TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl Markus Kõivastik 206662IADB

Ringmajandust toetav rentimisplatvorm

Bakalaureusetöö

Juhendajad: German Mumma
MSc
Lauri Anton
BSc

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: Karl Markus Kõivastik

15.05.2023

Annotatsioon

Käesoleva lõputöö probleemi esimene osa seisneb lineaarses majandusmudelis, mis ei ole enam paraku jätkusuutlik. Antud lõputöös käsitletakse ringmajanduse põhimõtteid, kui ühe võimaliku uue keskkonnasäästlikuma majandusmudelina, kus kesksel kohal on toodete rentimine, mitte nende omamine. Teine osa probleemist on kõigile ligipääsetava funktsionaalse ja kasutajasõbraliku rendiplatvormi puudumine erinevate toodete rentimiseks ja rendile andmiseks, mis igapäevast kasutust ei leia. Seetõttu jääb inimestel üle vaid konkreetne toode välja osta.

Probleemist tulenevalt on lõputöö eesmärkideks esiteks analüüsida nii funktsionaalseid kui ka tehnilisi võimalusi kõik ühes rendiplatvormi loomiseks, mis võimaldaks nii inimestel kui ka ettevõtetel vajaduspõhiselt tooteid rentida ja rendile anda toetades seejuures ringmajanduse põhimõtteid. Teiseks arendada analüüsile tuginedes valmis rendiplatvormi prototüüp, mida edasiarenduste käigus oleks võimalik ka avalikult lansseerida.

Lõputöö üheks tulemuseks on lahenduse analüüs, kus analüüsitakse olemasolevaid lahendusi, määratakse nõuded ning käsitletakse tehnoloogia valikut ja arhitektuurilisi otsuseid. Teise tulemusena valmib rendiplatvormi prototüüp, mis vastab analüüsis määratud nõuetele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 6 peatükki, 14 joonist, 8 tabelit.

Abstract

A Renting Platform Supporting Circular Economy

The first part of the problem addressed in this thesis is a linear economic model that is unfortunately no longer sustainable. This thesis examines the principles of the circular economy as one possible new, more environmentally sustainable economic model, where the focus is on renting products rather than owning them. Another part of the problem is the lack of a functional and user-friendly rental platform accessible to everyone for the rental and leasing of various products that are not in daily use. Therefore, people are left with the only choice of buying a specific product.

Due to the problem the objectives of the thesis are firstly to analyse the functional and technical possibilities of creating an all-in-one rental platform, which would allow people and businesses to rent and lease products according to their needs, while supporting the principles of the circular economy. Secondly, to develop a rental platform prototype based on the analysis, which could be further developed and launched publicly.

One of the outcomes of this thesis is the analysis of the solution, where existing solutions in Estonia are analysed, requirements are determined and, technology selection and architectural decisions are addressed. The second outcome is a prototype for a rental platform that meets the requirements identified in the analysis.

The thesis is in Estonian and contains 36 pages of text, 6 chapters, 14 figures, 8 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* – rakendusliides
BLL	*Business Logic Layer* – äriloogika kiht
CSS	*Cascading Style Sheets* – märgistuskeel, mida kasutatakse veebilehtede kujundamiseks
DAL	*Data Access Layer* – andmevahetuskiht
DTO	*Data Transfer Object* – andmete edastamise objekt
ERD	*Entity Relationship Diagram* – olemi-suhte diagramm
HTML	*HyperText Markup Language* – märgistuskeel, mida kasutatakse veebilehtede loomiseks
HTTPS	*HyperText Transfer Protocol Secure* – turvaline hüperteksti edastusprotokoll
IDE	*Integrated Development Environment* – integreeritud arenduskeskkond
JSON	*JavaScript Object Notation* – JavaScript programmeerimiskeelel põhinev andmevahetusvorming
ORM	*Object-Relational Mapping* – tehnika domeeni objektide vastendamiseks relatsiooniliste andmebaaside tabelitega
REST	*Representational State Transfer* – tarkvaraarhitektuuri stiil hajussüsteemide loomiseks
SDK	*Software Development Kit* – tarkvara arenduskomplekt
SPA	*Single Page Application* – üheleherakendus
UI	*User Interface* – kasutajaliides
URL	*Uniform Resource Locator* – veebiaadress
UX	*User Experience* – kasutajakogemus
WCAG	*Web Content Accessibility Guidelines* – veebisisu juurdepääsetavuse juhised

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeva lineaarne majandusmudel ei ole enam jätkusuutlik ning seetõttu on oluline välja töötada uusi tehnoloogiad ja võimalusi, mis toetaksid uuele keskkonnasäästlikumale majandusmudelile üleminekut. Jätkusuutliku majandusarengu tagamiseks vajame mudelit, mis suunaks loodusressursside tõhusamat kasutust läbi materjaliringluse [1]. Käesolevas lõputöös võetakse rendiplatvormi loomisel aluseks ringmajanduse põhimõtted, millega suunatakse inimesi keskkonnasäästlikumale tarbimisele ja toodete võimalikult kaua ringluses hoidmisele aidates seeläbi vähendada inimkonna sõltuvust tooraine suhtes.

Lõputöö eesmärgiks on luua kõik ühes rendiplatvormi prototüüp, mis võimaldaks kõigil tooteid rentida ja rendile anda toetades seejuures ringmajanduse mudelit. Rentijatel on võimalik toote väljaostmise asemel rentides raha kokku hoida ning rendileandjad saavad igapäevaselt mittekasutust leidvate toodete näol lisatulu teenida.

Töö teoreetilises osas kirjeldatakse probleemi olemust, seatakse lõputöö eesmärgid ja metoodika ning analüüsitakse olemasolevaid lahendusi, millest lähtudes koostatakse loodava rendiplatvormi analüüs. Analüüsi osas määratakse rakenduse funktsionaalsed ja mittefunktsionaalsed nõuded rendiplatvormi prototüübi loomiseks. Analüüsitakse ka lahenduse arhitektuuri ja tehnoloogia valiku võimalusi, mille juures üheks oluliseks kriteeriumiks on rakenduse skaleeritavus.

Praktilises osas kirjeldatakse rendiplatvormi serveri- ja kliendipoolse rakenduse arendusprotsessi, kus on lähtutud teoreetilise osa analüüsist. Samuti tuuakse välja platvormi olemi-suhte diagramm. Töö tulemusena valmib kõik ühes rendiplatvormi prototüüp, mis hõlmab endas funktsionaalset otsingulehte koos erinevate filtreerimisvõimaluste ja kaardivaatega, renditoodete lisamist ja broneerimist ning muid rendibroneeringute haldamisega seotud funktsionaalsusi.

2 Taust

Antud peatükis kirjeldatakse probleemi olemust ja seatakse paika lõputööga saavutatavad eesmärgid ning nende saavutamiseks kasutatav metoodika. Eesmärkidele tuginedes analüüsitakse olemasolevaid sarnaseid rendiplatvorme tuues välja nende eelised ja puudused, millele loodava lahenduse juures tähelepanu pööratakse.

2.1 Probleemi kirjeldus

Käesoleval ajastul on meie globaalne majandusmudel lineaarne, mis tähendab, et ressurss ammutatakse loodusest, tooraine töödeldakse ja sellest valmistatakse tooteid, mida tarbija tarbib mingi aja ning kõik jõuab lõpuks prügimäele [2]. Paraku antud mudel ei ole jätkusuutlik, kuna loodusressursid on maakeral piiratud ning pidev tarbimise ja ostujõu kasv viib tahes-tahtmata ühel hetkel loodusvarade kriisiga silmitsi. Sellest tulenevalt kujutab iga toote kasutamata jäänud osa endast potentsiaalset majanduslikku kahju ja keskkonnaprobleemi [3].

Käsitletavat probleemi silmas pidades on aina aktuaalsemaks muutunud just ringmajanduse põhimõtted ja nende rakendamine erinevatel viisidel, kus kesksel kohal on just toodete rentimine mitte nende omamine [4]. Ringmajanduse üheks sihiks on loodusressurssidest valmistatud toodete ringluses hoidmine võimalikult kaua. Vastavalt ÜRO Keskkonnaprogrammile *(UNEP)* võib 2050 aastaks inimkonna poolt tarbitavate mineraalide, maakide, fossiilkütuste ja biomassi tarbitav koguhulk kolmekordistuda ehk hinnanguliselt jõuda 140 miljardi tonnini, kui selle takistamiseks inimkond midagi ette ei võta [5]. Eelnevalt väljatoodud põhjustel ongi oluline luua uusi ringmajandusel põhinevaid ärimudeleid, mis toetaksid korduvkasutamist, toote eluea pikenemist ning juba valmistatud toote potentsiaali maksimaalselt ära kasutamist. Iga samm, millega vähendatakse loodusvarade ammutamise kiirust ja jäätmete hulka ning suurendatakse loodusvarade tootlikkust, ringlussevõttu ja taaskasutust, leevendab keskkonnale avaldatavat survet ja suurendab ökosüsteemi suutlikkust meid varustada [6].

Teine osa probleemist on kasutajasõbraliku ja funktsionaalse võimaluse puudumine toodete rentimiseks läbi ühtse platvormi, mis igapäevast kasutust ei leia. Seetõttu jääb inimestel üle vaid konkreetne toode välja osta, mis võib olla suur kulu ka rahakotile. Rendileandja vaatest on jällegi võimalused läbi ühtse platvormi toodete välja rentimiseks piiratud lisatulu teenimise eesmärgil.

2.2 Lõputöö eesmärk

Käesoleva lõputöö eesmärk on analüüsida võimalusi skaleeritava rendiplatvormi loomiseks, mis võimaldaks kõigil vajaduspõhiselt tooteid rentida ja rendile anda toetades seejuures ringmajanduse põhimõtteid. Teiseks arendada analüüsile tuginedes rendiplatvormi prototüüp, mida edasiarenduste käigus oleks võimalik ka avalikult lansseerida. Rakenduse lõplikuks eesmärgiks on täita kolme erinevat eesmärki kolmest erinevast vaatepunktist:

- rentija
- rendileandja
- keskkond

Rentija vaatepunktist on eesmärk luua võimalus kasutajasõbralikuks ja turvaliseks vahendite rentimiseks vajaduspõhiselt läbi ühtse platvormi. Selle tulemusena puudub rentijal vajadus konkreetne toode kohe välja osta, mis tõenäoliselt igapäevast kasutust ei leia.

Rendileandjate jaoks on eesmärk luua inimestele võimalus kodus igapäevaselt mittekasutust leidvate toodete väljarentimiseks toote potentsiaali maksimaalselt ära kasutamiseks ning seeläbi lisatulu teenimiseks. Samuti peab olema äriühingutel võimalus organiseeritud äritegevuse loomiseks läbi loodava platvormi.

Lõputöö eesmärgina on olulisel kohal ka keskkonnale avaldatava surve leevendamine toetades töö olemusega ringmajanduse mudelit hoides juba valmistatud tooteid ringluses võimalikult kaua. Samuti ringmajanduse ja keskkonnamõju teadlikkuse suurendamine läbi loodava rendiplatvormi.

2.3 Metoodika

Lahenduse väljatöötamisel esmalt kirjeldatakse probleemi olemust, millest tulenevalt seatakse ka lõputööga saavutatavad eesmärgid. Eesmärkide põhjal analüüsitakse olemasolevate lahenduste eeliseid ja puudujääke, et määrata loodava lahenduse peamised mittefunktsionaalsed ning funktsionaalsed nõuded.

Nõuete määramise järel analüüsitakse erinevaid tehnoloogiaid ja veebirakenduse arhitektuuri võimalusi. Seejärel põhjendatakse valikuid arvestades muu hulgas autori eelnevat kogemust erinevate tehnoloogiatega, platvormi nõudeid ja võimalikke edasiarendusi.

Järgnevalt arendatakse analüüsile ja nõuetele tuginedes rakenduse prototüüp ning seejuures kirjeldatakse ka arendusprotsessi. Arendusmeetodina kasutatakse agiilset tarkvaraarenduse metoodikat *Scrum*. Rendiplatvormi realiseerumisel kasutatakse objektiivsemaks võrdlemiseks analüütiliste hierarhiate ehk *Saaty* meetodit, et hinnata loodud lõpplahendust.

2.4 Olemasolevad lahendused

Teadaolevalt on Eestis olemas kaks sarnast rendiplatvormi lahendust, mille eeliseid ja puudusi järgnevalt täpsemalt analüüsitakse.

Analüüsitakse järgmisi olemasolevaid platvorme:

- Rentif - https://rentif.com/
- Forent - https://forent.ee/

2.4.1 Rentif

Töö kirjutamise hetkel on sarnaste kõik ühes rendiplatvormide seast Eestis kindel turuliider Rentif. Platvorm võimaldab kõigil ennast kasutajaks registreeruda ning sisse logida. Sisseloginuna on kasutajal võimalik rendile antava toote kuulutus lisada täites selleks ära vastava vormi, mis läbib ka verifitseerimisprotsessi. Samuti on võimalik soovitud toodet koheselt rentida koos kohese maksmisvõimalusega ning kogu rentimise protsess on seejuures kasutajasõbralik ja võimalikult väikeste klikkide arvuga. Rendikogemuse kohta on võimalik rendileandjale jätta tagasiside, mida on võimalik teistel rendihuvilistel avalikult näha.

Platvorm on selge ülesehituse ja struktuuriga, kuid kasutajaliidese disain on kohati konarlik ja põhilised disainiprintsiibid on jäetud tähelepanuta. Otsingulehe filtrid ei ole piisavad või on kasutaja seisukohast kasutud. Näiteks puudub võimalus filtreerida renditavaid tooteid alamkategooriate kaupa ning nende saadaval olevate kuupäevade järgi. Üheks puuduseks on ka interaktiivne kaardivaade, mis kuvaks otse kaardi peal renditooteid koos kaasaskäiva olulise infoga.

2.4.2 Forent

Teiseks analüüsitavaks platvormiks on Forent, mis täidab oma eesmärki rohkem kataloogina kui rendiplatvormina. Sisseloginud kasutajal on võimalik ainult lisada kuulutusi toodete väljarentimiseks ning rentijal tuleb rentimiseks rendileandjaga ise kontaktandmete abil ühendust võtta. Rendileandjate kohta puudub võimalus kirjutada arvustusi rendikogemuse kohta.

Eelisena võib antud lahenduse juures välja tuua vahendustasu puudumise tehingutelt, kuid teisalt toob taoline rentimisprotsess endaga kaasa omad riskid. Tulu teenitakse veebilehel kuvatavate reklaambännerite ja kuulutuste esiletõstmise võimalusega hinnakirja alusel. Kasutajaliidese- ja kogemuse disain on keskmisel tasemel ning otsingulehel on võimalik tooteid filtreerida ka alamkategooriate kaupa.

Kokkuvõte Eestis tegutsevatest kõik ühes rendiplatvormidest on välja toodud tabelis 1. Märk „+“ tähendab funktsionaalsuse olemasolu ja „-“ puudumist platvormil.

Tabel 1. Olemasolevate lahenduste võrdlustabel.

	Rentif	Forent
Ringmajanduse teadlikkuse suurendamine	-	-
Registreerumine ja sisse logimine	+	+
Läbi platvormi rentimine ja kohene maksmine	+	-
Filtreerimine alamkategooriate kaupa	-	+
Filtreerimine saadaolevate kuupäevade järgi	-	-
Otsingulehe kaardivaade	-	-
Rendileandjale tagasiside jätmine	+	-

Olemasolevate lahenduste analüüsist selgus, et mõlema platvormi puhul on puudulik ringmajanduse teadlikkuse suurendamine, piiratud filtreerimisvõimalused ning otsingulehe interaktiivne kaardivaade, mis võimaldaks tooteid otsida kaardil orienteerudes. Samuti esines kasutajakogemuse- ja liidese disainil ebaühtlust ning ebakõla esteetilisuse ja funktsionaalsuse vahel. Analüüsi tulemusi võetakse arvesse järgnevas peatükis lahenduse analüüsis.

3 Lahenduse analüüs

Antud peatükis määratakse loodava rakenduse funktsionaalsed ja mittefunktsionaalsed nõuded. Analüüsitakse ka erinevaid rakenduse arhitektuuri ning tehnoloogia valiku võimalusi. Samuti tehakse ülevaade, millest kliendipoolse rakenduse kasutajakogemuse disainil lähtutakse.

3.1 Nõuete määramine

Rakenduse nõuded on justkui spetsifikatsiooniks sellest, mida tuleb rakenduse valmimiseks implementeerida [7]. Nõuded jagunevad laias laastus funktsionaalseteks ja mittefunktsionaalseteks, mis kirjeldavad mida süsteem peaks tegema ja kuidas mingis kindlas olekus käituma. Need aitavad nii kliendil kui ka arendajal paremini aru saada, mida loodavast tarkvarast oodatakse ning loovad aluse tarkvara edukaks disainimiseks, arendamiseks, testimiseks ja juurutamiseks.

3.1.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded kirjeldavad süsteemi käitumist kindlatel tingimustel lõppkasutaja jaoks [7]. Platvormi funktsionaalsed nõuded on grupeeritud kasutajarollide kaupa eepostesse, mis omakorda jagunevad väiksemateks kasutajalugudeks. Eepos on suurem ja laiem tööühik, mida on võimalik jagada väiksemateks osadeks ehk kasutajalugudeks, mis on kirjeldatud nõuetena lõppkasutaja perspektiivist [8]. Nõuete määramisel lähtuti eelkõige olemasolevate lahenduste analüüsist, autori enda poolt kogutud ideedest ning töö eesmärkidest.

Loodava platvormi kasutajad jagunevad kahte kasutajarolli:

- rentija
- rendileandja

Alljärgnevalt on väljatoodud mõlema kasutajarolli põhilised funktsionaalsed nõuded, millest loodava rendiplatvormi prototüübi loomisel lähtutakse. Tabelis 2 on väljatoodud rentija kasutajarolli eeposed ja kasutajalood.

Tabel 2. Rentija eeposed ja kasutajalood.

ID	Roll	Tegevus	Eesmärk
1	Rentijana	soovin otsida renditavaid tooteid,	et leida endale sobiv renditoode.
1.1	Rentijana	soovin otsida tooteid tootenime järgi,	et leida toode selle nimetuse alusel.
1.2		soovin filtreerida tooteid asukoha järgi,	et leida tooteid kindlas asukohas.
1.3		soovin filtreerida tooteid kategooriate ja alamkategooriate järgi,	et leida tooteid kindla kategooria alusel.
1.4		soovin filtreerida tooteid soovitud rendikuupäevade järgi,	et leida tooteid, mis on saadaval valitud kuupäevadel.
1.5		soovin otsida tooteid kasutades kaardivaadet,	et leida lähedal asuvaid tooteid otse kaardilt.
2	Rentijana	soovin näha toote detailvaadet,	et saada rohkem informatsiooni renditava toote kohta.
2.1	Rentijana	soovin näha pilte renditootest,	et saada visuaalne ülevaade renditavast tootest.
2.3		soovin näha renditoote asukohta,	et teada kus renditav toode asub.
2.4		soovin näha renditoote kirjeldust,	et saada täpsemat informatsiooni toote kohta.
2.5		soovin näha kalendrivaatest saadaolevaid kuupäevi,	et näha saadaolevaid rendikuupäevasi.
3	Rentijana	soovin esitada tellimuse valitud tootele,	et kindlustada endale renditoode soovitud kuupäevadeks.
3.1	Rentijana	soovin valida rendikuupäevad,	et valida soovitud rendiperiood.
3.2		soovin täpsustada toote kogust,	et rentida toodet vastavalt soovitud kogusele.
3.3		soovin näha rentimise hinna arvestust,	et näha mille eest ja kui palju maksta tuleb.
3.4		soovin maksta rentimise eest pangalingi või krediitkaardiga,	et kinnitada broneering.

Tabelis 3 on välja toodud rendileandja kasutajarolli eeposed ja kasutajalood. Kuna ühe platvormi kasutaja alt on võimalik tooteid nii rentida kui ka rendile anda, siis kehtivad rendileandjale samad nõuded, mis rentijale ning vastupidi. Nõuete struktureerimise huvides on tehtud siiski kaks eraldi kasutajarolli.

Tabel 3. Rendileandja eeposed ja kasutajalood.

ID	Roll	Tegevus	Eesmärk
1	Rendileandjana	soovin lisada rendile antava toote kuulutuse,	et toode oleks klientidele leitav ja renditav.
1.1	Rendileandjana	soovin lisada pilte tootest,	et kliendid saaksid visuaalse ülevaate tootest.
1.2		soovin lisada toote asukoha,	et kliendid teaksid kus renditav toode asub.
1.3		soovin lisada toodete arvu,	et kliendid teaksid mitu eksemplari on võimalik rentida.
1.4		soovin lisada toote pealkirja ja kirjelduse,	et anda kliendile tootest tekstikujulist informatsiooni.
1.5		soovin lisada hinnad erinevate rendiperioodide kaupa,	et süsteem saaks arvutada rendi maksumuse.
2	Rendileandjana	soovin näha broneeringute nimekirja,	et oleks võimalik saada ülevaade broneeringutest.
2.3	Rendileandjana	soovin filteerida broneeringuid kuupäevade vahemiku ja renditava toote järgi,	et oleks võimalik leida broneeringuid kindlatel tingimustel.
2.4		soovin sorteerida broneeringuid kuupäeva alusel,	et oleks võimalik näha sissetulevaid broneeringud ajalises järjestuses.
3	Rendileandjana	soovin näha broneeringu detailvaadet,	et näha täpsemat informatsiooni ja uuendada staatust.
3.1	Rendileandjana	soovin näha klikitava lingina renditavat toodet,	et näha millise tootega on broneering seotud.
3.2		soovin näha rendiperioodi ja kogust,	et näha mis kuupäevadel rendiperiood algab ja lõppeb ning mis koguses.
3.4		soovin näha ja uuendada broneeringu staatust,	et hallata broneeringu staatust.

3.1.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded kirjeldavad süsteemi omadust või piirangut, mida süsteem peab järgima [7]. Need on olulised tagamaks süsteemi kvaliteedi, kasutatavuse ja turvalisuse. Tabelis 4 on väljatoodud rakenduse mittefunktsionaalsed nõuded arvestades ka antud töö skoopi.

Tabel 4. Rakenduse mittefunktsionaalsed nõuded.

ID	Mittefunktsionaalne nõue
1	Serveripoolne rakendus peab edastama infot läbi API liidese, mis peab olema dokumenteeritud.
2	Nii serveripoolne kui ka kliendipoolne rakendus peavad kasutama HTTPS protokolli.
3	Veebipõhine kasutajaliides peab vastama HTML 5, CSS 3 ja JavaScripti standarditele.
4	Veebipõhine kasutajaliides peab olema kohanduv erinevatele seadmetele.
5	Veebipõhine kasutajaliides peab olema implementeeritud ühe lehekülje rakendusena (SPA)
5	Kasutajaliides peab jälgima WCAG 2.1 juurdepääsetavuse juhiseid.
6	Rakendus peab olema kasutatav enamlevinud brauserites.
7	Rakenduse koodibaas peab olema versioneeritud.
8	Arendusel tuleb jälgida DRY ja SOLID printsiipe.

3.2 Kasutajakogemuse disain

Rakenduse arendamisega võrreldes on kasutajakogemuse disain omaette maailm, kus kehtivad omad standardid, tavad ja lähenemised. Kuid samuti on see lahutamatuks osaks iga rakenduse puhul, kus lõppkasutaja rollis on inimene. Käesolevas peatükis tehakse ülevaade, millest täpsemalt loodava lahenduse disainimisel lähtutakse.

Igal elemendil on mingil kujul kasutajakogemus sõltumata sellest, kas see on tehtud eesmärgi päraselt või mitte. UX (*User Experience*) disaini ülesandeks ei ole luua kasutajakogemust ennast, vaid teha see lõppkasutaja jaoks võimalikult efektiivseks [9].

See määrab ära lõppkasutaja jaoks saadud kogemuse rakendust või toodet kasutades. Kui kasutaja ei ole rahul saadud kogemusega, ei pruugi ta enam teist korda antud teenust kasutada. Seetõttu on ärilisest seisukohast oluline panustada ka rakenduse disaini. Rakendus ise võib arhitektuuri, tehnoloogiavaliku ja skaleeritavuse seisukohast olla küll väga heal tasemel, kuid kui halva kasutajakogemuse tõttu klient enam järgmine kord teenust ei kasuta, siis äri toimida ei saa. Järgnevalt on välja toodud viis UX disaini põhimõtet, millest käesoleva töö kasutajaliidese loomisel lähtutakse [10].

- **Kasutajakesksus** – kasutajakogemuse disaini eesmärk on luua efektiivne toode või teenus, mis lahendab kasutaja jaoks mingit kindlat probleemi. Kasutajakesksus seejuures tähendab seda, et otsuseid tehakse lõppkasutaja vajadustest lähtuvalt.

- **Järjepidevus** – rakenduse disain ja funktsioonid peavad olema läbivalt sarnased kõikidel lehtedel ja vaadetel. Samuti tuleb arvestada sihtkasutaja eelduste ja kogemustega, mis on saadud teiste sarnaste rakenduste kasutamisel. See tagab kasutaja jaoks kergesti kasutatava ja õpitava rakenduse.

- **Hierarhilisus** – nii informatsiooni kui ka visuaalse ülesehituse struktuur peab olema hierarhiline. Reeglina on kasutajale tähtsamad elemendid nähtavamal kohal, kuna tänu sellele on kasutajal lihtsam neid üles leida.

- **Juurdepääsetavus** – rakendus peab olema ligipääsetav ja kasutatav nii paljudele inimestele kui võimalik, sinna hulka kuuluvad näiteks erivajadustega inimesed. Alljärgnevas osas on lähemalt kirjeldatud juurdepääsetavuse juhiseid.

- **Kasutatavus** – Hea kasutajakogemuse disainiga rakendus peaks olema lihtsasti kasutatav, õpitav, efektiivne, meeldejääv ja rahulolu pakkuv.

Üheks mittefunktsionaalseks nõudeks on veebipõhises kasutajaliideses WCAG (*Web Content Accessibility Guidelines*) ehk juurdepääsetavuse juhiste 2.1 versiooni jälgimine. Juhised on koostatud W3C (*World Wide Web Consortium*) organisatsiooni poolt, mis keskendub ka erivajadustega inimestele veebi sisu juurdepääsetavaks muutmisele erinevate standardite loomise näol. WCAG jaguneb eri olukordade rahuldamiseks kolmeks tasemeks: A (madalaim), AA (keskmine) ja AAA (kõrgeim). Käesoleva lõputöö raames kõikide A või AA taseme nõuete täitmine töö skoopi ei mahu, kuid järgnevalt on

välja toodud WCAG juhistes kirjeldatud neli juurdepääsetavuse printsiipi, mida antud lahenduse kasutajaliidese loomisel arvesse võetakse [11].

- *Perceivable* ehk **tajutav** – teave ja kasutajaliidese komponendid peavad olema esitatud kasutajale viisil, mida on võimalik tajuda. Selle alla kuulub tekstiline alternatiiv, kohandatavus ja eristatavus. Näiteks peab HTML-i pildi elemendile alati olema lisatud *alt* atribuut ehk tekstikujuline alternatiiv, mis võimaldab ka vaegnägijatel veebi sisust aru saada. Eristatavuse koha pealt ei tohi kasutada värvi ainsa visuaalse teabe edastamise vahendina ning värvide kontrastsus peaks olema vähemalt 4,5:1. Näiteks vea esinemisel ei tohi infot edasi anda pelgalt punase värvikasutusega, vaid juurde peab olema lisatud ka tekstikujuline informatsioon.

- *Operable* ehk **talitlusvõimeline** – kasutajaliidese komponendid ja navigeerimine peavad olema talitletavad. Sisu funktsioonid peavad olema kasutatavad klaviatuuri abil ning kasutajale tuleb anda piisavalt aega sisu lugemiseks ja kasutamiseks. Samuti tuleb kasutajale tagada võimalused navigeerimiseks, sisu leidmiseks ja oma asukoha määramiseks.

- *Understandable* ehk **mõistetav** – selle põhimõtte kohaselt peab teksti sisu olema loetav, ettearvatav ja mõistetav. Näiteks ei tohiks komponendi fokuseerimine tuua endaga kaasa konteksti muutust ning vea esinemise korral tuleks anda juhised selle parandamiseks.

- *Robust* ehk **töökindel** – tagada tuleb võimalikult suur ühilduvus praeguste ja tulevaste brauseritega. Selleks on oluline jälgida HTML, CSS ja JavaScript standardeid, mis on ka rakenduse mittefunktsionaalsetes nõuetes välja toodud.

3.3 Arhitektuur

Tarkvaraarhitektuur on struktureeritud raamistik, mida kasutatakse tarkvara elementide, suhete ja omaduste kontseptualiseerimiseks [12]. Tarkvara juurutamisel on üheks olulisemaks alustalaks loodava tarkvara arhitektuur kohe laiemas pildis läbi mõelda, kuna hilisemad tarkvara arhitektuurilised muudatused võivad osutuda väga ressursimahukaks. Head tarkvaralahendused on jätkusuutlikud ning tuleb aru saada sellest, et tarkvara ei saa olla jätkusuutlik ilma hea arhitektuurita [13]. Hea ja läbimõeldud tarkvaraarhitektuur tagab pikemas perspektiivis rakenduse skaleeritavuse ja hooldatavuse. See tähendab, et

tarkvara on kättesaadav ajas aina suurenevale kasutajate arvule ning muudatuste tegemine koodibaasis ja uute arenduste loomine on oluliselt kiirem ja lihtsam.

Käesoleva rendiplatvormi mittefunktsionaalsetes nõuetes on väljatoodud, et serveripoolne rakendus peab edastama infot läbi API liidese. Tuleviku vaatest loob see võimaluse lisaks veebipõhisele kliendirakendusele arendada näiteks ka mobiilirakenduse ilma suuremate lisaarendusteta serveripoolses rakenduses. Sellest tulenevalt peab loodav terviklahendus olema teostatud hajussüsteemina (*distributed system*). Antud mustri puhul eksisteerivad lahenduse erinevad osad nagu kliendipoolne rakendus, serveripoolne rakendus ja andmebaas eraldiseisvalt. Hajussüsteemi tüüpe on mitmeid, kuid arvestades töö skoopi ja esmast kasutajate arvu platvormil on mõistlik kasutusele võtta kõige lihtsakujulisem variant ehk klient-server muster. See tähendab, et arhitektuur on jagatud kaheks põhiliseks vastutuseks [14]. Kliendipoolse rakenduse vastutus on kasutajaliides, mis teeb läbi REST API päringuid serveripoolsele rakendusele JSON kujul info saamiseks. Serveripoolse rakenduse vastutuseks on äriloogika, andmebaasiga suhtlemine ja päringutele vastuste saatmine. Joonisel 1 on visuaalselt väljatoodud loodava süsteemi pealiskaudne arhitektuur.

Joonis 1. Loodava süsteemi arhitektuur.

Mittefunktsionaalsetes nõuetes on veel kirjas, et rakenduse arendusel tuleb jälgida DRY ja SOLID tarkvaraarendus printsiipe, mis toetavad tarkvara head arhitektuuri. Nagu igal teiselgi teadusharul, on ka heal IT-süsteemide arendamisel reguleerivad aluspõhimõtted, millest eelnevalt väljatooduid põhimõtteid kirjeldatakse [15].

DRY printsiip lahti kirjutatult tähendab inglise keeles *Don't Repeat Yourself* ja selle kohaselt ei tohiks ühte ja sama tegevust väljendavat koodi eksisteerida kahes või enamas kohas [16]. Vastasel korral võib juhtuda, et kui tekib vajadus ühes kohas muudatus sisse

viia, siis võib teises kohas muudatuse tegemine ära ununenda, mis on murettekitav rakenduses stabiilsuse osas.

Teine põhimõte SOLID koondab raamatu „*Agile, Principles, Patterns, and Practices in C#*“ põhjal endasse viis tarkvaraarenduse printsiipi, mis on rakendatavad sisuliselt igasse rakenduse kihti nagu näiteks andmevahetuse või äriloogika kihti [17].

- ***S*** – *Single Responsibility Principle* ehk ainuvastutuse põhimõte ütleb, et klassil või moodulil peab olema ainult üks konkreetne vastutus. Kui klassil on rohkem kui üks vastutus, on kohustused seotud ning muudatused ühes vastutuses võivad hakata segama teist vastutust.

- ***O*** – *Open/Closed Principle* ehk avatuse-suletuse põhimõtte kohaselt peavad tarkvara osad olema laiendatavad, kuid mitte muudetavad. Tänu sellele on võimalik uut funktsionaalsust lisada ilma olemasolevat koodi muutmata.

- ***L*** – *Liskov Substitution Principle* ehk Liskovi asenduspõhimõte. Selle põhimõtte kohaselt peaksid superklassid olema asendatavad selle alamklassidega ilma, et rakendus katki läheks.

- ***I*** – *Interface Segregation Principle* ehk liidese eraldamise põhimõte kohaselt on parem eelistada väiksemaid grupeeritud liideseid kui ühte suuremahulist liidest.

- ***D*** – *Dependency Inversion Principle* ehk sõltuvuste inversiooni põhimõte ütleb, et kõrgema taseme moodulid ei tohiks sõltuda madalama taseme moodulitest. Mõlemad peaksid toetuma polümorfismile ja abstraktsioonidele.

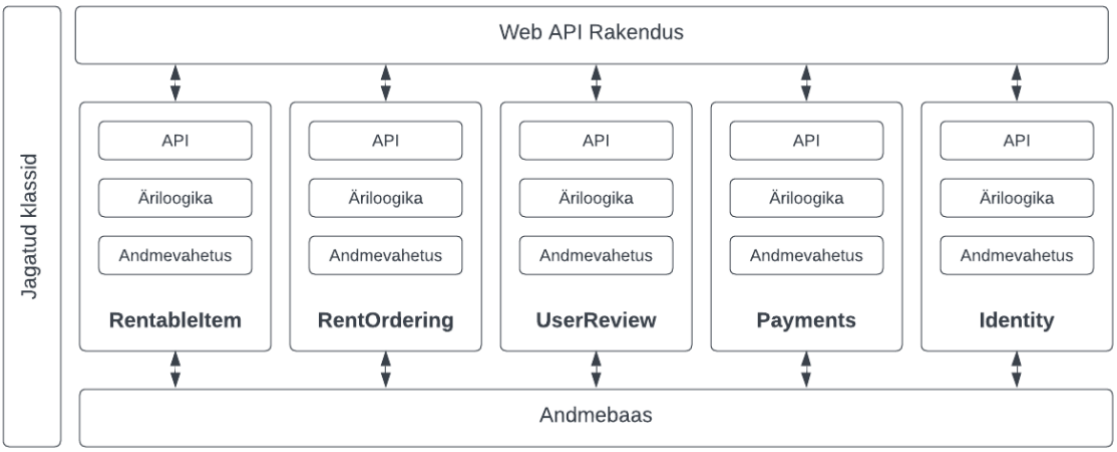
3.3.1 Serveripoolne rakendus

Serveripoolse rakenduse arendamisel võetakse aluseks modulaarse monoliidi arhitektuuri põhimõtted. Koodibaas on jagatud loogiliselt erinevatesse moodulitesse, mida on võimalik arendada, testida ja hooldada üksteisest sõltumatult [18]. Moodul on ärinõuetest tulenev loogiline ühik [19] ehk käesoleva töö nõuete põhjal on esialgseteks mooduliteks:

- renditoode (*RentableItem*)
- renditellimus (*RentOrdering*)
- kasutajate tagasiside (*UserReview*)

- maksed (*Payments*)
- identiteet (*Identity*)

Iga moodul sisaldab endas omakorda kihilist arhitektuuri ehk kood on struktureeritud andmevahetus, äriloogika ja API kihtideks. Selline lähenemine loob kohe alguses hea aluse rakenduse skaleeritavusele ning lihtsustab tulevikus oluliselt uute moodulite lisamist või vajadusel mikroteenustele üleminekut. Oluline kasutegur väljendub ka arendajate efektiivsuses ja koostöös, kuna arendajate vastutusi on võimalik jagada erinevate moodulite vahel ning seetõttu konfliktide tekkimise võimalus on minimeeritud võrreldes tavapärase monoliit rakendustega [18]. Joonisel 2 on visuaalselt väljatoodud serveripoolse rakenduse arhitektuur.



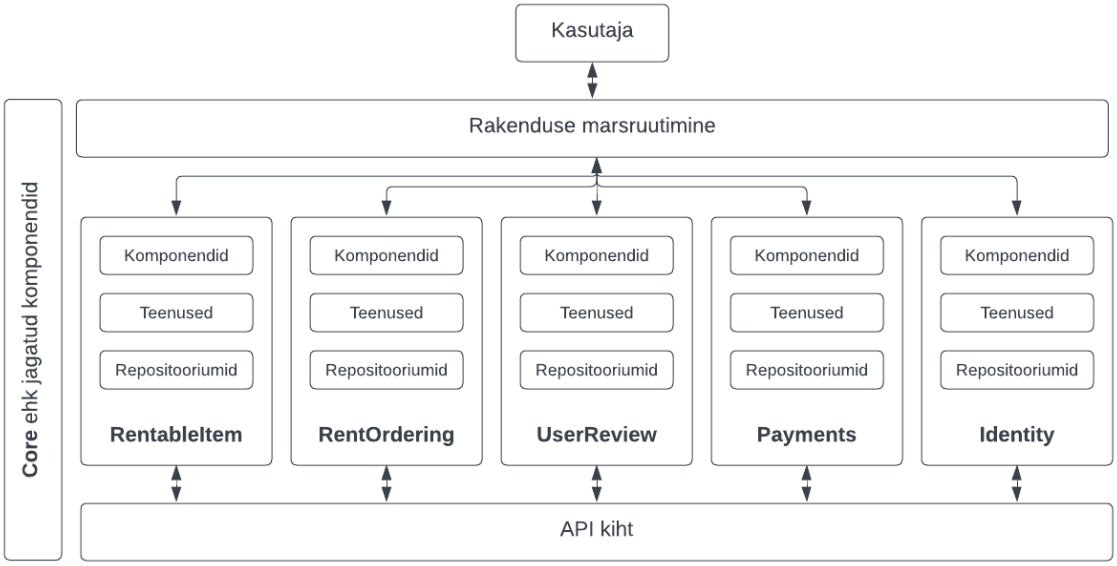
Joonis 2. Serveripoolse rakenduse arhitektuur.

3.3.2 Kliendipoolne rakendus

Kliendipoolse rakenduse arendamisel lähtutakse samuti modulaarse monoliidi arhitektuurist ning ka komponendipõhise arhitektuuri mustrist. Koodibaas on jagatud mooduliteks sarnaselt serveripoolsele rakendusele. Moodulite siseselt jälgitakse komponendipõhist arhitektuuri ehk kasutajaliides ehitatakse üles väikestest taaskasutatavatest komponentidest [20].

Olulisteks märksõnadeks on ka siinkohal skaleeritavus ja taaskasutatavus, millest töö autor on arhitektuuriliste otsuste tegemisel põhiliselt lähtunud. Igas moodulis sisalduvad eraldi sellekohased repositooriumid, teenused ja domeeni objektid. Moodulid kasutavad serveripoolsele rakendusele päringute tegemiseks API kihti. Põhimoodulite kõrvale

luuakse ka tuumik (*Core*) moodul, mis sisaldab endas üldkasutatavaid komponente nagu näiteks nupud või vormid. Joonisel 3 on visuaalselt väljatoodud kliendipoolse rakenduse arhitektuur.



Joonis 3. Kliendipoolse rakenduse arhitektuur.

3.4 Tehnoloogia valik

Tänasel päeval on rakenduste arendamiseks saadaval palju erinevaid tehnoloogiad, millest igaühel on omad eelised ja puudused. Sellest tulenevalt võib parima võimaliku variandi valimine osutuda üpris keeruliseks ülesandeks. Hea tehnoloogiline pagas on selline, mis skaleerub kiiremini kui tarkvara ülalpidamiseks kuluvate inimeste arv [21].

Tehnoloogia valikul tuleks esmalt lähtuda rakenduse nõuetest ja arhitektuurist. Kuna rakendus peab valmima hajussüsteemina, siis tuleb tehnoloogia valida nii kliendi- kui ka serveripoolsele rakendusele. Teise aspektina lähtutakse tehnoloogia valikul autori eelneva kogemuse ja eelistustega, mis on saadud Tallinna Tehnikaülikooli IT-süsteemide arenduse õppekaval õpituga või töökogemusega. Lõputöö skoop seab ajalised piirangud ning seetõttu on põhjendatud valida kasutatavaks tehnoloogiaks see, millega ollakse juba varasemalt kokku puutunud ning mis ei vaja enam nullist õppimist, vähendades nii juurdeõppimisele kuluvat aega. Samuti tuleb tehnoloogia valikul arvestada selle eeliseid ja puudusi, skaleeritavust, turvalisust ning üldist populaarsust tarkvaraarendus maailmas.

3.4.1 Serveripoolse rakenduse tehnoloogia valik

Autor on töötamise ja õpingute ajal serveripoolsete rakenduste arendamisel kokku puutunud järgnevate tehnoloogiatega: PHP ja Laravel raamistik, C# ja .NET Core raamistik ning Java ja Spring raamistik. Eelnevalt väljatoodud tehnoloogiatest on töö autoril kõige rohkem kogemust esimese kahega ning sellest tulenevalt on autori hinnangul mõistlik kitsendada tehnoloogia valikut just PHP ja Laravel ning C# ja .NET Core kombinatsioonidele. Kuna raamistiku valik sõltub eelkõige programmeerimiskeelest, siis järgnevalt on lühidalt analüüsitud eelmainitud keeli.

PHP on üks populaarseim serveripoolne programmeerimiskeel, mida kasutatakse koguni enam kui 77% veebilehtedest ühel või teisel moel [22]. Üheks populaarsuse põhjuseks on selle lihtsasti õpitavus ja kasutatavus ning seetõttu on arendusprotsess küllaltki kiire ning kuluefektiivne [23]. Teisalt on nõrkuseks PHP puhul see, et tegemist on dünaamiliselt ehk nõrgalt tüübitud ning interpreteeritud keelega. Interpreteeritud keel on enamasti aeglasem võrreldes kompileeritud keelega ja vigade leidmine ning silumine (*debug*) on raskendatud, kuna lähtekood transleeritakse interpretaatori poolt alles käitusajal [24]. PHP on autori hinnangul küll hästi sobilik kuni keskmise suurusega veebilehtede ja e-poodide arendamiseks, kuid kompleksete, suuremahuliste ja mitmete arendajatega rakenduste puhul võivad pikemas perspektiivis tekkida erinevad keelest tingitud probleemid ja piirangud.

C# on 2000. aastal Microsofti poolt loodud staatiliselt ehk tugevalt tüübitud objektorienteeritud keel, mis võimaldab arendada rakendusi väga erinevatele seadmetele nagu veebiplatvormidele, mobiilidele, arvutitele ja televiisoritele. [25]. Töö autori jaoks on oluline, et valitav programmeerimiskeel oleks tugevalt tüübitud ning seetõttu on antud keelel koheselt selge eelis. Stackoverflow 2022 uuringu [26] alusel kasutas arendamisel ligikaudu 28% vastanutest just C# programmeerimiskeelt. Antud keel on usaldatud paljude ettevõtete poolt, mis on nõudlikud jõudluse ja turvalisuse osas alustades pangandusest kuni suurte e-kaubanduse platvormideni välja, töödeldes tohutus hulgas päringuid ja tehinguid [25]. Samuti on tegemist kompileeritud keelega, mis loob hea aluse kiirele ja tõhusale rakendusele.

Võttes arvesse eelnevat analüüsi ja loodava lahenduse tuleviku perspektiivi, kus andmemaht võib kujuneda väga suureks ja võidakse kaasata erinevaid liidestusi või näiteks ka tehisintellektil põhinevaid lahendus, siis on autori hinnangul mõistlik PHP

valitava serveripoolse programmeerimiskeelena kõrvale jätta ning otsustada C# ja .NET Core raamistiku kasuks.

3.4.2 Kliendipoolse rakenduse tehnoloogia valik

Käesoleva töö kliendipoolne lahendus saab olema veebipõhine rakendus, mille puhul põhilisteks kasutatavateks tehnoloogiateks on HTML, CSS ja JavaScript. Suurem küsimus seisneb JavaScripti lihtsamaks ja sujuvamaks arendamiseks valitava raamistiku ja nendes sisalduvate sõltuvuste osas. Samuti on mittefunktsionaalsetes nõuetes kirjas, et kliendipoolne rakendus peab olema SPA ehk *Single Page Application* tüüpi, millele raamistiku kasutuselevõtt hea aluse loob. Stackoverflow 2022 uuringu [26] järgi on populaarsemateks veebipõhiste rakenduste arendamise raamistikkudeks Node.js, React.js, jQuery, Express, Angular ja Vue.js. Töö autoril on eelnimetatutest kõige rohkem kogemust React.js raamistikuga. Seetõttu on mõistlik kliendipoolse rakenduse tehnoloogia valik langetada Reactile, et rakenduse valmimisel ei kuluks vähemalt kahekordselt rohkem aega juurdeõppimisele. Reacti kasuks räägivad veel mitmed selle head omadused, mis sobituvad loodava lahenduse visiooniga ning mida järgnevalt lähemalt analüüsitakse.

Facebooki loodud React.js on ülal väljatoodud raamistike seast üks enimkasutatav, mille taga on suur kommuun ning sellest tulenevalt võib töö käigus esinevate probleemide korral lahenduse leidmine olla tänu sellele kiirem. Samuti on saadaval suur hulk kommuuni poolt loodud erinevaid sõltuvusi ja tööriistu. Arhitektuuri poole pealt toetab React ka peatükis 3.3.2 väljatoodud komponendipõhise mustri nõuet, mis tagab komponentide taaskasutatavuse. Tuleviku perspektiivis annab Reacti kasutuselevõtt hea võimaluse luua ka hübriid-mobiilirakenduse React Native raamistikku kasutades, kuna see on väga sarnane Reacti kontseptsioonide ja süntaksiga. Tänu selle hoiab ressurssi kokku uue tehnoloogia õppimise või uute arendajate palkamise pealt [27].

Kliendipoolse rakenduse arendamisel kasutatakse ka TypeScripti tuge, mis võimaldab kirjutada JavaScripti koodi toetades staatilist ehk tugevat tüüpimist. Hiljem TypeScripti kompileerimisel tehakse tüübi-kontroll, raporteeritakse olemasolul vead ning seejärel väljastatakse samaväärne JavaScripti kood [28].

4 Rakenduse arendus

Rendiplatvormi arendus jaguneb kolme etappi: andmebaasimudeli väljatöötamine, serveripoolse rakenduse arendamine ning kliendipoolse rakenduse arendamine. Antud peatükis antakse ülevaade eelnevalt välja toodud etappidest.

4.1 Andmebaasi olemi-suhte diagramm

Rakenduse arendamisega alustamisel on esimeseks sammuks ERD (*Entity Relationship Diagram*) skeemi loomine. See on sisuliselt loodavale rakendusele vundamendiks, mille põhjal rakendust üles ehitama hakatakse. Käesoleva andmebaasi olemi-suhte diagrammi loomisel on lähtutud peatükis 3.1 määratud funktsionaalsetest ja mittefunktsionaalsetest nõuetest.

ERD koosneb olemitest, mis jagunevad füüsilisteks (renditav toode) ja loogilisteks (renditava toote hind), ning olemite vahelistest seostest [29]. Seosed võivad olla kas üks-ühele, üks-mitmele või mitu-mitmele. Antud rakenduse puhul võib näiteks renditav toode olla seotud mitme erineva renditellimusega.

Rakenduses on igale olemi kirjele ette nähtud ka selle loomise, uuendamise ja kustutamise ajatempli ning tegevuse käivitanud isiku väljad. Olemi-suhte diagrammist on need lihtsustamise mõttes välja jäetud. Eelnevalt mainitud väljad sisaldavad olulist infot silumise seisukohast. Samuti kasutatakse kustutamise ajatemplit *Soft Delete* ehk pehme kustutamise funktsionaalsuse võimaldamiseks. See tähendab seda, et kirjet päriselt andmebaasist ära ei kustutata, vaid lihtsalt märgitakse kustutatuks. [30]

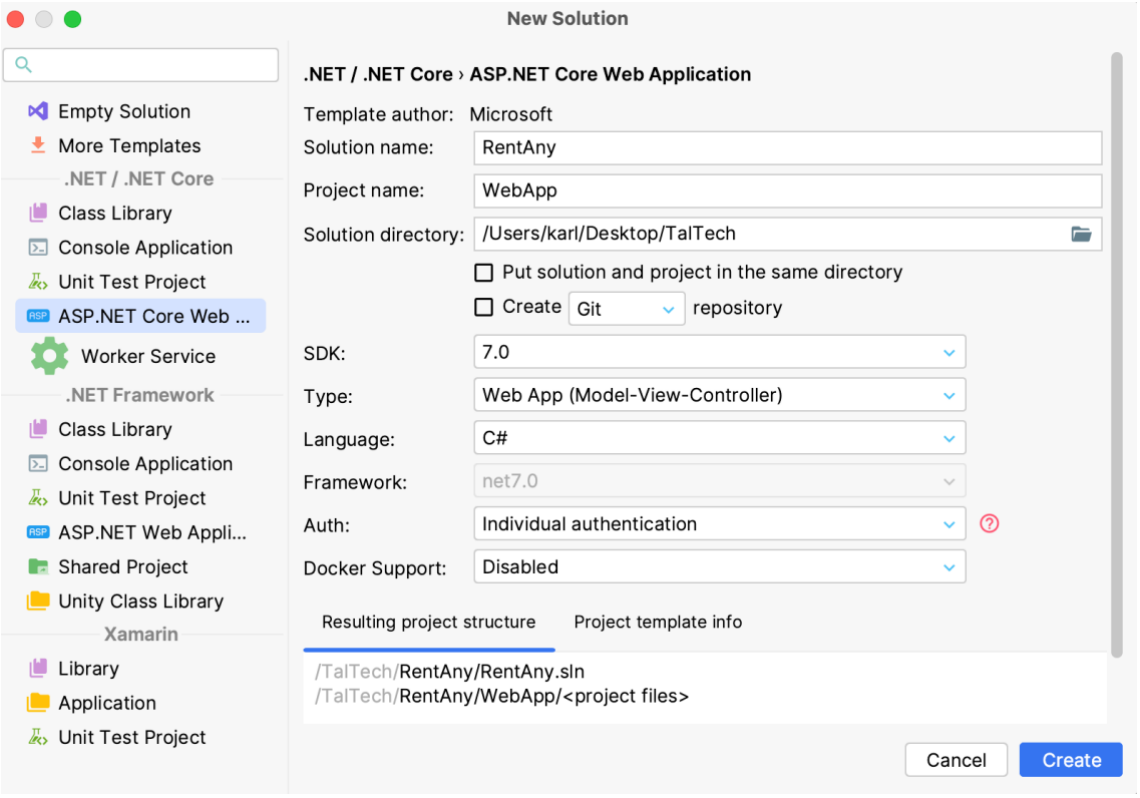
Käesoleva töö olemi-suhte diagramm on välja toodud lisas 2, mille koostamiseks kasutas autor Vertabelo veebirakendust. Joonisel on andmebaasi olemid jaotatud peatükis 3.3.1 määratletud moodulite kaupa sektsioonidesse. *Identity* ehk identiteedi sektsioonis *AspNet* prefiksiga tabelid tulenevad .NET Core raamistikku sisse ehitatud kasutajahalduse moodulist.

4.2 Serveripoolse rakenduse arendus

Serveripoolne rakendus on loodud kasutades C# programmeerimiskeelt ja .NET Core raamistikku. Integreeritud arenduskeskkonnana (*IDE*) kasutati JetBrains Rider arendustarkvara. Järgnevates peatükkides antakse ülevaade serveripoolse rakenduse arendusprotsessist.

4.2.1 Serveripoolse rakenduse loomine

Rakenduse loomiseks kasutati Rideri uue lahenduse lisamise tööriista, kus on võimalik valida sobiv mall ning teha soovikohaselt erinevaid seadistusi. Kuna tegemist on veebirakendusega, siis valiti projekti malliks „ASP.NET Core Web Application“ ning kasutati uusimat .NET raamistiku SDK (*Software Development Kit*) 7.0 versiooni. Autentimisviisiks valiti „Individual authentication“, mis võimaldab rakenduses kasutajaks registreeruda ning sisse logida. Serveripoolse rakenduse loomise modaali kuvatõmmis on välja toodud joonisel 5.



Joonis 4. ASP.NET Core veebirakenduse loomine Rideris.

Koodibaas on struktureeritud erinevatesse kaustadesse, kuhu kuuluvad rakenduse projektid. .NET raamistiku mõistes koondab lahendus (*solution*) kokku erinevad projektid (*projects*), mis on kompileeritavad käivitatavateks teekideks või rakendusteks [31]. Kaustade struktuuri kuvatõmmis on välja toodud joonisel 6 ning järgnevalt on kirjeldatud lähemalt projekte sisaldavate kaustade sisu:

- **App** – selles kaustas on käivitatava veebirakenduse WebApp projekt koos selles sisalduva konfiguratsiooniga, mis on rakenduse sisenemispunktiks ning töötleb API päringuid. Samuti on siin rakenduse domeeniklasside projekt ja andmebaasi ning rakenduse vahelist suhtlust korraldav projekt.

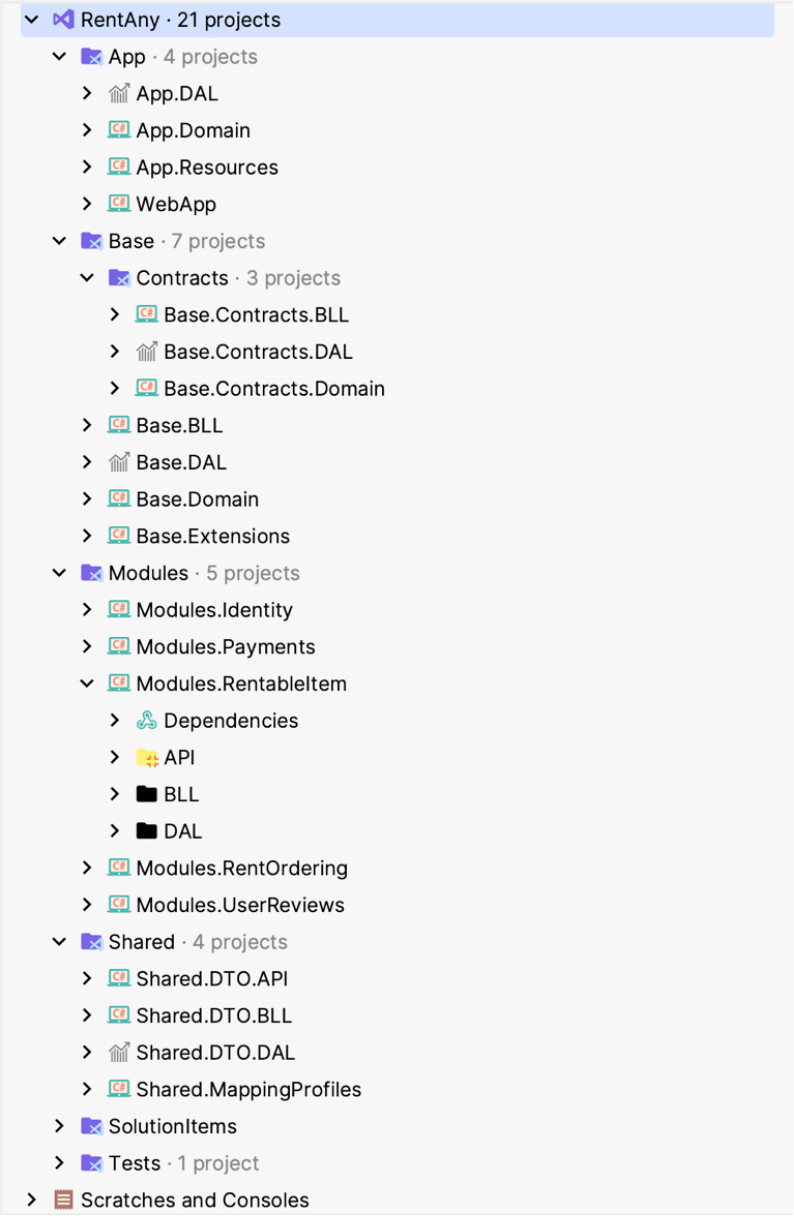
- **Base** – selles kausta projektid sisaldavad abstraktseid klasse ja liideseid, millest pärinevad või mida implementeerivad kõik rakenduse olemid, repositooriumid ja teenused. See aitab vähendada koodikordust ning paneb paika kindlad reeglid konkreetse klassi implementeerimiseks. Tänu abstraktsusele ei sõltu antud kausta sisu konkreetsest rakendusest ja on korduvkasutatav sisuliselt igas .NET raamistikku kasutatavas projektis.

- **Modules** – siia kausta on koondatud rakenduse erinevad moodulid. Lähtuvalt analüüsist sisaldab iga moodul endas DAL ehk andmevahetuskihti, BLL ehk teenusekihti ja API kihti.

- **Shared** – antud kaust sisaldab projekte, mis on jagatavad rakenduse eri moodulite või projektide poolt. Selleks võivad olla näiteks DTO (*Data Transfer Object*), vahevara (*Middleware*) või erinevate liideste projektid.

- **SolutionItems** – see kaust sisaldab muid mitte otseselt rakenduse endaga seotud faile. Näiteks on selles kaustas .gitignore, README ning Docker konteinerite loomise failid.

- **Tests** – selles kaustas on rakenduse testimiseks mõeldud projektid.



Joonis 5. Serveripoolse rakenduse kaustade struktuur.

4.2.2 Andmebaasi loomine

Andmebaasi loomisel kasutatakse *Code First* lähenemist ehk esmalt luuakse domeeniklassid koos väljadega ning seejärel andmebaas genereeritakse koodi põhjal [32]. Peatükis 4.1 on tehtud andmebaasi olemi-suhte diagramm, mille põhjal loodi esmalt üks ühele vastavuses rakenduse domeeniklassid kasutades C# programmeerimiskeelt. Joonisel 7 on välja toodud näide ühest domeeni objektist.

```
public class RentItemCategory : DomainEntityId
{
    public Guid? ParentCategoryId { get; set; }

    public RentItemCategory? ParentCategory { get; set; }

    [MaxLength(128)] public string Title { get; set; } = default!;

    [MaxLength(4096)] public string Description { get; set; } = default!;

    public string ImageUrl { get; set; } = default!;
}
```

Joonis 6. Renditoote kategooria domeeni klass.

Andmebaasi migreerimiseks kasutatakse ORM (*Object Relational Mapping*) raamistikku Entity Framework. Sisuliselt loob ORM rakendusesisese virtuaalse andmebaasi ning kõik muudatused peegeldatakse salvestamisel traditsioonilisse andmebaasi [33]. See võimaldab abstrakteerida seoseid relatsiooniliste andmebaasidega nii, et arendaja saab käsitleda andmebaasi olemeid domeeni objektidena. Sellise lähenemisega seotakse rakendus andmete juurdepääsu loogikast lahti ning tänu sellele on näiteks ka võimalik lihtsasti vahetada andmebaasimootorit [34].

Andmebaasi loomiseks tuleb genereerida enne migratsioonifail. Selle jaoks vastav käsklus „Migratsioonifaili loomine“ on välja toodud joonisel 8. Migratsioonifail vaadatakse arendaja pilguga üle, et kõik vastaks ootustele. Domeeni objektide muutmisel tuleb antud käsklust uuesti kasutada, mis loob uue migratsioonfaili võrreldes eelmist ja tehes sellisel viisil kindlaks erinevused.

Kui migratsioonifail tundub korrektne olevat, siis tuleb muudatused andmebaasis ka sisse viia. Selle jaoks vastav käsklus „Andmebaasi uuendamine“ on välja toodud joonisel 8. Antud käsklust tuleb kasutada peale igat uue migratsioonifaili genereerimist, mis loob või uuendab ära andmebaasi vastavalt migratsioonifaili sisule. Selle tulemusena on andmebaas valmis kasutamiseks.

Migratsioonifaili loomine

```
dotnet ef migrations add --project App.DAL --startup-project WebApp --context AppDbContext Initial
```

Andmebaasi uuendamine

```
dotnet ef database update --project App.DAL --startup-project WebApp
```

Migratsioonifaili eemaldamine

```
dotnet ef migrations remove --project App.DAL --startup-project WebApp --context AppDbContext
```

Andmebaasi kustutamine

```
dotnet ef database drop --project App.DAL --startup-project WebApp
```

Joonis 7. Andmebaasi migratsiooni käsklused.

4.2.3 Moodulid

Rakenduse moodulite siseselt on arendamisel lähtutud kihilise arhitektuuri põhimõtetest. Moodul jaguneb andmevahetus, teenuse ja API kihtideks. Järgnevalt on lähemalt kirjeldatud nende kihtide ülesehitust.

Andmevahetuskihis on kasutatud repositooriumi (*Repository*) ja UoW (*Unit Of Work*) mustreid. DAL sisaldab endas iga sellega seotud olemi kohta eraldi repositooriumit. Repositooriumid eraldavad äriloogikast ja API kihist vajalikud CRUD (*Create Read Update Delete*) operatsioonid andmebaasist andmete küsimiseks [35]. Antud muster toetab DRY printsiipi ning muudab rakenduse kergemini testitavaks ja laiendatavaks [36]. UoW seob kõik mooduli repositooriumid ühte objekti kokku ning võimaldab kõikide repositooriumite operatsioonid saata andmebaasi ühe päringuna mitme päringu asemel. Selline lähenemine tagab andmete järjepidevuse, kuna operatsioonid kas õnnestuvad või ebaõnnestuvad ühe päringuna [35].

Teenusekiht koosneb äriloogikat sisaldavatest teenustest, mis on loodud sarnaselt repositooriumitele iga mooduli olemi kohta eraldi. Teenused kasutavad sellele vastava olemi repositooriumit andmebaasist andmete küsimiseks, mille ümber on teenuses loodud ärilised protsessid. Selline lähenemine eraldab äriloogika nii andmevahetus kui ka API kihist ning vähendab äriloogilist koodikordust.

API kiht sisaldab endas iga olemi kohta eraldi kontrollerit, mis võtab vastu HTTP päringuid, töötleb neid kasutades teenusekihis paika pandud äriloogikat ning saadab päringu tegijale vastuse koos andmetega.

Kõikides kihtides on iga mooduli olemi kohta oma DTO. Objekti liikumisel ühest kihist teise vastendatakse (*mapping*) see automaatselt vastava kihi DTO-ks kasutades .NET raamistiku teeki AutoMapper. See võimaldab vastavalt vajadusele paika panna, mis välju igasse kihti edasi antakse. Näiteks ei taha me üldjuhul päringu tegijale saata kõikide andmebaasi tabelis sisalduvate väljade andmeid. Samas võib mõningaid välju olla tarvis teada näiteks äriloogika kihis. DTO-de abil saamegi teenuse ja API kihi vahel kliendipoolsele rakendusele mitteavaldatavad väljad välja vastendada.

4.2.4 REST API

REST API, mis on tuntud ka kui RESTful API, on rakenduste programmeerimisliides, mis vastab REST (*Representational State Transfer*) arhitektuurilise stiili piirangutele ning võimaldab suhelda RESTful veebiteenustega [37]. Antud liidese puhul suhtlevad kliendi- ja serveripoolne rakendus üle HTTP protokolli. HTTP sisaldab endas meetodeid, millega serveripoolse rakenduse poole pöördumisel määratletakse ära soovitud tegevuse tüüp. Põhilisteks HTTP meetoditeks on GET (andmete pärimine), POST (andmete lisamine), PUT (andmete muutmine) ja DELETE (andmete kustutamine).

Igale API kontrollerile on määratletud üks kindel URL, mille poole kliendipoolsel rakendusel on võimalik pöörduda kasutades HTTP protokolli ja eelnevalt kirjeldatud meetodeid. API kontrollerite loomiseks on kasutatud .NET raamistiku koodi generaatorit, mis genereerib rakenduse olemi põhjal automaatselt kontrolleri klassi. Seejärel on seda võimalik vastavalt vajadusele muuta või täiustada. Igale API kontrollerile on atribuutidega määratletud ka selle versioon ja marsruut. API versioneerimine on oluline, et uute funktsionaalsuste lisamisel olemasolevad kliendipoolsed rakendused selletõttu katki ei läheks [38].

API kontroller sisaldab endas hulga erinevaid meetodeid, mis töötlevad sissetulevaid päringuid vastavalt marsruudile ja HTTP meetodi tüübile. Kontrolleri meetodeid on kirjeldatud erinevate atribuutidega. Näiteks tuleb määratleda, millisele HTTP meetodile konkreetne kontrolleri meetod vastab ning mis tüüpi formaadis rakendus vastuse saadab. Käesoleva rakenduse puhul on vastuse formaadiks enamjaolt *application/json*. Joonisel 9 on välja toodud näide kontrolleri meetodist.

```
// GET: api/RentItem
/// <summary>
/// Returns all RentItem objects.
/// </summary>
/// <returns>RentItem[]</returns>
[HttpGet]
[Produces( contentType: "application/json")]
[ProducesResponseType(typeof(RentItem[]), statusCode: StatusCodes.Status200OK)]
[ProducesResponseType( statusCode: StatusCodes.Status400BadRequest)]
1 usage  kkoiva *
public async Task<ActionResult<IEnumerable<RentItem>>> GetRentItem()
{
    return (await _bll.RentItem.GetAllAsync()).Select(x :RentItem => _mapper.Map(x)!).ToList();
}
```

Joonis 8. API kontrolleri meetod.

Selleks, et nii inimesed kui ka arvutid saaksid paremini mõista rakenduse API võimalusi ilma lähtekoodi vaatamata, on oluline API dokumenteerida [39]. Serveripoolses rakenduses on kasutatud Swagger UI tööriista, mis loob automaatselt API kontrollerite ja nendes kasutatud atribuutide põhjal dokumentatsiooni. Selle abil on kasutajaliideses võimalik näha detailset informatsiooni API kontrolleris olevate meetodite kohta.

4.2.5 Autentimine ja autoriseerimine

Rakenduses on kasutatud individuaalset autentimist, mille kohaselt kasutatakse isiku tuvastamiseks kasutajanime ja parooli. Selleks on serveripoolses rakenduses kasutusel .NET Core raamistiku teek Identity, mis kasutab OAuth2 standardi kohast autentimise voogu. Järgnevalt on kirjeldatud OAuth2 autentimise voo samme [40].

1. Lõppkasutaja sisestab kliendipoolses rakenduses kasutajanime ja parooli.
2. Kliendipoolne rakendus saadab serveripoolsele rakendusele sisestatud kasutajanime ja parooli kasutades REST API-d.
3. Serveripoolne rakendus verifitseerib saadud kasutajanime ja parooli ning õnnestumise korral saadab kliendipoolsele rakendusele tagasi JWT (*JSON Web Token*) tokeni.
4. Kliendipoolne rakendus pääseb ligi kaitstud ressurssidele lisades igasse HTTP päringu *Authorization* päisesse eelnevas sammus saadud JWT tokeni.

API kontrollerites on võimalik atribuutidega piirata ressurssidele ligipääsemist sisseloginud kasutaja ja erinevate rollide tasemel. Selleks, et piirata sisseloginud kasutajale ligipääs näiteks ainult selle kasutajaga seotud renditellimustele, tuleb teha vastavad kontrollid äriloogika tasemel. Joonisel 10 on välja toodud näide kontrolleri juurdepääsupiirangu atribuutidest.

```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
[Authorize(Roles = App.Domain.Identity.AppRole.AdminRole)]
```

Joonis 9. API kontrolleri juurdepääsupiirangu atribuudid.

4.3 Kliendipoolse rakenduse arendus

Kliendipoolne rakendus on loodud kasutades JavaScript programmeerimiskeelt ja React.js raamistikku. Integreeritud arenduskeskkonnana (*IDE*) kasutati JetBrains WebStorm arendustarkvara. Järgnevates peatükkides antakse ülevaade kliendipoolse rakenduse arendusprotsessist.

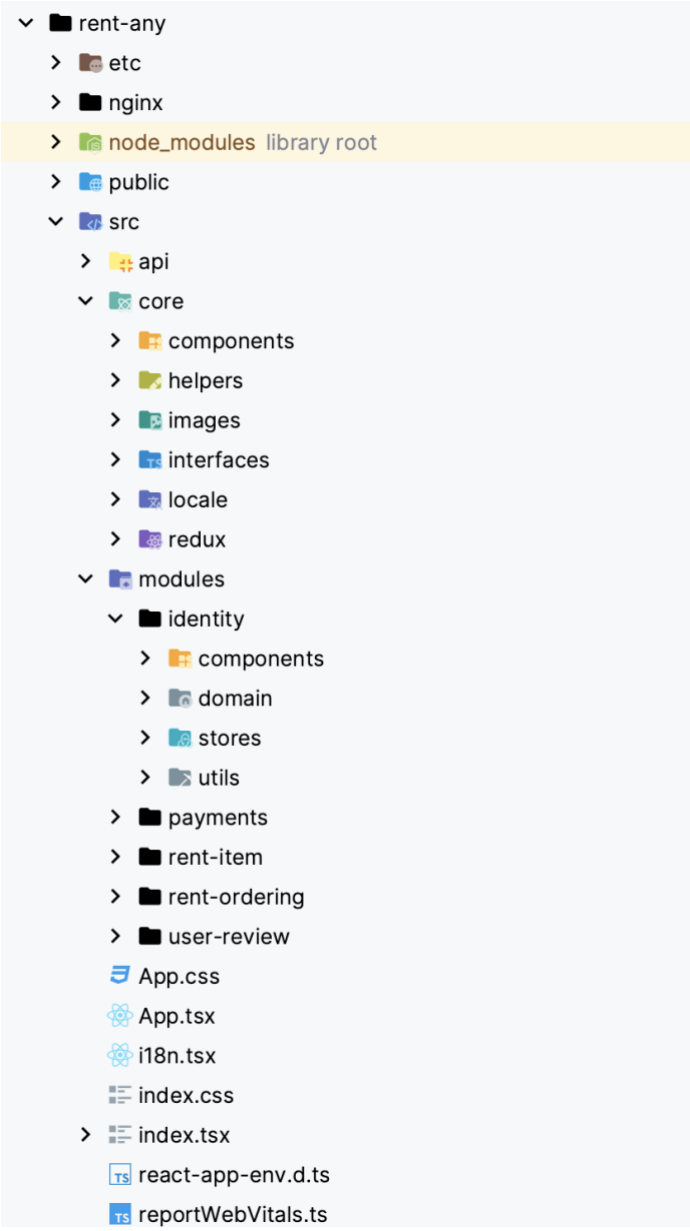
4.3.1 Kliendipoolse rakenduse loomine

Uue React.js raamistiku rakenduse loomiseks kasutas töö autor Facebooki poolt loodud *create-react-app* tööriista, mis loob arendajale automaatselt React rakenduse struktuuri. Selle tulemusena saab arendaja koheselt keskenduda koodile, mitte rakenduse ja erinevate tööriistade seadistamisele [41]. Joonisel 11 on välja toodud käsklus uue React.js rakenduse loomiseks. Analüüsist tulenevalt kasutatakse kliendipoolses rakenduses TypeScripti tüübikontrolli ning seega tuli käsklusesse lisada ka sellele vastav mall.

```
npx create-react-app rent-any --template typescript
```

Joonis 10. Käsklus uue React rakenduse loomiseks.

Koodibaas on struktureeritud põhiliselt moodulite alusel erinevatesse kaustadesse. Iga moodul sisaldab endas komponentide (*components*), domeeni (*domain*), andmehoidlate (*stores*) ja mooduliga seotud erinevate abifunktsioonide (*utils*) kaustasi. Lisaks on *core* kaust, kus hoitakse rakenduse üldkasutatavaid liideseid, komponente ja abifunktsioone. Joonisel 12 on välja toodud kuvatõmmis kliendipoolse rakenduse kaustade struktuurist.



Joonis 11. Kliendipoolse rakenduse kaustade struktuur.

Üheleherakenduse marsruutimiseks erinevate vaadete vahel ilma veebilehte uuesti laadimata on kasutatud React Router teeki. Tuleviku vaatest sujuvamaks keelevahetuse funktsionaalsuse lisamiseks on juba koheselt kasutusele võetud *react-i18next* tõlketeek. Antud teegi abil saab komponentides kasutada tõlkevõtit, mille järgi otsitakse vastava keele JSON failist sellele väärtus. Tänu sellele pole uue keele lisamisel vaja komponentide siseselt muudatusi teha ning piisab vaid uue tõlkefaili lisamisest ja vajalike konfiguratsioonide tegemisest.

Ühe olulise teegina on rakenduses kasutusel veel Redux. See võimaldab luua ühte globaalset olekut sisaldava andmehoidla, millele on võimalik ligi pääseda kõikidel komponentidel tagades samaaegselt andmete järjepidevuse rakenduse üleselt [42]. Antud rakenduses hoitakse Reduxi hoidlas näiteks sisse loginud kasutaja objekti ning otsingulehe filtrite valikuid. Käesoleva töö prototüübi kasutajaliidese loomise lihtsustamiseks on kasutatud Tailwind CSS raamistikku ja Headless UI komponente.

4.3.2 Suhtlus serveripoolse rakendusega

Kliendipoolses rakenduses HTTP päringute koostamise lihtsustamiseks on kasutatud JavaScripti maailmas populaarset Axios klienti. Kasutades Axios instantsi tehakse kõik päringud läbi konkreetsele olemile vastava teenuse, et eraldada komponentidest andmete küsimise loogika ja vähendada koodikordusi. Päringute tegemiseks tuleb Axios instantsile kaasa anda API lõpp-punkt (*endpoint*) ning olemasolu korral ka parameetrid.

Axios võimaldab luua instantsile ka pealtkuulajaid (*interceptors*), mille abil on võimalik igat saadetavat päringut või saabuvat vastust modifitseerida. Antud rakenduses lisatakse iga sisseloginud kasutaja saadetava päringu puhul *Authorization* päis koos JWT tokeni väärtusega. Juhul kui kasutajal puudub ligipääs soovitud ressursile või token ei ole valiidne, siis tagastab serveripoolne rakendus veateate. Joonisel 13 on välja toodud näide päringu saatmisest kasutades Axios instantsi.

```
postRentOrder(rentOrder: IRentOrderPost) : Promise<AxiosResponse<IRentOrd...  {
  return api.post<IRentOrder>(RENTORDER_URL, rentOrder);
}
```

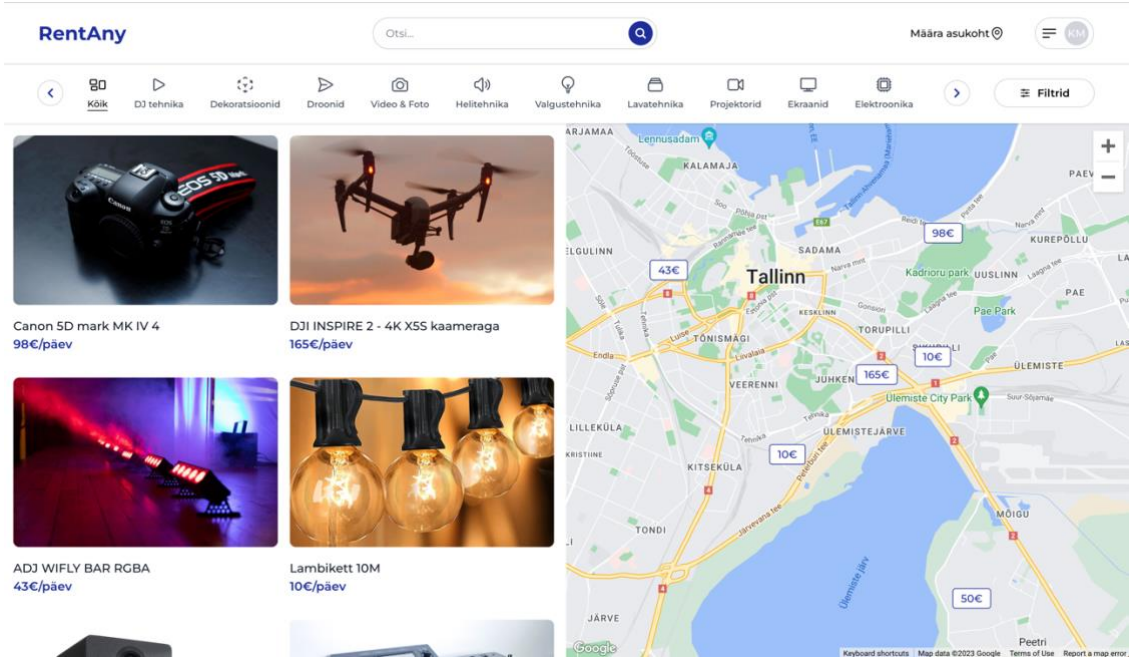
Joonis 12. Serveripoolsele rakendusele päringu saatmine.

4.3.3 Otsingulehe arendus

Olemasolevatest lahendustest eristumisel on otsinguleht loodava lahenduse puhul üks olulisemaid osasi, mille abil peab klientidel olema võimalik jõuda kiirelt ja mugavalt rentimiseks sobiva tooteni. Analüüsi peatükis 3.1 määratud funktsionaalsetele nõuetele vastavalt peab kasutajal olema võimalik renditoodete otsimiseks kasutada kaardivaadet ning teisi filteerimise ja otsimise kriteeriumeid. Sellest lähtuvalt on rakenduse esilehe vaade jagatud kaheks, kus vasakul pool on nimekiri renditoodetest ning paremal pool interaktiivne kaardivaade koos märgistatud renditoodete asukohtadega (Joonis 14).

Otsingulehe loomiseks on kasutusele võetud Algolia tehisintellektil põhinev otsinguplatvorm, mis võimaldab luua võimekaid, paindlikke ja skaleeritavaid digitaalseid ärilahendusi [43]. Uue renditoote lisamisel saadetakse serveripoolses rakenduses Algolia platvormil seadistatud indeksisse uus kirje. Kasutades Algolia poolt pakutavat React raamistiku teeki, tehakse kliendipoolses rakenduses päringuid Algolia otsingumootori pihta, mis tagab kiired ja relevantsed tulemused. Tulemusi on võimalik filtreerida otsingufraasi, kategooriate, saadaolevate kuupäevade ning koordinaatide alusel. Filtreerimine ja sorteerimine tehakse Algolia otsingumootori poolt vastavalt kasutaja sisendiga päringule kaasa antud kriteeriumitele.

Kaardivaade on loodud kasutades Google Maps JavaScript API-d, mille peale on arendatud kaardivaate filtreerimine kasutades selleks Algolia tööriistu. Kaardi suumimisel või liigutamisel uueneb automaatselt vasakul pool asuv renditoodete nimekiri vastavalt sellele, kas renditoode jääb kaardivaate raamide sisse või mitte. Kui nimekirjas minna hiirega mõne konkreetse renditoote kohale, muutub kaardivaates vastava toote märgistuse värv. Kaardivaates on märgistused klikitavad, mille tulemusena avaneb lisaks hinnale ka detailsem info.



Joonis 13. Rakenduse otsinguleht.

5 Tulemused

Käesolevas peatükis antakse ülevaade antud lõputöö tulemustest. Esmalt kirjeldatakse töö käigus loodud funktsionaalsusi. Seejärel kasutatakse töö tulemuste objektiivsemaks hindamiseks Saaty meetodit, mille raames viiakse läbi paaritised võrdlused ning tehakse järeldused.

5.1 Loodud funktsionaalsused

Lõputöö tulemusena said seatud eesmärgid täidetud ehk valminud analüüs toetas sellise kõik ühes rendiplatvormi arendamist ning praktilise osana valminud rendiplatvormi prototüüp täidab kõiki analüüsis määratud nõudeid. Rakendus võimaldab kasutajatel platvormil registreeruda ning süsteemi sisse logida, mille tulemusena on võimalik teostada samaaegselt nii rentija kui ka rendileandja rollis olevaid tegevusi.

Rentijal on võimalik kasutada võimekat otsingulehte endale sobiva renditoote leidmiseks kasutades erinevaid filtreid ning interaktiivset kaardivaadet. Otsingulehe detailsem ülevaade on kirjeldatud 4.3.3 peatükis. Huvipakkuvale renditootele vajutades avaneb detailvaade (Lisa 3). Detailvaatel on võimalik valida soovitud rendikuupäevad, mille tulemusena tehakse päring serveripoolsesse rakendusse ning tagastatakse hinnakiri. Renditellimuse kinnitamiseks tuleb täita nõutud arveldusinfo väljad ning valida sobiv makseviis.

Rendileandjana on võimalik lisada platvormile uus renditoode täites ära selleks vastava vormi, kus tuleb lisada kuni viis pilti ning täita kõik muud funktsionaalsetest nõuetest tulenevad väljad. Rakendus võimaldab näha ka sissetulevate broneeringute nimekirja, kus konkreetsele broneeringule vajutades avaneb renditellimuse detailvaade võimaldades rendileandjana uuendada tellimuse staatust (Lisa 4).

Rendiplatvormi prototüüp vastab kõigile seatud mittefunktsionaalsetele nõuetele välja arvatud WCAG juurdepääsetavuse juhiste jälgimine, mille puhul jõuti antud lõputöö raames jälgida ainult värvide kontrastsuse ning piltide tekstikujulise alternatiivi nõuet.

5.2 Saaty meetodi kasutamine lõpptulemuse hindamiseks

Järgnevalt viiakse lihtsustatud kujul läbi lõpplahenduse hindamine kasutades analüütiliste hierarhiate ehk Saaty meetodit, mis võimaldab saada subjektiivse arvamuse asemel objektiivsema hinnangu. Meetodi esmasteks etappideks on eesmärgi määramine ning kriteeriumite ja alternatiivide valimine. Seejärel viiakse läbi kriteeriumite omavaheline paaritine võrdlus kriteeriumite osatähtsuste leidmiseks. Viimaseks tehakse alternatiivide paaritine võrdlus iga kriteeriumi suhtes eraldi, mille põhjal leitaksegi lõpptulemusena iga alternatiivi suhteline süsteemi headus.

Kuna eesmärk on võrrelda erinevaid rendiplatvorme, siis on üheks alternatiiviks töö käigus valminud uus rendiplatvorm ning ülejäänud alternatiivideks töös eelnevalt analüüsitud kaks sarnast Eestis olemasolevat lahendust.

Paaritiste võrdluste läbiviimiseks kasutatav hinnangute skaala on järgnev:

1 - Võrdtähtis
2 - Mõõdukas paremus
3 - Oluline paremus
4 - Tugev paremus
5 - Väga tugev paremus

5.3 Kriteeriumite paaritine võrdlus

Kriteeriumiteks on valitud rakenduse peamised omadused, mis tulenevad analüüsis määratud funktsionaalsetest ja mittefunktsionaalsetest nõuetest. Arvuliste hinnangute andmisel lähtus autor konkreetse kriteeriumi headuse olulisusest kasutaja perspektiivist. Tabelis 5 on näha kriteeriumite omavaheline paaritine võrdlus kriteeriumite osatähtsuste leidmiseks, millest sõltub olulisel määral hindamise lõpptulemus. Selgub, et kõige suurema kaaluga kriteeriumiks on otsinguleht (43.3%) ning sellele järgneb UI/UX disaini kriteerium (28.7%).

Veendumaks tulemuste stabiilsuses on välja arvutatud ka CR (*Consistency Ratio*) ehk kokkulangevuse suhtarv. Vastuolu võib lugeda lubatavaks mõõtmisveaks ainult siis, kui CR näitaja väärtus on 10%-st väiksem [44]. Antud maatriksi puhul arvutati CR väärtuseks 1.9% ning seega võime järeldada, et vaadeldav hinnangute süsteem on stabiilne.

Tabel 5. Kriteeriumite paaritine võrdlus.

Kriteerium	UI/UX disain	Otsinguleht	Renditoote lisamine	Broneerimine	**Osatähtsus**
UI/UX disain	1	1/2	3	2	**0.287**
Otsinguleht	2	1	4	2	**0.433**
Renditoote lisamine	1/3	1/4	1	1/2	**0.097**
Broneerimine	1/2	1/2	2	1	**0.183**
Kokku	**3.83**	**2.25**	**10**	**5.5**	

5.4 Alternatiivide paaritine võrdlus

Alternatiivideks on valitud kaks olemasolevat lahendust Rentif ja Forent ning kolmandaks käesoleva lõputöö raames loodud uus lahendus nimega RentAny. Arvuliste hinnangute andmisel lähtuti autori saadud kogemustest antud rendiplatvormide kasutamisel. Kõikide võrdluste koostamisel on jälgitud ka CR väärtust stabiilsete hinnangute tagamiseks. Alternatiivide võrdlus suhtelise headuse hindamiseks on läbi viidud iga kriteeriumi suhtes eraldi, kuid korduste vähendamiseks on tabelis 6 välja toodud paaritise võrdluse näide ainult otsingulehe kriteeriumi osas.

Otsingulehe võrdlemise puhul võeti arvesse otsimise ja filtreerimise võimalusi ning üldist kasutajamugavust. Näiteks Forenti puhul puudus võimalus samaaegselt otsida märksõna järgi ja filtreerida kategooria alusel. Samuti olid need ainsateks otsinguvõimalusteks ning seega on antud alternatiivi puhul madalaim kaal põhjendatud (10.6%). Järgneb rohkemate ja otstarbekamate otsinguvõimalustega Rentif (26%). Parima tulemuse saavutas RentAny (63.4%) koos interaktiivse kaardivaate ning mitmete kasulike otsimise ja filtreerimise võimalustega, mis toetavad olulisel määral kasutajatel soovitud renditoote leidmist.

Tabel 6. Otsingulehe kriteeriumi paaritine võrdlus.

Alternatiiv	Rentif	Forent	RentAny	**Kaal**
Rentif	1	3	1/3	**0.26**
Forent	1/3	1	1/5	**0.106**
RentAny	3	5	1	**0.634**
Kokku	**4.33**	**9**	**1.53**	

Tabelis 7 on välja toodud kokkuvõte alternatiivide kaaludest iga kriteeriumi suhtes. Selgub, et kolme kriteeriumi puhul neljast saavutas käesoleva lõputöö raames loodud uus lahendus RentAny parima tulemuse. Rentif saavutas parima tulemuse renditoote lisamise kriteeriumi osas, mille põhjuseks oli suuresti kategooria põhiste lisainformatsiooni väljade täitmise võimalus. Näiteks auto kategooria renditoote lisamisel oli võimalik rendileandjal täita ära ka auto marki, mudelit, tootmisaastat ja muud sarnast kirjeldavad väljad.

Tabel 7. Kokkuvõte alternatiivide kaaludest iga kriteeriumi suhtes.

Kriteerium	Rentif	Forent	RentAny
UI/UX disain	0.239	0.137	**0.624**
Otsinguleht	0.26	0.106	**0.634**
Renditoote lisamine	**0.571**	0.143	0.286
Broneerimine	0.354	0.09	**0.556**

5.5 Tulemuste analüüs

Saadud alternatiivide kaalud iga kriteeriumi kohta on korrutatud läbi kriteeriumi osatähtsusega ning seejärel saadud väärtuste summeerimisel leitaksegi suhteline süsteemi headus. Saaty meetodi rakendamise lõpptulemus on välja toodud tabelis 8.

Lõpptulemusena selgub, et toodete rentimiseks ja rendile andmiseks sobib olemasolevate lahenduste seast kõige paremini käesoleva lõputöö raames loodud rendiplatvorm

RentAny. Edu saavutati peamiselt otsingulehe kriteeriumi alusel, kuid oluline paremus avaldub ka UI/UX disaini kriteeriumi osas. Teisele kohale jäänud Rentif paistis silma põhjaliku renditoodete lisamise vormiga, kuid antud kriteeriumi madala osatähtsuse tõttu see lõpptulemustes suuremat rolli ei mänginud. Kolmandaks jäänud Forent rakendusel puudus läbi platvormi renditoote broneerimise võimalus ning toote rentimiseks tuli võtta ühendust toote omanikuga kontaktandmete abil. Samuti oli miinuseks kasutajakogemuse- ja liidese disain ning renditoodete piiratud otsimisvõimalused.

Tabel 8. Saaty meetodi lõpptulemus.

Kriteerium	Rentif	Forent	RentAny
UI/UX disain	0.068	0.039	0.179
Otsinguleht	0.113	0.046	0.274
Renditoote lisamine	0.055	0.014	0.028
Broneerimine	0.065	0.017	0.102
Süsteemi headus	**0.301**	**0.116**	**0.583**

6 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli esiteks analüüsida võimalusi kõik ühes rendiplatvormi loomiseks, mis võimaldaks erinevaid tooteid rentida ning rendile anda. Teiseks arendada analüüsile tuginedes valmis rendiplatvormi prototüüp. Rakenduse loomise ajendiks oli käesoleva ajastu lineaarne majandusmudel, mis ei ole enam paraku jätkusuutlik ning viib potentsiaalselt ühel hetkel loodusvarade kriisini. Seetõttu on oluline välja töötada uusi tehnoloogiaid, lahendusi ja võimalusi, mis toetaksid uuele keskkonnasäästlikumale mudelile üleminekut ning vähendaks inimkonna sõltuvust toorainest.

Lahenduse väljatöötamisel esmalt analüüsiti probleemi olemust ning olemasolevaid lahendusi koos nende eeliste ja puudustega, mille alusel määrati loodava rakenduse mittefunktsionaalsed ja funktsionaalsed nõuded. Eelneva põhjal koostati tehniline analüüs rendiplatvormi loomiseks, mis hõlmab endas tehnoloogiate valikut, arhitektuurilisi otsuseid ning kasutakogemuse- ja liidese disaini. Töö praktilises osas arendati analüüsile ja nõuetele tuginedes rendiplatvormi prototüüp kirjeldades seejuures nii serveri- kui ka kliendipoolse rakenduse arendusprotsessi.

Lõputöö eesmärgid autori hinnangul saavutati ning töö tulemusena valmis rendiplatvormi prototüüp, mis võimaldab kõikidel erinevaid tooteid rentida ja rendile anda. Rakendus hõlmab endas funktsionaalset otsingulehte koos erinevate filtreerimisvõimaluste ja interaktiivse kaardivaatega, renditoodete lisamist ja broneerimist ning muid renditellimuste haldamisega seotud funktsionaalsusi. Lisaarenduste korral on rakendust võimalik ka avalikult lansseerida. Lõpplahenduse objektiivsemaks võrdlemiseks olemasolevate lahendustega kasutati Saaty meetodit, mille tulemusena selgus, et toodete rentimiseks sobib olemasolevate lahenduste seast kõige paremini käesoleva lõputöö raames loodud rendiplatvorm.

Kasutatud kirjandus

[1] Keskkonnaministeerium, “Ringmajandus,” [Võrgumaterjal]. Kättesaadav: https://ringmajandus.envir.ee/et/ringmajandus. [Kasutatud 27.04.2023].

[2] F. Preston, “ A Global Redesign? Shaping the Circular Economy,” Chatham House, 2012 [Võrgumaterjal]. Kättesaadav: https://www.chathamhouse.org/2012/03/global-redesign-shaping-circular-economy [Kasutatud 20.03.2023].

[3] Euroopa Keskkonnaagentuur, “Majandus: ressursitõhusus, rohemajandus ja ringmajandus,” 11. september 2014 [Võrgumaterjal]. Kättesaadav: https://www.eea.europa.eu/et/eka-signaalid/signaalid-2014/artiklid/majandus-ressursitohusus-rohemajandus-ja-ringmajandus [Kasutatud 12.02.2023]

[4] Ragn-Sells AS, “Ringmajandus,” 2015 [Võrgumaterjal]. Kättesaadav: https://www.ragnsells.ee/keskkond/ringmajandus/ [Kasutatud 20.03.2023]

[5] UNEP, “Decoupling Natural Resource Use and Environmental Impacts from Economic Growth,” 2011 [Võrgumaterjal]. Kättesaadav: https://www.unep.org/resources/report/decoupling-natural-resource-use-and-environmental-impacts-economic-growth-summary [Kasutatud 26.03.2023]

[6] S. Talve, “Tulevikus ressursitarbimine muutub – loodetavasti saab rentida ka rösterit,” Tallinna Tehnikakõrgkool, 2018 [Võrgumaterjal]. Kättesaadav: https://www.tktk.ee/uudised/tulevikus-ressursitarbimine-muutub-loodetavasti-saab-rentida-ka-rosterit/ [Kasutatud 12.02.2023].

[7] K. E. Wiegers and J. Beatty, Software Requirements. Microsoft Press, 2013.

[8] M. Rehkopf, “Stories, epics, and initiatives,” Atlassian [Võrgumaterjal]. Kättesaadav: https://www.atlassian.com/agile/project-management/epics-storiesthemes. [Kasutatud 02.04.2023].

[9] J. Marsh, UX for Beginners. Sebastopol: O'Reilly Media, Incorporated, 2016.

[10] “7 fundamental UX design principles all designers should know,” UX Design Institute, 2022 [Võrgumaterjal]. Kättesaadav: https://www.uxdesigninstitute.com/blog/ux-design-principles/. [Kasutatud 16.04.2023].

[11] “Web Content Accessibility Guidelines (WCAG) 2.1,” W3C, 2018 [Võrgumaterjal]. Kättesaadav: https://www.w3.org/TR/WCAG21/. [Kasutatud 16.04.2023].

[12] M. Rouse, “Software Architecture,” 2018 [Võrgumaterjal]. Kättesaadav: https://www.techopedia.com/definition/24596/software-architecture. [Kasutatud 05.04.2023].

[13] Francesco Abbruzzese and Gabriel Baptista, Software Architecture with C# 10 and .NET 6 - Third Edition. Packt Publishing, 2022.

[14] K. Zettler, “What is a distributed system?,” Atlassian [Võrgumaterjal]. Kättesaadav: https://www.atlassian.com/microservices/microservices-architecture/distributed-architecture. [Kasutatud 05.04.2023].

[15] V. Podkamennyi, “Principles of Software Engineering,” 2020 [Võrgumaterjal]. Kättesaadav: https://vpodk.medium.com/principles-of-software-engineering-6b702faf74a6. [Kasutatud 05.04.2023].

[16] A. Hunt, D. Thomas, and Safari Tech Books Online, Hunt/The Pragmatic Programmer, First Edition. Addison-Wesley Professional, 1999.

[17] M. Martin and R. C. Martin, Agile, Principles, Patterns, and Practices in C#. Pearson Prentice Hall, 2021.

[18] Y. Tanikin, “Exploring the Benefits and Limitations of a Modular Monolith Architecture,” Medium, 2023 [Võrgumaterjal]. Kättesaadav: https://medium.com/javarevisited/exploring-the-benefits-and-challenges-of-a-modular-monolith-architecture-5ee9f69988c8. [Kasutatud 14.04.2023].

[19] M. Murugan, “Modular Architecture in ASP.NET Core – Building Better Monoliths,” Codewithmukesh, 2021 [Võrgumaterjal]. Kättesaadav: https://codewithmukesh.com/blog/modular-architecture-in-aspnet-core/?utm_content=cmp-true. [Kasutatud 14.04.2023].

[20] J. Martí, “Componentization principles to build reusable interfaces,” JM., 2021 [Võrgumaterjal]. Kättesaadav: https://joaquinmarti.com/posts/componentization-principles-to-build-reusable-interfaces/. [Kasutatud 14.04.2023].

[21] Y. Brikman, “Choosing a Tech Stack,” in Hello, Startup, United States: O'Reilly Media, Incorporated, 2015.

[22] „Usage statistics of PHP for websites,“ W3Techs [Võrgumaterjal]. Available: https://w3techs.com/technologies/details/pl-php. [Kasutatud 08.04.2023].

[23] S. Mino, “8 Reasons Why PHP Is Still So Important For Web Development,” Jobsity, 2022 [Võrgumaterjal]. Kättesaadav: https://www.jobsity.com/blog/8-reasons-why-php-is-still-so-important-for-web-development. [Kasutatud 09.04.2023].

[24] B. Limón, “Compiled vs interpreted language: Basics for beginning devs,” Educative, 2022 [Võrgumaterjal]. Kättesaadav: https://www.educative.io/blog/compiled-vs-interpreted-language. [Kasutatud 09.04.2023].

[25] Jason Hales, Almantas Karpavicius, and Mateus Viegas, The C# Workshop. Packt Publishing, 2022.

[26] Stackoverflow, “2022 Developer Survey,” 2022 [Võrgumaterjal]. Kättesaadav: https://survey.stackoverflow.co/2022/. [Kasutatud 09.04.2023].

[27] KnowledgeHut, “What are the Pros and Cons of React,” 2022 [Võrgumaterjal]. Kättesaadav: https://www.knowledgehut.com/blog/web-development/pros-and-cons-of-react. [Kasutatud 12.04.2023].

[28] Josh Goldberg, Learning TypeScript. O'Reilly Media, Inc, 2022.

[29] S. S. Bagui and R. W. Earp, Database Design Using Entity-Relationship Diagrams. Milton: Auerbach Publishers, Incorporated, 2022.

[30] A. Belyaev, “To Delete or to Soft Delete, That is the Question!,” JMIX, 2020 [Võrgumaterjal]. Kättesaadav: https://www.jmix.io/blog/to-delete-or-to-soft-delete-that-is-the-question/. [Kasutatud 17.04.2023].

[31] Microsoft, “What are solutions and projects in Visual Studio?,” 2023 [Võrgumaterjal]. Kättesaadav: https://learn.microsoft.com/en-us/visualstudio/ide/solutions-and-projects-in-visual-studio. [Kasutatud 18.04.2023].

[32] G. Hedge, “Entity Framework Code First v/s Database First Approach,” C# Corner, 2017 [Võrgumaterjal]. Kättesaadav: https://www.c-sharpcorner.com/blogs/entity-framework-code-first-vs-database-first-approach. [Kasutatud 18.04.2023].

[33] A. Zang, “.NET Basics: ORM (Object Relational Mapping),” Telerik, 2022 [Võrgumaterjal]. Kättesaadav: https://www.telerik.com/blogs/dotnet-basics-orm-object-relational-mapping. [Kasutatud 18.04.2023].

[34] E. Musso, “Entity Framework using C#,” C# Corner, 2020 [Võrgumaterjal]. Kättesaadav: https://www.c-sharpcorner.com/article/entity-framework-introduction-using-c-sharp-part-one/. [Kasutatud 18.04.2023].

[35] J. Singh, “Unit of Work in Repository Pattern,” C# Corner, 2018 [Võrgumaterjal]. Kättesaadav: https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/. [Kasutatud 18.04.2023].

[36] Andres Käver, Building Distributed Systems Course material. TalTech 2022.

[37] “What is a REST API?,” Red Hat, Inc. 2020 [Võrgumaterjal]. Kättesaadav: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Kasutatud 18.04.2023].

[38] S. Kharche, “API Versioning in ASP.NET Core,” C# Corner, 2020 [Võrgumaterjal]. Kättesaadav: https://www.c-sharpcorner.com/article/api-versioning-in-asp-net-core-web-api/. [Kasutatud 18.04.2023].

[39] C. Nienaber and R. Suter, “ASP.NET Core web API documentation with Swagger / OpenAPI,” Microsoft, 2022 [Võrgumaterjal]. Kättesaadav: https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-7.0. [Kasutatud 18.04.2023].

[40] R. Gunasundaram, ASP.NET web API security essentials, 1st ed. PACKT Publishing, 2015.

[41] Facebook Open Source, “Create React App,” [Võrgumaterjal]. Kättesaadav: https://create-react-app.dev/. [Kasutatud 22.04.2023].

[42] Redux, “Getting Started with Redux,” [Võrgumaterjal]. Kättesaadav: https://redux.js.org/introduction/getting-started. [Kasutatud 22.04.2023].

[43] Algolia, “Site Search & Discovery powered by AI,” [Võrgumaterjal]. Kättesaadav: https://www.algolia.com/. [Kasutatud 23.04.2023].

[44] T.L. Saaty, “How to Make a Decision: The Analytic Hierarchy Process,” Ce.S.E.T. Aestimum, no. 24, 2009, doi: 10.13128/Aestimum-7138.

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

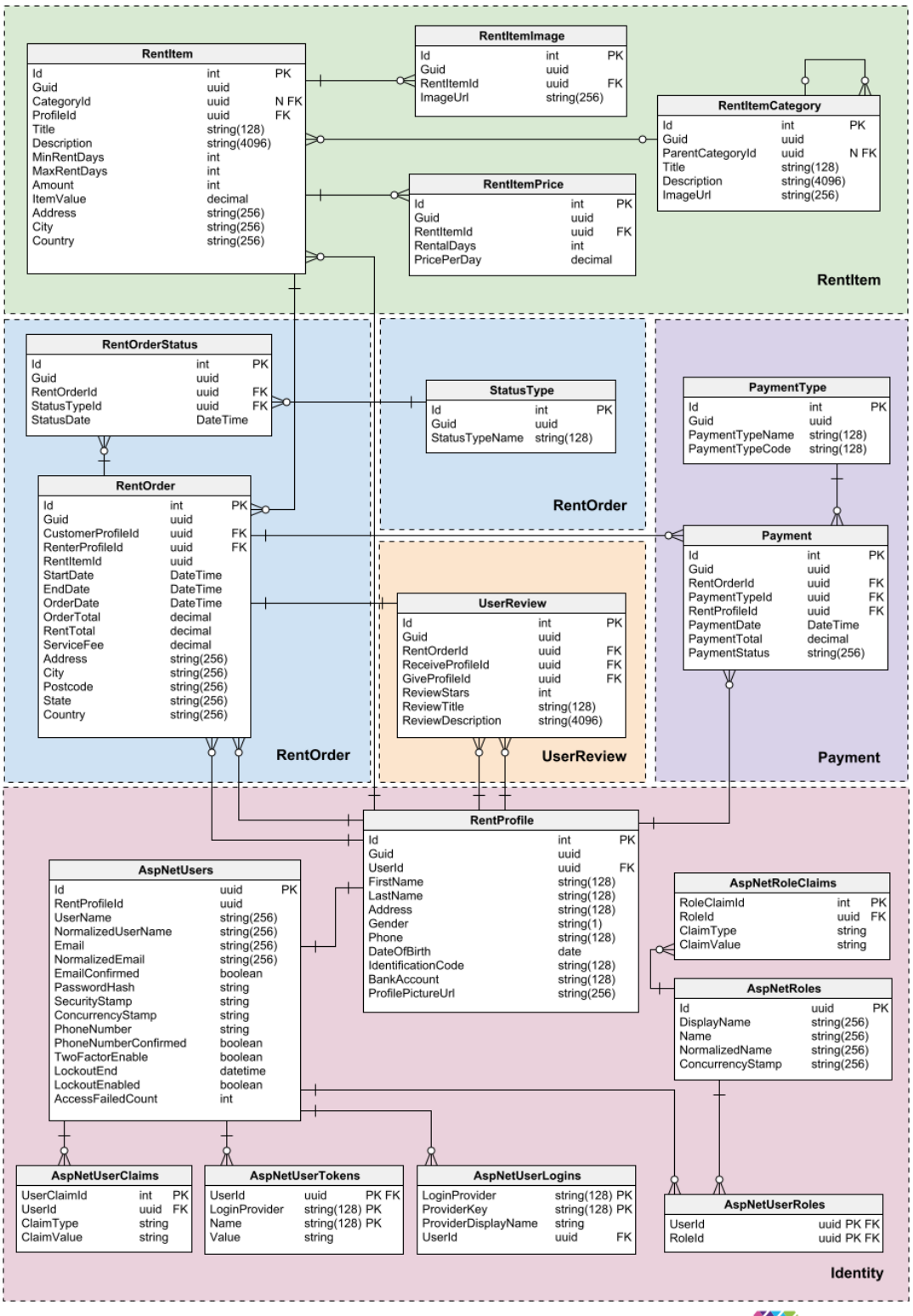
Mina, Karl Markus Kõivastik

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Ringmajandust toetav rentimisplatvorm“, mille juhendaja on German Mumma ja kaasjuhendaja Lauri Anton
   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.

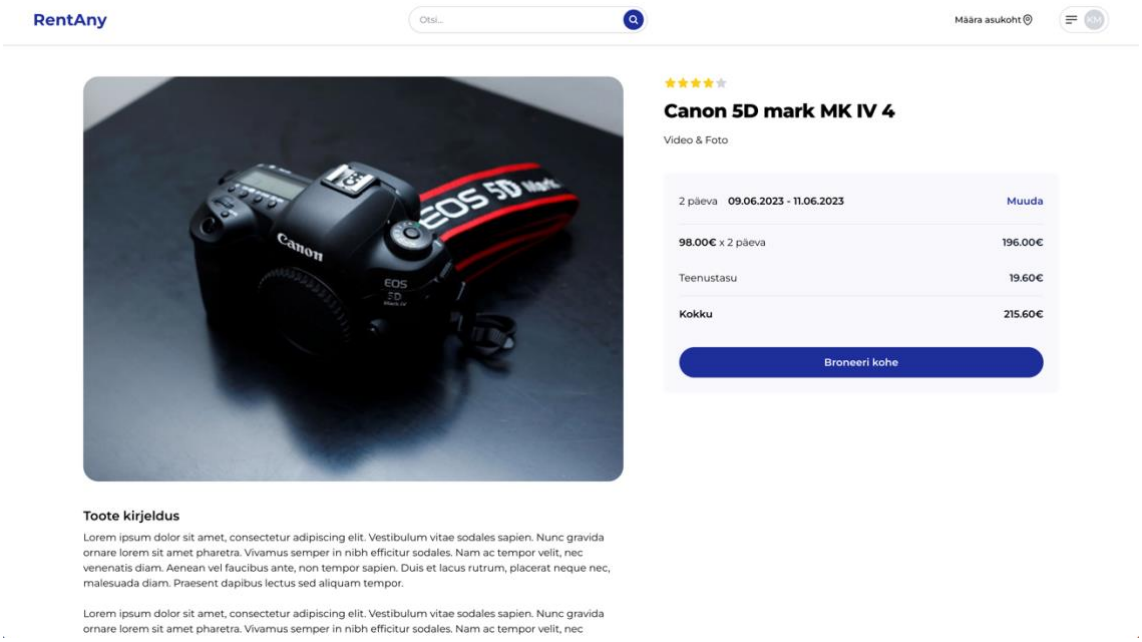
15.05.2023

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 – Andmebaasi olemi-suhte diagramm



Lisa 3 – Renditoote detailvaade



Lisa 4 – Renditellimuse detailvaade