse ja määrata täpselt, missuguseid ressursi väljasid ja missugusel kujul vastuses soovitakse. Nii väheneb klientrakenduse poolt tehtavate HTTP päringute arv ning vastuse suurus (ja edastatav andmemaht) [19].

Uurimise tulemusena, võttes arvesse trende, arendusprotsessi keerukust ja arendajate hinnanguid [20], valis töö autor klient- ja tagarakenduse vaheliseks suhtluseks RESTful API liidese.

3.2.4 Klientrakenduse keel

Kõige populaarsem kaasaegsete klientrakenduste programmerimise keel on JavaScript, mis võimaldab teha interaktiivseid kasutajaliideseid. JavaScript olnud StackOverflow iga-aastases arendajate küsitluses 10 aastat järjest kõige populaarsem programmeerimiskeel [13].

Selleks, et arendamisel oleks võimalik kasutada koodiredaktoris staatilist tüübi-kontrolli tuge, kasutab autor TypeScripti. TypeScript on JavaScriptil põhinev tugevalt tüübitud programmeerimiskeel, millele pani aluse Microsoft [21]. TypeScript transpileeritakse hiljem (*build* käigus) tagasi JavaScriptiks.

3.2.5 Klientrakenduse raamistik

Töö autor valis klientrakenduse arendamiseks kõige populaarsema JavaScripti raamistiku [22], milleks on Facebooki loodud React (versioon 18). JetBrains DevOps ’21 küsitluse [23] põhjal kasutab Reacti regulaarselt 49% JavaScripti arendajatest, Vue.js vastavalt 43% (+9% võrreldes eelmise aastaga). StackOverflow 2022 küsitluses [13] oli React raamistikku viimase aasta jooksul kasutanud 43% kõigist arendajatest, Vue3 vastav näitaja oli 19%. Npm (Node Package Manager) paketihalduri avaliku statistika põhjal (seisuga 15.11.22) on vue paketi nädalane allalaadimiste arv 3.9 miljonit, react paketil samas 18.2 miljonit.

Reacti alternatiivina oli valikus ka Vue3, kuid sellega on autoril vähem kogemust ning kuna Vue raamistik on Reactist vähem levinud, siis tõenäoliselt oleks ka töö käigus tekkivatele probleemidele lahenduse leidmine keerukam.

3.2.6 Koodihalduskeskkond

Rakenduse lähtekoodi versioonihalduseks (*version control*) kasutati Giti. Git on kõige enam levinud versioonihaldustarkvara, mis võimaldab koodis tehtud muudatusi säilitada, hallata ning neile ligi pääseda erinevatest arvutitest [24].

Giti repositooriumite majutamise (*hosting*) ja arendusprojektide haldamise teenustest on autoril kogemus GitLab (GitLab Inc), GitHub (Microsoft Corporation) ja Bitbucket (Atlassian) kasutamisel. Kuna Giti puhul tegemist on ennekõike käsureal põhineva tööriistaga, siis lõputöö kontekstis võib kõik eelpool loetletud teenusepakkujaid pidada funktsionaalsuse poolest võrdseteks. Samuti ei rakendu käesoleva lõputöö projekti arendusel ühegi teenuseosutaja puhul tasuta paketi piirangud, mis võiksid otsustavaks saada ärikasutajate ja suuremate arendusmeeskondade puhul. Sellest tulenevalt puuduvad ratsionaalsed argumendid ühe või teise eelistamiseks.

Koodihalduse keskkonnaks valiti GitLab, kuna selles teenuses haldab töö autor ka teisi enda arendusprojekte. Klientrakenduse (React) ja serveripoolse rakenduse (Laravel) lähtekoodi hallati eraldi Giti repositooriumites.

3.3 Andmebaasisüsteemi valik

Autori eelistuseks oli kasutada vabavaralist (ja tasuta) relatsioonilist andmebaasi (täpsemalt DBMSi ehk andmebaasisüsteemi). Selliste piirangute tulemusena jäävad valikusse MySQL (Oracle), MariaDB (Maria DB Foundation) ja PostgreSQL (PostgreSQL Global Development Group). Laravel 9 põhineval serveripoolsel rakendusel on vaikimisi olemas kõigi eelnimetatud andmebaaside tugi [11].

Valituks osutus MariaDB relatsiooniline andmebaas. Otsust kujundasid mitu erinevat tegurit.

- MariaDB on avatud lähtekoodiga ja vabavarana levitatav relatsioonilise andmebaasi süsteem (RDBMS), millele panid aluse MySQLi arendajad ja mida arendab MariaDB Foundation [25]. MariaDB tuntumate kasutajatena on välja toodud Wikipedia, WordPress.com ja Google [25].
- Eesti suurim virtuaalserveri teenusepakkuja Zone Media, kelle teenustega autoril on varasem kogemus ja kuhu serverisse rakendus koos andmebaasiga (võimalusel) paigaldatakse, kasutab MariaDB andmebaasi. Autor soovib oma töös keskenduda rakenduse tarkvara arendusele. Jagatud virtuaalserveri kasutamine rakenduse paigaldamiseks võimaldab vastutuse jaotuse, kus tarkvaraplatvormi (sh andmebaasi) ja operatsioonisüsteemi turvalisuse ja käideldavuse (uuendused, turvalisus, toimepidevus, jõudlus, varukoopiad) eest vastutab serveri teenuse pakkuja [10].
- Kui mingil põhjusel tekib tulevikus vajadus teenuseosutajat vahetada, siis MariaDB andmebaasiserveri leiab täisteenusena (*managed database*) kõigi suuremate teenuseosutajate (DigitalOcean, Azure, AWS jne) hinnakirjast. Samuti on võimalik soovi korral andmebaasiserverit ise seadistada ja hallata.

Andmebaaside paigaldust kirjeldatakse lähemalt peatükis 4.2.3.

4 Veebirakenduse arendus

Veebirakenduse prototüübi arendus jaguneb kolme etappi: andmemudeli väljatöötamine, serveri poolse rakenduse arendamine (ja testimine) ning klientrakenduse arendamine (ja testimine). Järgnevalt antakse ülevaade eelnimetatud arenduse etappidest. Lisaks käsitletakse rakenduse paigaldust ning ligipääsetavust rakenduse osadele.

4.1 Andmebaasi olemi-suhte diagramm

Rakenduse arenduse esimene etapp oli andmebaasiskeemi (relatsioonilise andmemudeli) väljatöötamine.

Andmemudel (joonis 1) peaks vastama andmete hoiustamise headele tavadele, tagatud peaks olema andmete terviklikkus, ühekordsus ja ligipääsetavus [26]. Autori eesmärgiks oli tagada loodava andmemudeli aegpüsivus, mis tähendab, et mudeli muudatused seisnevad ainult uute komponentide ja seoste lisandumises mudelisse [27]. Tabelite ja olemi atribuutide nimetamisel (notatsioon, grammatika) lähtuti Laraveli raamistiku tavadest [28], [29].

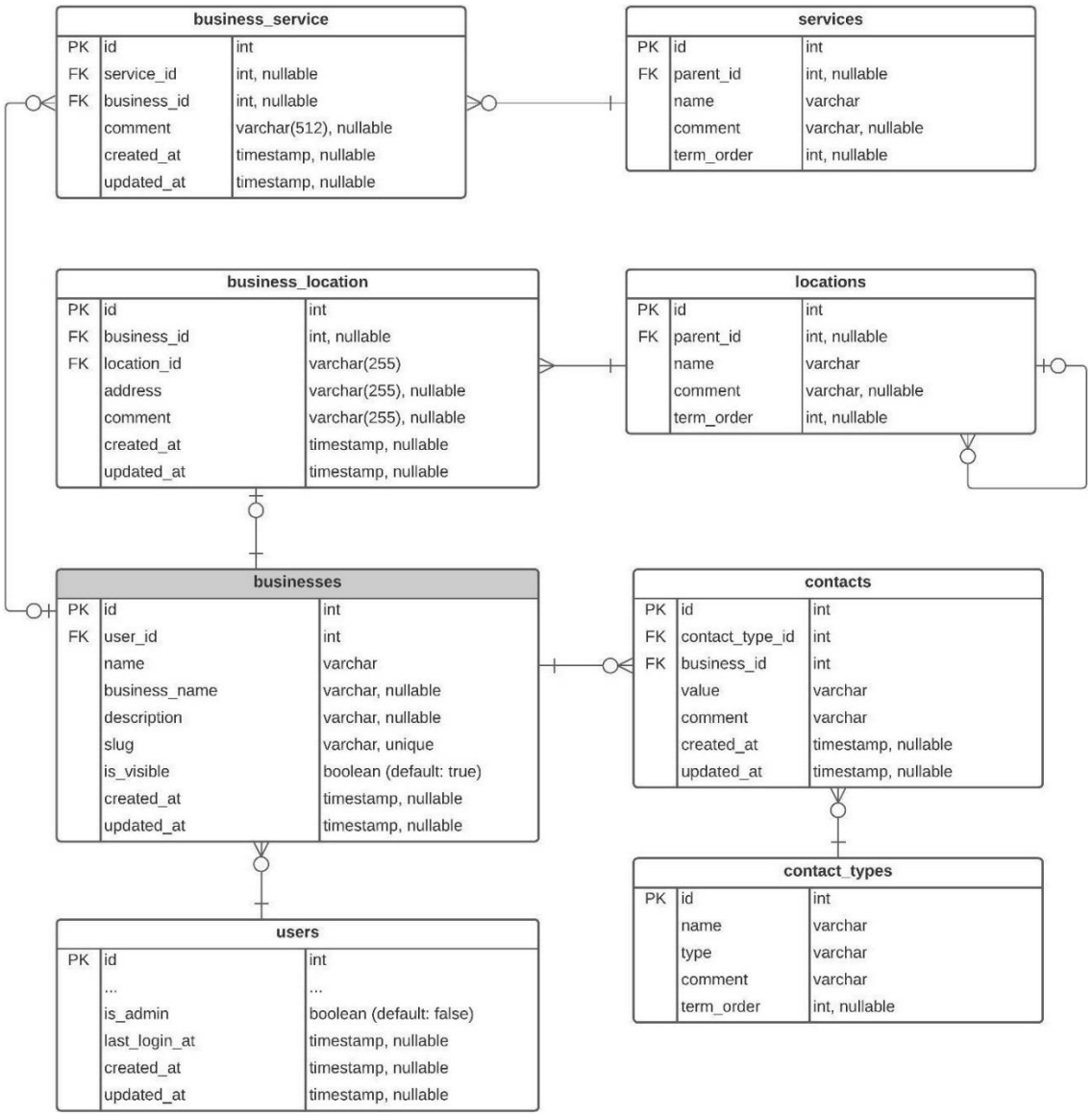
Andmete hoiustamise loogikat ja väljade nimetamist puudutavate küsimuste tekkimisel uuris autor, kuidas sarnaseid probleeme on lahendatud levinud platvormide nagu Wordpress, Drupal ja Magento andmemudelite puhul.

Kuna tegemist on prototüübiga, siis tehti keerukuste vältimiseks mõne andmemudeli puhul ka lihtsustusi. Andmemudeli osas, mis sisaldab teenuste loetelu (tabel „services“) ei teostatud iseendaga rekursiivses suhtes oleva teatmik-tüüpi olemina (nagu tehti asukohtade haldamisel), mis oleks vajalik, kui selgub, et teenustel on ka alamteenused. Näiteks teenuse „rehvivahetus“ alamtüübiks saaks olla kaubiku „kaubiku rehvivahetus“, „sõiduauto rehvivahetus“ jne. Samuti ei loodud teenuse seost sõiduki tüübiga, kuna selline lahendus oleks liiga spetsiifiline. Juhul, kui teenus vajab lisaselgitust, saab kasutada võimalust lisada teenuse kande juurde vabatekstilist kommentaari, mis on olemite sisemise struktuuri puhul soovituslik [27].

Ettevõtte ja asukoha vaheline seos „business_location“ on olemite tasemel mitu-mitmele seos aga äriloogiliselt saab ühel töökojal olla vaid üks kindel asukoht. Seetõttu on see olemi-suhte diagrammil toodud suhtena, kus ettevõttel võib olla „üks või ei ühtegi“

asukohta. Asukoht puudub, kui ettevõte on rakenduses loodud, kuid asukohta puudutavat kannet ei ole veel tehtud.

Laravel raamistiku loodud kasutajate info hoiustamiseks mõeldud tabelit „users“ täiendati ajatempli tüüpi väljaga „last_login_at“, et oleks võimalik tuvastada inaktiivseid kasutajaid ning boolean tüüpi väljaga „is_admin“ (vaikimisi *false*), mis annab võimaluse luua ligipääsuõiguseid kasutaja rollist tulenevalt.



Joonis 1. Andmebaasi olemi-suhte diagramm (andmemudel)

Olemi-suhte diagrammi joonistamiseks (käesolevas töös) kasutas autor veebirakendust Lucidchart.

Igal tabelil on andmebaasis oma roll, mida on kirjeldatud tabelis 6.

Tabel 6. Andmebaasi tabelid ja nende kirjeldus.

Tabeli nimi	**Kirjeldus**
businesses	Subjekt-tüüpi olem ettevõtet puudutava info hoiustamiseks.
business_location	Ettevõtte ja tema asukoha vaheline seos-tüüpi olem. Lisaks välisvõtmetele sisaldab aadressi väärtust ja kommentaari. Laraveli tavade kohaselt nimetatud ainsuses, tähestikulises järjekorras.
business_service	Seos-tüüpi olem ettevõtte ja tema osutavate teenuste vaheliste seoste hoiustamiseks.
contacts	Seos-tüüpi olem (mitu-mitmele) ettevõtte ja kontakti tüübi vahelise seose hoiustamiseks.
contact_types	Teatmik-tüüpi olem kontakti tüüpide hoiustamiseks.
locations	Iseendaga rekursiivses suhtes olev teatmik-tüüpi olem asukohtade ja nende omavaheliste seoste haldamiseks. Alam-teatmik sisaldab välisvõtmena vanem-asukoha (*parent location*) id'd. Hierarhia ahel võib olla kuitahes pikk.
services	Teatmik-tüüpi olem teenuste nimetuste hoiustamiseks.
users	Subjekt-tüüpi olem rakenduse kasutajat puudutava info hoiustamiseks.
Sanctum paketi ja Laravel raamistiku vaikimisi tabelid (kokku 4), mida olemi-suhte diagrammil ei ole välja toodud.	
failed_jobs	Ebaõnnestunud tööde logimiseks [30]
migrations	Andmebaasi migratsioone puututava info hoiustamiseks
password_resets	Parooli taastamise tokeni ja selle kehtivusaja hoidmiseks
personal_access_tokens	Kasutajate API tokenite, õiguste, viimase päringu ja kehtivusaja hoiustamiseks. Token hoitakse andmebaasis SHA2-256 räsina.

4.2 Serveripoolse rakenduse arendus

Käesolevas peatükis antakse ülevaade serveripoolse rakenduse arendusest ja teenustest, millele rakendus on erinevates arenduse etappides paigaldatud.

4.2.1 Laravel rakenduse loomine

Serverrakenduse projekti loomisel lähtus autor Laravel raamistiku dokumentatsiooni peatükist, milles käsitleti uue projekti loomist [11]. Arendaja arvutisse on eelnevalt installeeritud PHP 8 ja sõltuvuste haldamise programm Composer. Uus Laraveli projekt loodi käsureal käsuga:

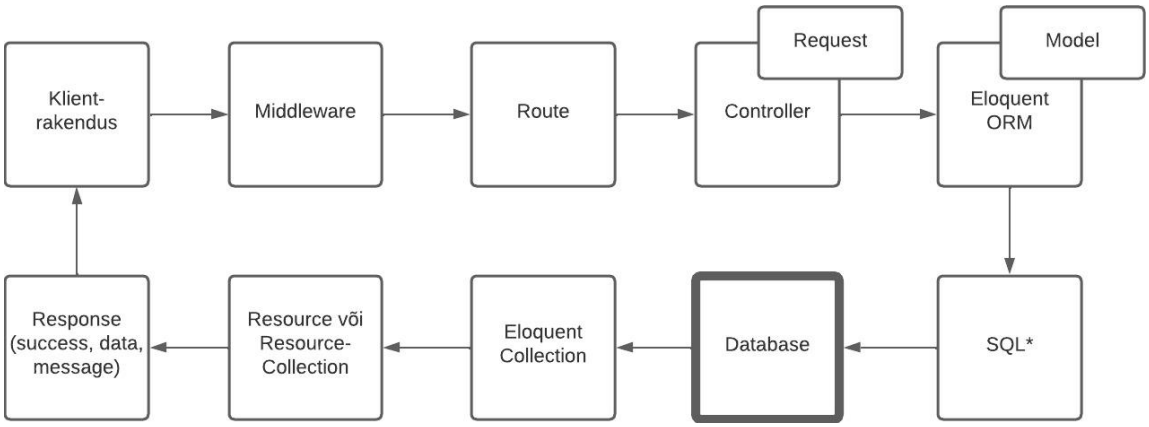
```
composer create-project laravel/laravel rehvivahetus-laravel
```

Kuna raamistikuväliseid sõltuvusi projektis ei kasutatud, siis täiendavalt Composeriga midagi installeerida ei olnud vaja. Sõltuvused on määratud rakenduse juurkataloogis composer.json failis ja neid on võimalik Composer'i käsurealiidese kaudu sõltuvuste loetelusse lisada (`composer require`), installeerida (`composer install`), eemaldada (`composer remove`) ja uuendada (`composer update`). Kuna sõltuvusi projekti Giti repositooriumisse ei lisata, siis repositooriumi kloonimisel tuleb sõltuvused Composeri abil installeerida.

Vastavalt GitLabi (ptk 3.2.6) juhendile loodi projekti kausta **Giti repositoorium** (käsuga `git init`), kontrolliti .gitignore failide olemasolu, korrektsust, lisati failid repositooriumisse ning laaditi kood üles Giti serverisse.

Seejärel **seadistati konfiguratsioonifailides** .env ja .env.testing andmebaasiühenduse parameetrid arendus- ja testkeskkonna jaoks ning selleks, et saada põhjalikumad veateated, määrati arenduskeskkonnas muutuja `APP_DEBUG` väärtuseks `true`.

Järgnevalt kirjeldatakse rakenduse arendust. Rakenduse erinevate komponentide arendamist kirjeldava osa jälgimise lihtsustamiseks koostas autor päringu teekonna joonise (joonis 2), mis aitab paremini mõista, missuguses rakenduse osas kõnealune komponent paikneb.



Joonis 2. Päringu teekond rakenduses.

Tarkvara arendust alustati varasemalt koostatud andmemudeli (kirjeldatud peatükis 4.1) põhjal andmebaasi migratsioonifailide koostamisest, mille põhjal (migratsiooni käigus) luuakse andmebaasitabelid. Migratsioonifailides määrati andmebaasitabeli atribuudid, esialgsed piirangud väärtustele (teksti pikkus, unikaalsus, null-väärtuse lubatavus, *cascade* lubatavus kustutamisel), andmetüübid ja omavahelised seosed (välisvõtmed, primaarvõtmed) andmebaasi tasemel.

Seejärel, tuginedes samuti andmemudelile, koostati Model klassid, mis annavad võimaluse andmebaasiga suhtlemiseks kasutada Eloquent ORMi mugavusmeetodeid. Model klassides kirjeldatakse andmemudelit, seoseid teiste mudelitega ning seatakse vajadusel piiranguid (ligipääs konkreetsetele väljadele ja nende muutmiste õigustele). Eloquent ORMi esmane kasu (autori hinnangul) on koodi muutumine loetavamaks (semantiliselt sõnastatud seosed on inimestele paremini mõistetavad) ning ORMi koostatud SQL päring on tõenäoliselt efektiivsem kui ise koostatud SQL päring (eriti keerukamate päringute puhul). Lisaks, kuna ORMi abil koostatud päring ei ole andmebaasi tüübi (Postgres, MySQL vms) spetsiifiline, siis tänu sellele muutub programm sõltumatuks konkreetsest andmebaasi tüübist. Joonisel 3 on toodud näide ORM abil koostatud päringust, mis tagastab ühe ettevõtte objekti asukohaga, teenuste loeteluga, kontaktandmetega (koos kontaktandme tüübiga).

```
$business = Business::where('id', $id)
	->with([
		'location',
		'services',
		'contacts',
		'contacts.contactType'
	])->first();
```

Joonis 3. Ühe ettevõtte päring id järgi, kasutades Eloquent ORMi.

ORMi abil koostatud päringu teisendatakse (vastavalt konfiguratsioonis määratud draiverile) SQL päringuks, tehakse andmebaasipäring ja vastusena tagastatakse Eloquent Collection tüüpi objekt. Collection objektiga kaasnevad mugavusmeetodid, mis võimaldavad päringu vastusest väärtuseid küsida vastavalt Model klassis kirjeldatud seostele. Eelnevas näites (joonis 3) toodud ettevõtte aadressi on Eloquent Collection tüüpi objektist võimalik pärida järgnevalt: `$business->location->address;`

Arendusprotsessi (eelkõige silumise) lihtsustamiseks loodi iga mudel kohta ka Factory klass, mis oskab mudeli põhjal objekte genereerida ning mudelile vastav Seeder klass, mille ülesandeks on Factory abil genereeritud (test)andmeid andmebaasi lisada. Testandmete puudumisel (tühja andmebaasi korral) ei oleks kontrolleritel kunagi midagi tagastada. Käsitsi andmete testandmete baasi lisamine on ebamõistlikult töömahukas. Soovitud hulgal testandmeid sai andmebaasi lisada migratsiooni käigus `--seed` lipu lisamisega käsurea käsule `php artisan migrate:fresh --seed.`

Mudeli valmimise järgselt alustati kontrolleri kirjutamist. Kontrolleri malli loomiseks kasutati artisan'i käsku, näiteks:

```
php artisan make:controller PostController --resource
```

Resource lipu lisamisel käsule luuakse kontrollerile kõikide CRUD meetodite mallid, millede hulgast töö autor jättis alles vaid API jaoks vajalikud – *index*, *show*, *store*, *update*, *destroy*.

Kontrollerite loomisel lähtuti NWCA reeglist (*never write custom actions*) [31], mis seisneb selles, et kontrolleris (üldjuhul) ei peaks olema rohkem meetodeid, kui eelnevalt loetletud viis CRUD meetodit. Samuti ei ole alati tingimata vaja kõiki viit meetodit. Kui tekib vajadus mõne täiendava meetodi järgi, siis tõenäoliselt on mõistlik selle ressursi jaoks teha uus kontroller ning puuduolev CRUD meetod sinna lisada.

Ainuke kontroller, mille puhul NWCA reeglist esialgu kõrvale kalduti, oli AuthController, mis teenindab kasutajaks registreerumise, sisse- ja väljalogimisega seotud päringuid (meetodid *login*, *register*, *logout*). Kõik juhendid, millest autor autentimise osa ehitamisel lähtus, kasutasid autentimisel vaid üht kontrollerit, kuid autoril on plaanis ka see NWCA reeglile vastavaks kirjutada.

AuthControlleri puhul loodi päringu sisendi valideerimiseks kaks *request*'i – UserLoginRequest ja UserStoreRequest. Tegemist on klassidega, mida saab kasutada päringut teenindava meetodi parameetri tüübina. Näiteks POST päring meetodile `store(UserStoreRequest $request) {…}` puhul kontrollitakse, kas `$request` vastab `UserStoreRequest` klassis määratud nõuetele (nt e-maili unikaalsus, nõuded paroolile).

Eloquent ORM tagastab Eloquent Collection tüüpi objekti, milles andmed (väljade nimed ja väärtused) on sellisel kujul nagu nad on andmebaasis. Sellisel korrastamata kujul ei ole mõistlik kogu infot kliendile vastusena saata. Selleks, et andmed viia vastavusse JSON:API spetsifikatsiooniga [32], on Laravel raamistikus võimalik kasutada Resource (ressurss) klasse. Igale äriloogilisele andmeühikule on võimalik kujundada enda Resource klass sõltuvalt sellest, missugune on päringu vastus, mida soovitakse anda. Resource klass võtab parameetriks andmebaasipäringu vastuse ja teisendab selle vastavalt Resource klassides määratud kujule.

Seejärel vormistati API päringuteekonnad (*routes*, juurkataloogis routes/api.php failis) ning viited vastavat päringut teenindavale ressursile (kontroller ja täpne meetod kontrolleris). Joonisel 4 on toodud näide ühe ressursi (/api/v1/businesses) teekondadest (*routes*) ja *middleware*'i abil autentimist eeldavale päringuteekonnale juurdepääsu piiramisest.

```
// Public route

Route::group(['prefix' => 'v1'], function () {
	Route::resource('/businesses', BusinessesController::class)
		->only('index', 'show');
	}
);

// Private route

Route::group(
	['prefix' => 'v1', 'middleware' => ['auth:sanctum']],
	function () {
		Route::resource('/businesses', BusinessesController::class)
			->only('store', 'update', 'destroy');
		}
);
```

Joonis 4. Päringuteekondade vormistamise näited routes/api.php failis kasutades grupeerimist.

Middleware (vahetarkvara) on funktsionaalsus, millega saab filtreerida rakenduse vastu tehtud HTTP päringuid. Kõik päringuteed, mis paiknevad autentimist kontrolliva *middleware*'iga kaetud grupis (joonis 4, alumine näide), on piiratud juurdepääsuga ning täiendavalt kasutaja õiguseid (näiteks kontrolleri meetodis) ei ole vaja kontrollida. Kuna osa rakenduse funktsionaalsusest eeldab lisaks autentimisele ka kasutajarolli „is_admin", kirjutas töö autor *middleware*'i, mis selle tingimuse täitmist kontrollib.

Andmebaasitabelite loomiseks ehk migratsioonide tegemiseks kasutati (arenduse käigus) käsureal käsku `php artisan migrate:fresh --seed`, mis kustutab kõik andmebaasi tabelid, teeb kõik migratsioonid uuesti ning täidab tabelid testandmetega vastavalt DatabaseSeeder klassi seadistustele.

Arenduse igas etapis, kus valmis testimist võimaldav ühik tööd (näiteks API päring tagastas vastuse), siis kaeti see automaattestidega (kirjeldatud põhjalikumalt peatükis 4.2.5) tagamaks, et rakenduses muudatuste tegemisel edaspidi seda funktsionaalsust katki ei tehta.

4.2.2 Kasutajate ligipääs tagarakendusele

Kasutajatel on võimalik tagarakendusele ligi pääseda API kaudu. Käesoleva lõputööga seotud arenduste ajal on API ligipääsetav aadressilt https://dev-rehvivahetus-api.that.ee/api/v1, millele on vaja lisada viide soovitud ressursile.

API ressursse on põhjalikumalt kirjeldatud peatükis 4.2.4. Tulenevalt rakenduse iseloomust on enamus ressursside kasutamise eelduseks autentimine kasutajanime ja parooli abil ning ligipääsuõiguste olemasolu. Autentimise vastusena tagastatakse token, mis tuleb lisada päringu tegemisel päisesse. Tokeni põhjal kontrollitakse õiguste olemasolu (*authorization*). Tokenil põhinev autentimise ja autoriseerimise süsteem on teostatus Laravel Sanctum paketi abil.

Selleks, et oleks võimalik ligipääs tagarakendusele, valis autor tagarakenduse paigaldamiseks Zone Media virtuaalserveri teenuse (*shared hosting*).

Jagatud majutuse kasutamine annab ligipääsu võimsale riistvarale väga soodsa hinnaga (võrreldes privaatserveriga või virtuaalse privaatserveriga (VPS)) ning võimaldab keskenduda rakenduse haldamisele (selgitatud peatükis 3.3). Kuna nõudlus rehvivahetuse teenuse järgi on hooajaline, siis sobib selline jõudluse kasutamise muster ka jagatud majutuse põhimõtetega, et enamasti on kulub ressurssi vähe aga vajadusel saab lühiajalise ligipääsu suuremale hulgale jõudlusele.

Uurides serveri jõudlust terminalis lscpu ja htop tööriistadega, leidis autor, et Zone Media kõige soodsam pakett (kuu hind 6.55 eur + KM) annab ligipääsu 126 GB mälule ja 39 protsessorituumale. Sarnase jõudluse ühe kuu hind populaarses Digital Oceani VPS teenuses (General Purpose Droplet, 128GiB, 32 vCPU, seisuga 15.11.22) ilma lisateenusteta (hallatud andmebaasid, varukoopiad) on 1008 dollarit.

Rakenduse paigaldamiseks (*deployment*) on rakenduse juurkataloogis bashi skript, mis loob SSH ühenduse serveriga ning käivitab kaustas, kus asub rakenduse Giti repositooriumi põhiharu (*main*) kloon, joonisel 5 toodud käsud.

```
git fetch
git pull
composer install
php artisan migrate
```

Joonis 5. Bashi skripti käsud rakenduse paigaldamiseks pärast SSH ühenduse loomist.

4.2.3 Andmebaasi loomine rakendusele

Rakenduse andmebaasisüsteemiks (DBMS) valiti MariaDB (otsustusprotsessi kirjeldatud peatükis 3.3). Andmebaasiga ühendumiseks määrati rakenduse .env ja .env.testing konfiguratsioonifailides andmebaaside parameetrid. Seejärel loodi andmebaasi tabelite kirjeldused migratsiooni failidena. Tabelite loomiseks kasutati Laraveli migratsiooni funktsionaalsust ja Laravel artisan käsurealiidest.

Kuna andmebaasitabelid luuakse migratsioonifailide tähestikulises järjekorras, siis on oluline jälgida, et omavahel seostatavad olemid loodaks enne nendevahelise seose loomist. Migratsioonide üle peetakse arvet vastavas andmebaasitabelis „migrations“.

Arenduse ajaks lisati andmebaasi testandmed, mille genereerimiseks on Laravelis olemas vastavad võimalused (*seeder, faker, factory*).

- Arenduse andmebaasiserverina kasutati MariaDB versioon 10.4, mis paiknes arendaja arvutis Dockeri konteineris koos phpMyAdmin kasutajaliidesega. Konkreetse MariaDB versiooni valikul ei olnud muud põhjendust, kui et see oli arendaja arvutis muude arendusprojektide tarbeks juba olemas ning Laraveli nõue on MariaDB versioon alates 10.3. Sellest tulenevalt 10.4 oli antud juhul sobiv.
- *Prelive* andmebaasiserver paiknes Hetzneri Helsingi andmekeskuses VPSis (pakett CPX11, AMD protsessor, 2vCPU, 2GB RAM) samuti Dockeri konteineris koos phpMyAdmin kasutajaliidesega.
- Toodangu andmebaas asus Zone.ee serveris.
- Testandmebaas, mida kasutasid *feature testid* asus samuti eelpool nimetatud Hetzneri VPSis Dcokeri konteineris.

Lokaalse, *prelive* ja testimiseks kasutatud andmebaasiserverite ja phpMyAdmin kasutajaliideste Dockeri konteinerite loomiseks ja käivitamiseks kasutati docker käske, mis on toodud joonisel 6.

```
docker run --restart always --name mariadb10_4 --env
MARIADB_ROOT_PASSWORD=4mm2p9nn7h -d -p 3306:3306 mariadb:10.4
docker run --restart always --name phpmyad -d --link mariadb10_4:db -p
8090:80 phpmyadmin
```

Joonis 6. Dockeri käsud andmebaasiserveri ja phpMyAdmin kasutajaliidesega Dockeri konteineri loomiseks ja käivitamiseks

4.2.4 RESTful API

API on nõuete kogum (leping), mis määratleb, kuidas rakenduse osapooled saavad omavahel suhelda [14]. REST (*Representational State Transfer*) on rakendusliidese arhitektuuriline lahendus, mis põhineb 5-6 piirangul, millega tuleks veebiteenuste arendamisel arvestada ja hulk tavasid [33]. Piirangute üheks eesmärgiks on tagada RESTful arhitektuuril põhivate liideste ühetaolisus. Kui arendaja on mõnd REST APIt

kasutanud, siis tõenäoliselt oskab ta kasutada (tarbida) ka teisi REST APIsid, sest eeldus on, et nad on ühetaolised.

REST API võimaldab klientrakenduse ja serverrakenduse omavahelise suhtluse üle HTTP protokolli. REST API puhul antakse kliendile komplekt URLe, millest igaüks vastutab mingi kindla ressursi eest. URLi struktuurist peaks olema võimalik aru saada, missugust ressurssi konkreetne aadress teenindab. URLi poole pöördumiseks on kasutusel (enamasti) neli verbi, mis pannakse päringuga kaasa – GET, PUT, POST, DELETE – mis kirjeldavad, mida serverile saadetud sõnumiga teha soovitakse.

REST API päringute ressursside (tabel 7) aluseks võeti funktsionaalsed nõuded. Alljärgnevas tabelis tuuakse välja kõige olulisemad rakenduse funktsionaalsustega seotud päringud ning teostatud meetodid. Enamusele neist kehtivad kliendi kasutajaõigustest tulenevad juurdepääsupiirangud (märgitud tabelis tärniga).

Igale pöördumine eelneb eesliide /api/v1, mis viitab sellele, et tegemist on API liidesega ning selle liidese esimese versiooniga.

Tabel 7. API ressursid

Ressurss	**HTTP meetodid**
/businesses	GET
/businesses/{id}	GET, POST*, PUT*, DELETE*
/businesses/{id}/location	GET*, POST*, PUT*, DELETE*
/businesses/{id}/services	GET*, POST*
/locations	GET
/locations/{id}	GET*, POST*, PUT*, DELETE*
/services	GET
/services/{id}	GET, POST*, PUT*, DELETE*
/contacttypes	GET*
/contacttypes/{id}	GET*, POST*, PUT*, DELETE*
/businesses/{id}/contacts	POST*
/businesses/{id}/contacts/{id}	GET*, PUT*, DELETE*
/user	GET*
/login	POST
/register	POST
/logout	POST*

API päringuteekondade ja vastuste (ressursside) disainimisel järgis autor JSON:API spetsifikatsiooni ja soovitusi [34].

Filtreerimist võimaldavate päringuparameetrite kasutamisel olid aluseks JSON:API spetsifikatsioon ja soovitused [32]. Lisaks tutvus autor rakendust kavandades spatie/laravel-query-builder paketiga. Tegemist on Laraveli paketiga API päringute põhjal andmebaasipäringute koostamiseks (kontrollerites) ning sisaldab muu hulgas filtreerimise funktsionaalsust. Esialgu planeeriti seda paketti kasutada filtreerimist sisaldavate päringute mugavamaks (ja eeldatavasti veakindlamaks) koostamiseks. Viimasest siiski loobuti, et tagada päringute ühetaolisus läbi kogu rakenduse (kõikjal kasutusel Laravel query builder ja Eloquent ORM). Samuti selgus töö käigus, et keerulisemate päringute koostamiseks paketi mugavuslahendustest ei piisanud ning neid tulnuks omakorda laiendama hakata, mille tulemusena oleks kood muutunud hoopis keerulisemaks.

Filtreerimine on võimalik /businesses ressursil ning alljärgnevalt (tabel 8) on toodud mõned näited, kuidas filtreerimist sisaldavate päringute tegemise võimalus on APIs teostatud.

Tabel 8. Ettevõtete filtreerimine API päringu parameetritega

Ressurss ja parameetrid	**HTTP meetod ja vastuse sisu**
/businesses?filter[location]=1	GET, tagastatakse ettevõtted, mille asukoha id on 1 ja kõik ettevõtted, mille asukoha juurtipu id on 1.
/businesses?filter[service]=1,2	GET, tagastatakse ettevõtted, mis ostutavad teenuseid, mille id on 1 ja 2. Mõlemad tingimused peavad olema täidetud.
/businesses?filter[location]=1& filter[service]=1,2	GET, kombinatsioon kahest eelnevast päringust asukoha ja kahe teenuse järgi.

Üks RESTful arhitektuuriga API üks piirangutest [33] on, et päringu vastuseid peab olema võimalik ajutiselt talletada kiiresse puhvermälusse (*cache*). Sagedamini esinevad päringute vastused on võimalik salvestada puhvermälusse. Laravelis on olemas failipõhine puhvermälu ja tugi populaarsete *cache* lahendustele nagu Memchached, Redis ja DynamoDB. Samuti on Laravelis olemas Eloquent ORM events ja Observer classes [28], et jälgida andmete muudatusi ja uuendada vajadusel puhvermälu, et päringu vastusena ei tagastataks aegunud andmeid. Puhvermälust tagastatava vastuse puhul ei

tehta pöördumist andmebaasi poole (ei ole vaja luua ühendust andmebaasiga), mis teeb vastuse tagastamise kiiremaks.

4.2.5 Serveripoolse rakenduse testimine

Serveripoolse rakenduse testimine koosnes automaattestidest (*unit* testid, *feature* testid) ja manuaaltestidest (Postman tarkvara abil). Töö kirjutamise hetkel sisaldab projekt 6 *feature* testi (kokku 60 test-juhtu) ja 1 *unit* testi (5 test-juhtu).

API päringute manuaaltestimine Postmaniga annab hea ettekujutuse, kas andmebaasipäring toimub efektiivselt ning kas kõik vajalik on vastuses olemas. Päring võiks aega võtta (lokaalse andmebaasi vastu) suurusjärgus 20 ms. Kui päring võtab aega 1 sekundi või enam, siis tõenäoliselt on päringu koostamisel esinenud vigu. Enamlevinud veaks arenduse käigus oli n+1 päringu rakendumine olukorras, kus seos oli unustatud *eager load* päringusse lisada ja toimus andmete laadimine laisalt (*lazy load*).

Postman võimaldab mõningal määral API päringu vastuseid automaatselt testida [35], kuid testide haldamine on võrreldes PHPUnitiga keerukam. Postmani API testimise funktsionaalsust kasutati töös selleks, et päringu vastusest konkreetseid väärtuseid lugeda (näiteks autentimise token või uue kande identifikaator) ja need edasiseks kasutamiseks muutujasse kirjutada

Suur osa tagarakenduse arendusest on andmebaasipäringute koostamine ja nende optimeerimine. Mahuka vastuse puhul sisu korrektsust ainult visuaalse vaatlusega testida ei ole mõeldav. Sellest tuleneval otsustas töö autor kasutada tagarakenduse testimiseks ka automaatteste. Testide loomiseks kasutas autor PHPUnit raamistikku, mis on Laraveliga vaikimisi kaasas (ei pea paketti eraldi lisama) koos testi mallide näidistega.

Rakenduse korrektseks toimimiseks oli vaja kirjutada üksikud *unit* testid autori koostatud abimeetodite (*helpers*) testimiseks. Näiteks meetod, mis leiab asukohtade filtris, kus puu juurtipu id järgi leitakse kõigi vahetippude ja lehtede id’d. Selline meetod on kasutusel töökodade filtreerimisel asukoha järgi, kus päringu juurtipu id (näiteks maakonna id) järgi pärimisel on vaja tagastada vahetippude ja lehtede id’d (linnad, linnaosad).

Suurema osa testidest moodustasid *feature* testid, mille eesmärgiks oli tagada API päringute ja vastuste vastavus nõuetele ning programmi veakindlus koodimuudatuste järgselt. Seal hulgas juurdepääsupiirangute testimine. Testiti nii vastuse JSON kuju (võtmed, väärtused), vastuse koode (*response code*), vastuste sisu. Kui *feature* töötab

korrektselt, siis võib eeldada, et ka teostud lahenduse detailid töötavad korrektselt ehk neid detaile eraldi *unit*-testida ei ole vaja. Autori hinnangul *feature* testide arv kindlasti kasvab, kuna testidega katmist vajavad kõik API päringud kõigi kasutajaõiguste (admin, kasutaja, külastaja) vaatest.

Automaattestid kasutavad testandmete loomiseks ja hoiustamiseks eraldi test-andmebaasi (samuti MariaDB). Testandmebaasi seaded on määratud rakenduse konfiguratsioonifailis .env.testing.

Kuna Laraveli raamistikus on olemas mugavusmeetodid suure hulga andmete genereerimiseks (*database seeder*, *faker*, *model factory*), siis on teoreetiliselt võimalik (koormus)testida ka rakenduse tööd suuremate andmehulkade puhul. See võimaldaks välja selgitada, kas serveri poolse rakenduse kiirus jääks samaks ka juhul, kui andmeid oleks 100 või 1000 korda rohkem. Käesoleva töö raames sellisel kujul koormustestimist ei teostatud.

4.3 Klientrakenduse arendus

Klientrakenduse arendus lähtus analüüsi etapis selgitatud nõuetest ning arendusega alustati, kui suurem osa tagarakenduse kõige kõrgema prioriteediga funktsionaalsustest oli valmis ning testitud (veendumaks klientrakenduses vigade esinemisel, et toimib korrektselt on töökorras).

Järgnevates peatükkides kirjeldatakse klientrakenduse loomist, suhtlust serverrakendusega, kasutajate ligipääsu rakendusele ning testimist.

4.3.1 Klientrakenduse loomine

Tagarakenduse valmimise järgselt alustati klientrakenduse (käesolevas peatükis edaspidi ka „rakenduse“) arendusega. Rakenduse põhja loomiseks kasutati Create React App tööriista [36]. Täiendavalt vajalike pakettide installeerimiseks kasutati käsureal npm käske. Rakenduse loomiseks ja pakettide installeerimiseks kasutatud käsud on toodud joonisel 7.

```
npx create-react-app rehvivahetus-react --template typescript
npm install @popperjs/core bootstrap axios react-router-dom
npm install --save-dev @types/bootstrap
```

Joonis 7. Käsurea käsud React rakenduse loomiseks.

Esimese npx käsuga loodi rakenduse põhi, lisati TypeScripti tüübikontroll [37] ning määratakse rakenduses kasutatavaks paketihalduriks npm. Seejärel installeeriti npm käsuga rakenduses kasutatavad paketid popperjs, bootstrap, axios ja react-router-dom. Lõpus lisati TypeScript tüüpide tugi (arenduse ajaks sõltuvustesse) Bootstrap pakettidele.

Pakett bootstrap annab rakendusele stiiliraamistiku Bootstrap toe, mida otsustati prototüübi disainimiseks kasutada. Bootstrap on kõige enam kasutatud stiiliraamistik kasutajaliideste kujundamiseks [38]. Prototüübi disainimisel oli kasutusel Bootstrapi kõige uuem versioon 5.2.2. Osa Bootstrapi võimalustest vajab toimimiseks popper.js paketti, mistõttu see sai rakendusele lisatud.

HTTP päringute tegemiseks (APIga suhtlemiseks) installeriti rakendusele Axios [39] pakett (versioon 1.1.2). Axios on HTTP klient, mis põhineb JavaScript Promise objektidel, tänu millele mugav hallata ka ebaõnnestunud päringuid (tuleb lahendada rakenduse käitumine ebaõnnestumise korral).

Rakenduses navigatsiooni teostamiseks valiti React Router (versioon 6.4.2) pakett [40].

Rakenduse globaalse oleku (*state*) haldamiseks kasutati React Contexti, kuhu loodi kaks konteksti – authContext ning filterContext.

- FilterContextis hoiti filtreerimise komponentide olekut, et lehel navigeerides filtri algolek igakordselt ei taastuks. Kontekstis hoitakse filtris valitud asukoha id väärtust ning listina valitud teenuste id-sid. Lehe uuesti laadimisel (*refresh*) on ootuspärane filterContexti algoleku taastumine.
- AuthContextis hoitakse autenditud kasutaja andmeid (kasutajaliideses kuvamiseks) ja juurdepääsuõiguseid defineerivat tokenit. Autentimisel salvestatakse token ka brauseri mälusse (*localStorage*). Lehe uuesti laadimise korral (*refresh*), kui token on brauseri mälus olemas, tehakse lehe laadimisel päring /user endpointi, kust päritakse „sisse loginud“ kasutaja andmed. Kui token on olemas (ja kehtiv), siis pannakse kasutaja info uuesti authContexti ja edaspidi käitub rakendus autenditud kasutaja vaatest. Kui päringu tulemusena selgub, et token brauseri mälus ei ole korrektne (kehtiv), siis kustutati see brauseri mälust.

4.3.2 Suhtlus serveripoolse rakendusega

Suhtlus serveripoolse rakendusega toimub üle REST API (kirjeldatud peatükis 4.2.4). REST API võimaldab klientrakenduse ja serverrakenduse omavahelise suhtluse üle HTTP protokolli. Suhtluse algatamine toimub ühesuunaliselt – klientrakendus küsib serverrakenduselt. Klientrakenduses HTTP päringute koostamiseks kasutati Axios klienti.

Selleks, et vältida vajadust kirjutada ühetaolist koodi mitu korda, loodi päringute tegemiseks üldkasutatav baasklass (base-service.tsx), kus on teostatud geneerilised meetodid GET, POST, PUT ja DELETE päringute tegemiseks. Kõik rakenduse komponendid saavad neid baas-meetodeid kasutada. Päringut algatav komponent peab meetodile lisama viite API ressursile (loetletud peatükis 4.2.4), tüübiparameetrid ja juurdepääsupiirangu korral ka autentimise tokeni (mis lisatakse seejärel päringu päisesse).

Juurdepääsupiiranguga API päringute tegemiseks on nõutav kasutaja autentimine (ehk kasutaja tuvastus). Kasutaja tuvastamiseks kasutatakse antud rakenduses kasutajanime ja parooli. Autentimise päringu vastusena tagastatakse token ja user objekt. Piiranguga päringute tegemiseks tuleb päringu päisesse lisada token ja selle põhjal otsustatakse, kas päringu tegija on autenditud (kas token on olemas, kehtiv ja kellele see kuulub) ning autoriseeritud või mitte. Token on komponentidele kättesaadav React Contexti kaudu, kuhu loodi autenditud kasutaja info hoidmiseks authContext.

4.3.3 Kasutajate ligipääs rakendusele

Klientrakendus on avalik ning sellele on ligipääs kõigil kasutajatel. Rakenduse äriloogikast tulenevalt on tavakülastaja, registreerunud kasutaja ja administraatori kasutajaliidese vaated erinevad. API piirangutest lähtuvalt erinevad ka nende kasutajate õigused teha erinevaid toiminguid.

Klientrakenduse majutuseks valis autor teenuse Cloudflare Pages tasuta paketi, mis sisaldab 500 rakenduse paigaldust (*deployment*) kuus. Paigaldus toimub automaatselt Giti repositooriumi *development* haru ühildamisel (*merge*) põhiharusse (*main*).

Käesoleva lõputöö kirjutamise ajal on klientrakendus ligipääsetav aadressilt https://demo.rehvivahetus.eu.

4.3.4 Klientrakenduse testimine

Klientrakenduse puhul kasutati manuaaltestimist arvuti ja nutitelefoni brauseris. Chrome brauseri *network* ja *console* vaates uuriti rakenduse kasutamise ajal API päringute käivitumise järjekorda, tagastuskoode (et need oleks informatiivsed, asjakohased) ja päringute kiirust.

Rakenduse edasise arenduse puhul oleks mõistlik kasutusele võtta Cypress testid [41]. Cypress.io on raamistik lõppkasutajatestide (*end-to-end*) automatiseerimiseks. Sellest on (autori hinnangul) enim kasu pikemate kasutajateekondade testimise puhul, kus kogu teekonna korduv läbikäimine (näiteks test-stsenaariumite põhjal) võib muutuda kiiresti väga ajamahukaks. Cypress testid käivituvad brauseris ning seejärel rakenduse käitumist interaktiivselt jälgida. Samuti on võimalik Cypress teste integreerida CI torusse (*pipeline*).

5 Tulemused

Käesolevas peatükis kirjeldatakse lõputöö tulemust ning edasiarenduse võimalusi. Töö tulemustele objektiivse hinnangu andmiseks vaadatakse, kuidas valminud rakendus vastab analüüsi käigus seatud nõuetele, hinnatakse kasutatud tehnoloogiate sobivust seatud eesmärkidega ning nende ajakohasust.

5.1 Loodud funktsionaalsused

Lõputöö praktilise osana teostati kõik kõrgeima prioriteediga (*Must have*, M) funktsionaalsed nõuded, mida kirjeldati peatükis 3.1.1. M prioriteediga nõuete täitmine tähendab, et rakenduse prototüüpi on võimalik eesmärgiks seatud määral kasutada.

Kõigi M prioriteediga nõuete teostamisel on (täiendava keerukusena) arvestatud rakenduse kasutajale kehtivate juurdepääsupiirangutega nii klient- kui tagarakenduse puhul.

Kõige keerukama ja kõige põhjalikumat uurimist nõudnud funktsionaalsusena on võimalik esile tuua töökodade asukoha- ja teenustepõhist filtreerimist, mis on ühtlasi rakenduse äriloogika kõige olulisem komponent. Asukohtade haldus on andmemudelis teostatud iseendaga rekursiivses suhtes oleva olemina, tänu millele on andmemudel aegpüsiv ja kohanduv erinevate (maailmas esinevate) haldusjaotustega, võimaldades maksimaalselt paindlikku filtreerimist.

Rakenduse prototüüp vastab kõigile kolmele mittefunktsionaalsetele nõuetele, mis seati analüüsi käigus (ptk 3.1.2) – kasutatav enamlevinud brauserites, kasutatav väikesel ekraanil (kohanduv disain) ja vastavuses kasutajaõigustega. Mittefunktsionaalsete nõuetega on tagatud ligipääs klientrakendusele ja rakenduse kasutatavus erinevates võimalikes olukordades.

Arenduse käigus tekkivate vigade võimalikult varajaseks avastamiseks ja seeläbi töökindluse tagamiseks on tagarakendus kaetud automaattestidega.

5.2 Kasutatud tehnoloogiate uudsus ja sobivus

Tehnoloogiate valikul võeti arvesse nende populaarsust (tuginedes erinevatele küsitlustele ja avalikule statistikale), tänapäevaseid veebirakenduste arenduspraktikaid, sobivust rakendusele seatud nõuete ning autori kogemust. Sellest tulenevalt leiti sobivaks lahenduseks luua eraldiseisvad klient- ja tagarakendus, mis omavahel suhtlevad üle RESTful arhitektuuril põhineva liidese (API).

Klientrakenduse loomiseks kasutati React teeki (versioon 18, kõige uuem), mis on JavaScriptil põhinevate klientrakenduste puhul kõige levinum, aktiivselt arenev ning millelt on võimalik edasi liikuda React teegil põhineva raamistiku Next.js kasutamisele.

Tagarakenduse ja REST API loomiseks kasutati raamistikku Laravel (versioon 9, kõige uuem), mis on PHP raamistikest kõige populaarsem, aktiivselt arendatav, veebirakenduste jaoks sobiv ning võimaldab teostada tulevikukindlalt rakenduse toimimiseks tarvilikke arendusi. Laraveli sobivust kinnitab ka asjaolu, et loodud tagarakendus on sõltumatu konkreetsest andmebaasi tüübist (piisab draiveri vahetusest) ning omab keerukama (OAuth2 standardiga) autentimise tuge.

Rakenduse arenduseks valitud tehnoloogiad ja kasutatud arendusvõtted on autori hinnangul tänapäevased ning sobivad seda tüüpi veebirakenduste arendamiseks.

5.3 Edasiarenduse võimalused

Töö edasised arendused jagunevad autori hinnangul kaheks – tehniliste puudujääkide eemaldamine ning madalama prioriteediga nõuete edasine teostamine.

Esiteks vajavad analüüsi ja arendust tehnilised puudujäägid, mis ilmnesid arenduse käigus või arenduse järel. Näiteks, klientrakendusele võimekama SEO toe saamiseks võiks React teegi (*library*) asendada React teegil põhineva raamistikuga Next.js. Otsimootorile optimeeritud ja hästi ligipääsetav sisu on rakenduse puhul ärikriitiline. JavaScriptil põhinevate teekide kasutamisega kaasnevat SEO probleemistikku ei ole varasemates lõputöödes (millega autor tutvus) käsitletud. Samuti on autori hinnangul võimalusi koodi refaktoreerimiseks nii klient- kui ka tagarakenduse poolel.

Näiteks tagarakenduse API autentimist puudutava koodi struktuur tuleks viia vastavusse NWCA (*Never Write Custom Actions*) reegliga, eeskujuks võiks võtta Laravel Breeze API stardikomplekti (*starter kit*) autentimise kontrollerid, mille on koostanud Laraveli autorid.

Teiseks, jätkata madalama prioriteediga nõuete teostamisega, mis lõputöö skoobist välja jäid ning vajadusel nõudeid uuesti prioritiseerida. Senise tagasiside põhjal looks kasutajatele enim väärtust võimalus lisada profiilile (referents)hinnakiri (võimaldaks hinnataseme järgi sorteerimist), lahtiolekuajad ning võimalus näha töökoja kaugust teenuse otsijast. Viimane neist eeldab tõenäoliselt mõne välise teenuse kasutamist, mis enamasti tasulised (näiteks Bing Maps REST API, here, Google Distance Matrix API). Selleks, et tagada info korrektsus lehel on autori hinnangul oluline luua mugav võimalus teavitada administraatorit ebakorrektsest informatsioonist lehel.

Kui tehnilised puudujäägid on eemaldatud ning järgmised väärtust loovad funktsionaalsused teostatud, siis on võimalik anda uus hinnang prototüübi potentsiaalile punktis 2.1. tutvustatud probleemide lahendamisel ning võimalusel edasi mõelda võimalikele ärimudelitele.

6 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli luua veebirakenduse prototüüp rehvitöökodade leidmiseks ja reklaamimiseks. Vajaduse uudse veebirakenduse järele tekitas olemasolevate alternatiivide sobimatus nii spetsiifilise teenuse reklaamimiseks kui ka sobiva teenuseosutaja leidmiseks.

Töö käigus valmis veebirakenduse prototüüp, mis võimaldab töökodadel (ettevõtetel) oma teenuseid reklaamida ning aitab teenust vajaval inimesel sobivat töökoda leida. Töö tagarakenduse loomiseks kasutas autor Laravel raamistikku (PHP keeles) ning klientrakenduse loomiseks React teeki (TypeScript keeles).

Töö eesmärgid autori hinnangul saavutati. See tähendab, et kõige kõrgema prioriteediga funktsionaalsused on rakenduse prototüübis teostatud ning rakenduse põhifunktsionaalsus on kasutatav. Madalama prioriteediga ja ressursimahukamad funktsionaalsused, mis muudaksid rakenduse kasutamist mugavamaks ning looksid kasutajatele täiendavat väärtust, on plaanis teostada arenduse järgnevates etappides arvestades ka kasutajate tagasisidet.

Töö edasised arendused jagunevad autori hinnangul kaheks – tehniliste puudujääkide eemaldamine (esmalt peamiselt koodi refaktoreerimine) ning madalama prioriteediga nõuete edasine teostamine.

Kasutatud kirjandus

[1] A. Hudaib, M. Qasem, R. M. Masadeh ja A. I. Alzaqebah, „Requirements prioritization techniques comparison,“ *Modern Applied Science,* p. 62, January 2018.

[2] M. Rehkopf, „Stories, epics, and initiatives,“ Atlassian, [Võrgumaterjal]. Available: https://www.atlassian.com/agile/project-management/epics-stories-themes. [Kasutatud 11 November 2022].

[3] M. Vestola, „A comparison of nine basic techniques for requirements prioritization,“ *Helsinki University of Technology,* pp. 1-8, 2010.

[4] „What is a distributed system?,“ Atlassian. Pty Ltd, [Võrgumaterjal]. Available: https://www.atlassian.com/microservices/microservices-architecture/distributed-architecture. [Kasutatud 11 november 2022].

[5] „Microsoft Learn - The world is distributed,“ Microsoft , [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/dapr-for-net-developers/the-world-is-distributed. [Kasutatud 11 November 2022].

[6] E. Doherty, „MVC Architecture in 5 minutes: a tutorial for beginners,“ Educative, Inc, 5 May 2020. [Võrgumaterjal]. Available: https://www.educative.io/blog/mvc-tutorial. [Kasutatud 11 November 2022].

[7] „History of PHP ¶,“ The PHP Group, [Võrgumaterjal]. Available: https://www.php.net/manual/en/history.php.php. [Kasutatud 11 November 2022].

[8] „Usage statistics of PHP for websites,“ W3Techs, [Võrgumaterjal]. Available: https://w3techs.com/technologies/details/pl-php. [Kasutatud 11 November 2022].

[9] „University Information - What is the difference between a compiled and an interpreted program?,“ Indiana University Knowledge Base, 20 08 2021. [Võrgumaterjal]. Available: https://kb.iu.edu/d/agsz. [Kasutatud 11 November 2022].

[10] „Privaatserver - Klienti haldus,“ Zone Media OÜ, [Võrgumaterjal]. Available: https://www.zone.ee/et/privaatserver/puhas-raud/. [Kasutatud 11 November 2022].

[11] Laravel LLC, „Laravel Documentation for version 9.x,“ Laravel LLC, 2022. [Võrgumaterjal]. Available: https://laravel.com/docs/9.x. [Kasutatud 2022].

[12] „PHP Programming - The State of Developer Ecosystem in 2021,“ JetBrains s.r.o., 2021. [Võrgumaterjal]. Available: https://www.jetbrains.com/lp/devecosystem-2021/php/. [Kasutatud 11 November 2022].

[13] Stack Overflow, „Stack Overflow Developer Survey 2022,“ Stack Overflow, 2022. [Võrgumaterjal]. Available: https://survey.stackoverflow.co/2022/. [Kasutatud 11 November 2022].

[14] „What is a REST API?,“ IBM, [Võrgumaterjal]. Available: https://www.ibm.com/cloud/learn/rest-apis. [Kasutatud 11 November 2022].

[15] S. Stubailo, „GraphQL vs. REST,“ Apollo GraphQL Blog, 13 October 2021. [Võrgumaterjal]. Available: https://www.apollographql.com/blog/graphql/basics/graphql-vs-rest/. [Kasutatud 11 November 2022].

[16] „October 2021 Edition,“ GraphQL, 2021. [Võrgumaterjal]. Available: https://spec.graphql.org/October2021/. [Kasutatud 11 November 2022].

[17] „GraphQL Best Practices,“ The GraphQL Foundation, [Võrgumaterjal]. Available: https://graphql.org/learn/best-practices/. [Kasutatud 11 November 2022].

[18] „Tutorial - What is GraphQL?,“ Lighthouse, [Võrgumaterjal]. Available: https://lighthouse-php.com/tutorial/#what-is-graphql. [Kasutatud 11 November 2022].

[19] „What is GraphQL?,“ Amazon Web Services, [Võrgumaterjal]. Available: https://aws.amazon.com/graphql/. [Kasutatud 11 November 2022].

[20] „GraphQL kinda sucks,“ Hacker News, 06 August 2022. [Võrgumaterjal]. Available: https://news.ycombinator.com/item?id=32366759. [Kasutatud 11 November 2022].

[21] „Javascript With Syntax For Types,“ TypeScript, [Võrgumaterjal]. Available: https://www.typescriptlang.org/. [Kasutatud 11 November 2022].

[22] „Stack Overflow Trends,“ Stack Overflow, [Võrgumaterjal]. Available: https://insights.stackoverflow.com/trends?tags=reactjs,vue.js,angular,svelte,angularjs,vuejs3,next.js,nuxt.js. [Kasutatud 11 November 2022].

[23] „JavaScript Programming - The State of Developer Ecosystem in 2021,“ JetBrains s.r.o., 2021. [Võrgumaterjal]. Available: https://www.jetbrains.com/lp/devecosystem-2021/javascript/. [Kasutatud 11 November 2022].

[24] Atlassian Git Tutorial, „What is Git,“ Atlassian, [Võrgumaterjal]. Available: https://www.atlassian.com/git/tutorials/what-is-git. [Kasutatud 12 September 2022].

[25] „About MariaDB Server,“ MariaDB Foundation, [Võrgumaterjal]. Available: https://mariadb.org/about/. [Kasutatud 11 November 2022].

[26] C. Coronel ja S. Morris , „Why Database Design is Important,“ %1 *Database Systems: Design, Implementation, & Management, 12th Edition,* Cengage Learning, 2016, pp. 11-14.

[27] *Relatsiooniliste andmemudelite koostamise juhend v 1.0,* Tallinn: Riigi Infosüsteemi Amet, 2015.

[28] „Laravel 9.x - Eloquent: Getting Started - Events,“ Laravel LLC, [Võrgumaterjal]. Available: https://laravel.com/docs/9.x/eloquent#events. [Kasutatud 11 November 2022].

[29] „Laravel Guidelines,“ Follow Laravel naming conventions, [Võrgumaterjal]. Available: https://xqsit.github.io/laravel-coding-guidelines/docs/naming-conventions/. [Kasutatud 11 November 2022].

[30] „Queues - Dealing With Failed Jobs,“ Laravel LLC, [Võrgumaterjal]. Available: https://laravel.com/docs/9.x/queues#dealing-with-failed-jobs. [Kasutatud 11 November 2022].

[31] A. Wathan, „Cruddy by Design,“ Laracon US, 2017. [Võrgumaterjal]. Available: https://www.youtube.com/watch?v=MF0jFKvS4SI&t=334s. [Kasutatud 11 11 2022].

[32] Y. Katz, D. Gebhardt, G. Sullice ja J. Hanschke, „Latest Specification (v1.1) - Recommendations,“ JSON:API, [Võrgumaterjal]. Available: https://jsonapi.org/recommendations/.

[33] L. Gupta, „REST API Tutorial - REST Architectural Constraints,“ 9 March 2022. [Võrgumaterjal]. Available: https://restfulapi.net/rest-architectural-constraints/. [Kasutatud 11 November 2022].

[34] L. Gupta, „REST Resource Naming Guide,“ REST API Tutorial, 2021 November 2021. [Võrgumaterjal]. Available: https://restfulapi.net/resource-naming/. [Kasutatud 11 November 2022].

[35] „API testing with Postman,“ Postman, Inc., [Võrgumaterjal]. Available: https://www.postman.com/api-platform/api-testing/. [Kasutatud 11 November 2022].

[36] „Create React App,“ Facebook, Inc, [Võrgumaterjal]. Available: https://create-react-app.dev/. [Kasutatud 11 November 2022].

[37] „Adding TypeScript,“ Create React App, [Võrgumaterjal]. Available: https://create-react-app.dev/docs/adding-typescript/. [Kasutatud 11 November 2022].

[38] „Bootstrap - The most popular HTML, CSS, and JS library in the world.,“ [Võrgumaterjal]. Available: https://getbootstrap.com/. [Kasutatud 22 November 2022].

[39] „Promise based HTTP client for the browser and node.js,“ Axios, [Võrgumaterjal]. Available: https://axios-http.com. [Kasutatud 22 November 2022].

[40] „React Router - v6.4.3,“ Remix Software, Inc, [Võrgumaterjal]. Available: https://reactrouter.com/. [Kasutatud 22 November 2022].

[41] „JavaScript End to End Testing Framework,“ Cypress.io, 2022. [Võrgumaterjal]. Available: https://www.cypress.io. [Kasutatud 2022].

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

Mina, Robert Laursoo

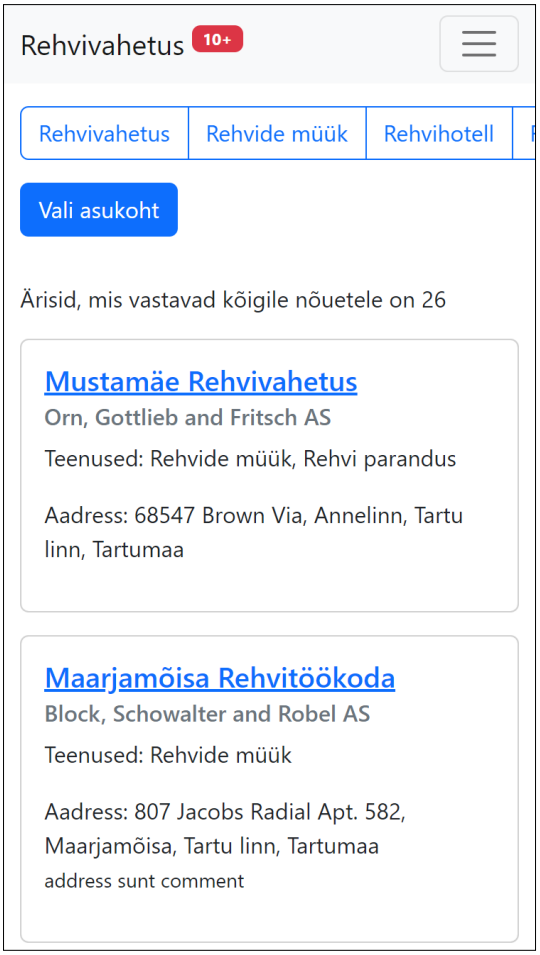
1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Veebirakenduse prototüübi loomine rehvitöökodade reklaamimiseks ja leidmiseks", mille juhendaja on Meelis Antoi
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli 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.

03.01.2023

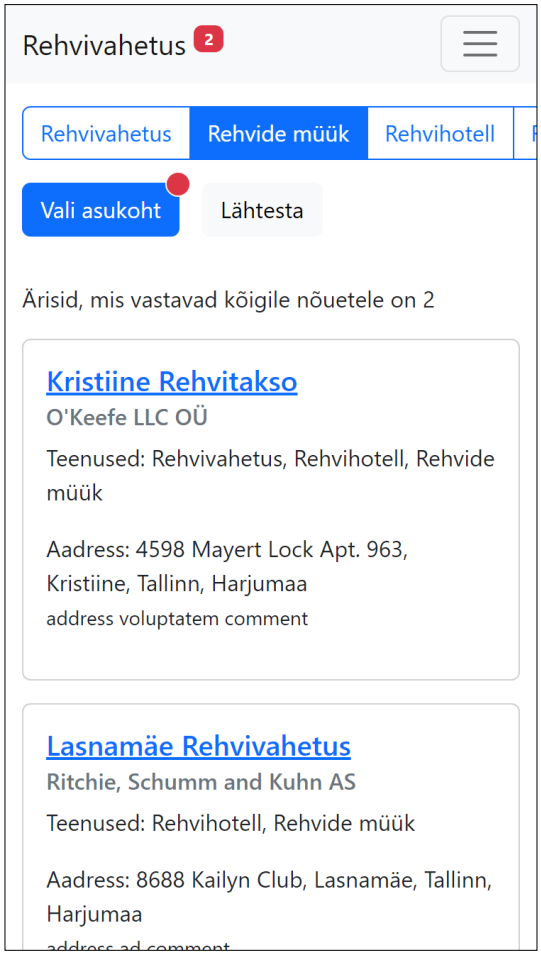
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või 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 – Rakenduse prototüübi avalehe vaated

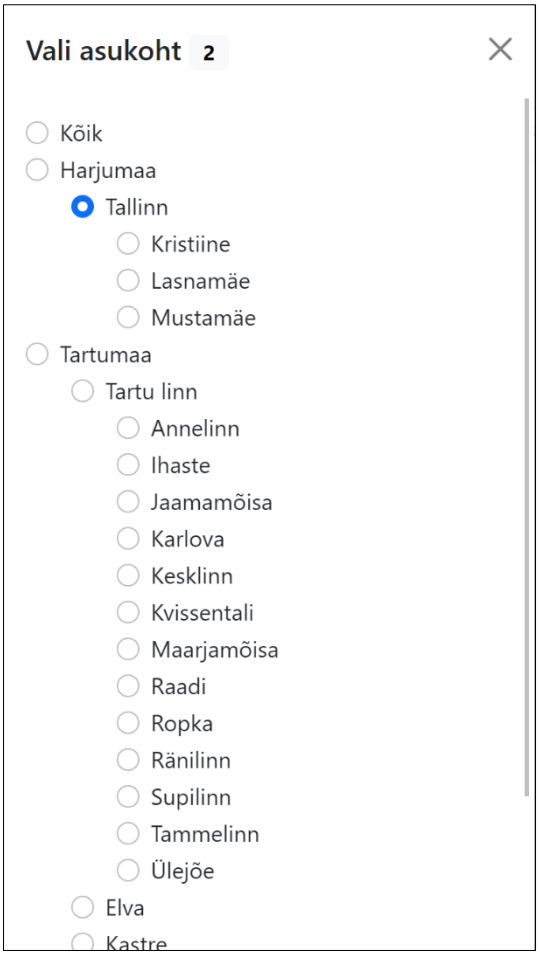
Rakenduse prototüübi vaated seadme iPhone SE ekraani suuruses (375 x 667 px).



Avalehe vaade ilma aktiivsete filtriteta.



Avalehe vaade aktiivse teenuse- ja asukohafiltriga.



Avatud asukohafiltri vaade.