TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Joonas Kaal 206769IADB

Veebipõhise väikesemahulise elektroonikakomponentide lao kasutajarakendus

Bakalaureusetöö

Juhendaja: Andres Kütt
MSc

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: Joonas Kaal

15.05.2023

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua veebipõhine kasutajarakendus, mis lahendab väikesemahuliste elektroonikakomponentide ladude probleemi. Rakenduse loomise eesmärk on anda elektroonikaprojektide ehitajatele parem ülevaade oma laost. Rakendus on loodud tuleviku arengut silmas pidades ja on saadaval avatud lähtekoodiga tarkvarana.

Arendusprotsessi käigus luuakse veebirakendus, mis võimaldab kasutajatel oma komponente ja projekte salvestada ning hallata. Komponente saab lisada, vaadata, uuendada ja kustutada. Projekte saab samuti luua, vaadata, muuta ja kustutada, projekti saab loodud komponente lisada ning projekte saab ehitada. Rakendus suudab koostada ka aruande, mis näitab millised komponendid hakkavad laost otsa saama. Arendus on jaotatud neljaks alamosaks: veebiteenuse loomine, selle testimine, dokumenteerimine ja publitseerimine.

Töö oluline osa on olemasolevate lahenduste ja platvormide analüüsimine, et näha mis töötab ja mis on ebavajalik. Õigete tehnoloogiate valmiseks ning projekti struktuuri loomiseks ja testimiseks pühendati samuti tublisti aega ja energiat. Arendustöö tulemuseks on töötav veebirakendus

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 54 leheküljel, 7 peatükki, 22 joonist, 1 tabel.

Abstract

Web-based User Application for Small-scale Electronic Component Storage

The aim of current thesis is to create a web-based user application that solves the issue with small-scale electronic component storage. The purpose of creating the application is to allow electronic project builders to have a better overview of their storage. The application is built with future development in mind and is available as open-source software.

During the development process, a web application is created that allows users to save and manage their components and projects. Components can be added, viewed, updated, and deleted. Projects can also be added, viewed, updated, and deleted, but also allow adding created components into projects and building the project. The application can generate a report, which shows what components are getting low. Development is divided into multiple parts: creating the web service, testing, documenting, and publishing.

An important part of the work is analysing existing solutions and platforms: finding what works and what is unnecessary. A lot of thought is put into finding the correct technologies to use and how to structure and test the project. The result of the development is a working web application.

The thesis is in Estonian and contains 54 pages of text, 7 chapters, 22 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
BOM	*Bill of Materials*, komponentide nimekiri
CAD	*Computer-aided design*, raalprojekteerimine
CRUD	*Create-read-update-delete*, neli põhilist funktsiooni andmete hoiustamiseks
CSS	*Cascading Style Sheets*, stiililehe keel
CSV	*Comma-separated values*, piiritletud tekstifail mis kasutab väärtuste eraldamiseks koma
ERD	*Entity Relationship Diagram*, olemi-suhte diagramm
frontend	Kasutajaliides, osa rakendusest mida lõppkasutaja näeb
HTML	*HyperText Markup Language*, veebilehtede märgendamise keel
IBN	*International Barcode Network*, Rahvusvaheline vöötkoodide võrk
IDE	*Integrated development environment*, integreeritud programmeerimiskeskkond
JSON	*JavaScript Object Notation*, lihtsustatud andmevahetusvorming
lightweight	Efektiivne, vähese keerukusega
MIT	*Massachusetts Institute of Technology*, Massachusettsi Tehnoloogiainstituut
multithreading	Hargtöötlus
MVC	*Model-View-Controller*, tarkvara arhitektuuri muster
ORM	*Object-relational mapping*, objektide ja relatsioonilise andmete tõlkimine
PCB	*Printed circuit board*, trükkplaat
PDF	*Portable Document Format*, elektrooniliste dokumentide vorming
SaaS	*Software as a Service* – Tarkvara litsentsimise ja tarnimise mudel, mille puhul pilveteenuse pakkuja haldab rakendusi ja teeb need Interneti kaudu lõppkasutajale kättesaadavaks
URL	*Uniform resource locator*, internetiaadress
WSGI	*Web Server Gateway Interface*, veebiserveri lüüsi liides

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Elektroonikakomponendid on elektroonikaseadmete koostisosad, nende hulka kuuluvad näiteks transistorid, takistid, kondensaatorid, jms. Elektroonikakomponente kasutatakse elektroonikaseadmete ehitamiseks, remontimiseks, prototüüpimiseks ja väiketootmiseks.

Igaühel, kes elektroonikakomponentidega tihti tegeleb tekib vajadus organiseerida mingil kujul komponentide ladu, sest muidu muutub aja jooksul vajaliku komponendi olemas olevate hulgast leidmine raskeks. Seepärast on komponentide lao majandamine kõigi elektroonikaga tegelevate inimeste ja organisatsioonide probleem. Väikesemahuliste ladude jaoks ei ole väga häid tasuta lahendusi, käesolev töö analüüsib väikeste ladude omanike probleeme ja ka mõnesid eksisteerivad lahendusi.

Antud bakalaureusetöö eesmärk on töö raames luua vabavaraline elektroonikakomponentide lao veebirakendus, mis on MIT[1] litsentsiga kõigile kättesaadav. Veebikeskkond võimaldaks sisestada ja salvestada komponente ning koostada komponente sisaldavaid projekte. Lisaks on veel mõned kasulikud funktsionaalsused nagu projektide importimine, komponentide hulgi importimine ja projektide ehitamine.

Lõputöö autor arendab rakendust kliendile, kes on elektroonikakomponentide ehitamisega tegelenud pikka aega. Klient annab oma kogemusele ja vajadustele tuginedes rakenduse arendamise käigus pidevalt tagasisidet, et rakendus oleks sobiv lahendus eelnimetatud probleemidele ning vastab ka kliendi soovidele.

[1] Massachusettsi Tehnoloogiainstituudi litsents lubab tasuta kasutada kopeerida, muuta, ühendada, avaldada, levitada, all-litsentsida ja müüa tarkvara koopiaid

2 Metoodika

Käesoleva lõputöö käigus uuritakse olemasolevaid analoogseid rakendusi ning tutvutakse pakutavate võimaluste ja puudustega. Analüüsi järeldusel pakutakse sobiv lahendus, mis sobib kliendi nõuetega, jääb lõputöö skoopi ning oleks ka edasiarendatav.

Lõputöö neljandas osas analüüsitakse sobivaid tehnoloogiaid rakenduse loomiseks, valitakse kõige sobivaim raamistik ning võrreldakse erinevaid andmebaasisüsteeme ja arenduskeskkondi. Lisaks kirjeldatakse ka loodava veebirakenduse arhitektuuri ja disaini.

Rakenduse arendusprotsessi kirjeldatakse nii serveri- kui ka kasutajaliidese poolselt. Rakenduse funktsionaalsuse testimiseks kasutatakse moodulteste. Lõpuks analüüsitakse lõputöö raames valminud rakenduse kvaliteeti ja funktsionaalsust ning hinnatakse võimalusi süsteemi edasi arendamiseks.

3 Ülevaade probleemist

Tavaliselt ei osteta elektroonikakomponente ükshaaval, niipalju kui vaja, vaid ostetakse vähemalt mingi liiasusega, sest iial ei tea millal mingi komponent arenduse käigus kannatada saab või millal juba kord kasutatud komponenti uuesti vaja läheb. Mõnedel juhtudel on komponentidel ostmise miinimumkogus, näiteks saab osta ainult 5 või 10 ühikut korraga. See aga tähendab, et ehitajatel tekib vajadus pidada vähemalt väikest ladu. Isegi väikeses laos on tavaliselt sadu erinevaid komponente ja omamata ülevaadet lao seisust võtab komponentide leidmine asjatult kaua aega ning võib juhtuda, et vajalikke komponente pole laos kui neid vaja on, või komponente, mis juba olemas on, ostetakse asjatult juurde. Samas ei ole väikesemahulise lao majandamiseks vaja liiga keerulist süsteemi ning samuti ei ole majanduslikult otstarbekas näiteks kuumakse põhise SaaS[1] süsteemi kasutamine.

Lao rakenduse loomise idee tekkis kliendil, kellel ülal väljatoodud probleemid on olnud juba mitmeid aastaid. Kliendi soov on enda arvutis lokaalselt või võimalusel kohalikku serverisse paigaldada rakendus, mis aitaks tal järge pidada oma lao seisust. Samuti on tähtis, et lahendus kasutaks Pythoni programmeerimiskeelt ja oleks võimalik kasutada MariaDB andmebaasisüsteemi. Programmeerimiskeele ja andmebaasi nõuded tulenevad kliendi soovist kasutada rakendust juba olemasoleval serveril ilma, et selle administreerimine keerukus kasvaks.

3.1 Eksisteerivad lahendused

Järgnevas osas on välja toodud ja analüüsitud mõnesid eksisteerivaid lahendusi eelnimetatud probleemide lahendamiseks, kliendi nõudeid silmas pidades. Enamus nendest rakendustest on pilveteenused, mille jaoks tuleb end kasutajaks registreerida. Vastavalt kliendi nõuetele peaks olema uus lahendus kasutatav ilma Interneti ühenduseta,

[1] Tarkvara litsentsimise ja tarnimise mudel, mille puhul pilveteenuse pakkuja haldab rakendusi ja teeb need Interneti kaudu lõppkasutajale kättesaadavaks [55]

kuid pilveteenust kasutavate rakenduste analüüsimine annab ülevaate vajalikest funktsionaalsustest uuele lahendusele.

3.1.1 Inventory Management by Mouser

Mouser Electronics on firma, mis asustati aastal 1964 ning tegeleb pooljuhtide ja elektroonikakomponentide edasimüümisega [1]. Mouseri laohaldusplatvorm on veebipõhine süsteem, mis võimaldab organisatsioonidel kergelt hallata ja jälgida oma elektroonikakomponente ning nendega seotud tarvikute seisu [2].

Veebiteenuse kasutamiseks tuleb algselt luua endale konto ja lisada oma organisatsioon. Juba see teeb eraisikule platvormi kasutamise ebameeldivaks, sest organisatsiooni lisamine on kohustuslik ja selle lisamiseks tuleb täita väljad nagu postiindeks ja telefoninumber. Pärast organisatsiooni lisamist tuleb lisada üldine asukoht, näiteks laohoone. Nüüd on ka võimalus oma organisatsiooni lisada kasutajaid ning on aru saada, et rakendus on rohkem mõeldud suurtematele organisatsioonidele, kus on mitu erinevat füüsilist asukohta ja palju erinevaid töötajaid.

Rakenduse kodulehel on juhtpaneel, mis suunab igale olemasolevale featuurile. Uue komponendi lisamine on kerge ning lisaks olulistele väljadele nagu nimi, tootja ja kogus on võimalik ka lisada pilt, tarnija, kirjeldus, miinimum kogus, hind, staatus, jms. Loendist on võimalik komponente otsida nime, tootja ja kirjelduse järgi. Lisaks on võimalik komponente lisada ka Mouseri pakendi sildil olevat IBN[1] vöötkoodi lugedes. Samuti on võimalik mitu komponenti korraga lisada, täites ära Exceli mallis olevad väljad. Komponendi lattu lisamine ja väljastamine toimub läbi laotehingute ning seetõttu on väga keeruline ükshaaval komponentide koguseid muuta. Komponente on võimalik ka laenata, mis eemaldab vastavad kogused kuid saadab teavituse sätitud kuupäeval, et komponente tagasi lisada.

Veel üks featuure on aruannete genereerimine. Võimalik on valida asukohtade vahel ning erinevate kategooriate vahel nagu praegune inventuur, inventuuri ajalugu, viivislaenud, jms. Lisaks on ka võimalus sättida ajavahemik, et saada täpsem aruanne.

[1] *International Barcode Network*, Rahvusvaheline vöötkoodide võrk

Lisaks kõigele on olemas ka mobiilirakendus, millel on olemas kõik samad võimalused, mis veebirakendusel.

Sellegi poolest on teenusel palju puudujääke, millest üks kõige tähtsamaid on projektid. Kuigi on võimalik teha tehinguid, milles on mitmeid erinevaid komponente, siis ei ole see piisavalt ülevaatlik ning taaskasutatav projektide ehitamiseks. Lisaks on liiga palju ebavajalikku funktsionaalsust, näiteks pole eraisikul vaja jälgida oma lao hinnaseisu või komponente ära laenata. Nagu ülalpool juba mainitud, on rakendus mõeldud suurtematele organisatsioonidele, kes keskenduvad mitmele asukohale ja sealsetele komponentidele ning nende ajaloole.

3.1.2 Digi-Key myLists

Digi-Key Electronics on asustatud aastal 1972 ja on tuntud elektroonikakomponentide ning automaatikatoodete turustaja üle maailma [3]. Digi-Key myLists on veebiteenus komponentide loendi tegemiseks, komponentide info lugemiseks ja hinnapakkumiseks teisendamiseks [4].

Rakendust on võimalik kasutada ilma registreerimata ning loendi alustamine on kiire ja kerge. Komponentide lisamine on keerukam kui varasematel lahendustel, sest lisamine toimub komponendi numbri järgi. Pärast komponendi leidmist on võimalik sellele lisada kogus, kliendi viide, tootmiskao protsent, osa nimi skeemis ja kirjeldus. Võimalus on ka hulgi lisada kuid ka siis on vaja teada iga komponendi numbrit. Numbri järgi lisamisel on plussiks see, et kuna iga komponent on Digi-Key veebipoe nimekirjas, siis on komponentide juurde tellimine ja hinna vaatamine väga lihtne. Samas on miinuseks see, et alati ei pruugi teada või leida õige komponendi numbrit.

Loendit on võimalus eksportida erinevatesse formaatidesse ning seda on võimalik jagada meili teel. Vastavat faili saab hiljem kasutada, et loendit importida. Loendit on võimalus ka 30 päevaks pilve salvestada kuid selleks tuleb end registreerida Digi-Key leheküljel.

Rakendus on mõeldud rohkem projekti ja komponentide hindade leidmiseks ning jagamiseks töökaaslaste või partneritega. Komponentide laona on seda teenust keeruline kasutada, sest loendi vaatamiseks ja salvestamiseks pidevalt failide importimine ning eksportimine on väga tülikas.

3.1.3 Binner

Binner on avatud lähtekoodiga elektroonikaosade inventuuri jälgimise süsteem. Binner on C# keeles kirjutatud .NET Core veebirakendus, mida on võimalik jooksutada eraldiseisva serverirakendusena või teenusena [5]. Binneril on ka pilvepõhine rakendus Binner Cloud kuid selle tasuta versiooni kasutamine on limiteeritud ning seda eraldi ei ole mõtet analüüsida.

Binneri paigaldamine ja kasutamine võtab rohkem aega kui teiste lahendustega, sest tegemist pole pilveteenusega. Sellegi poolest on juhend selgesti kirjutatud ning ilma suurema mureta saab rakenduse tööle. Lisaks on võimalus muuta erinevaid rakenduse seadeid nagu pordi number või andmebaasiühendus.

Komponentide lisamine on lihtne ning töötab samamoodi nagu juba ülalpool väljatoodud. Komponentidele on lisaks võimalik lisada ka pilt ja PDF[1] tüüpi fail. Inventuurist on võimalik otsida ning inventuuri on ka võimalik eksportida erinevatesse faili formaatidesse mida saab hiljem importida.

Binner pakub integreerimisvõimalusi ülaltoodud teenustest Mouser ja Digi-Key ning lisaks veel teistest populaarsetest API[2]'dest nagu OctoPart. Andmeintegratsioon on veel arenduses ning tulevikus on plaanis ka lisada integratsioon Arrow ja AliExpressiga.

Binneril on ka võimalus projektide loomiseks ja ehitamiseks, kuid see on tehtud ebavajalikult keeruliseks. Projekti komponentide lisamine on keerukas ja algselt arusaamatu, lisaks on lisatud erinevaid valikulisi välju mida pole tingimata vaja. Puudub võimalus elektroonikaprojektide disainimise rakendustest nagu EagleCAD eksporditud BOM[3]'i importimine.

Rakendusele on lisatud ka erinevad abivahendid nagu Ohmi seaduse kalkulaator ja vöötkoodi lugeja.

---

[1] *Portable Document Format*, elektrooniliste dokumentide vorming

[2] *Application Programming Interface*, rakendusliides

[3] *Bill of Materials*, komponentide nimekiri

Binner on väga hea rakendus kuid see pole oma puudusteta. Üldiselt on kasutajamugavus hea kuid leidub palju kontrastsusvigu. Samuti on lisatud liiga palju ebavajalikku funktsionaalsust nagu komponentide ja lao hinnaväärtus. Kuigi enamus väljad võib täitmata jätta on siiski liiga palju valikuid ja välju mida peab täitma, et komponente ja projekte lisada. Kõige suurem ebasobivus seisneb selles, et kliendil on vaja Pythoni keeles kirjutatud rakendust, sest olemasolevale serverile ei soovita paigaldada uusi süsteeme.

3.1.4 BOMIST

BOMIST on elektroonika osade inventuuri ja tootematerjalide haldamise tarkvara. BOMIST'i kõige suurem erinevus on see, et tegemist pole veebipõhise rakendusega. Rakendus on saadaval Windowsi, macOSi ja ka Linuxi operatsioonisüsteemidel. BOMIST'i on võimalik kasutada ilma Interneti ühenduseta ning ka töökaaslastega, kuid rakendusel on erinevad kuumakse plaanid, mis piiravad rakenduse kasutamist [6].

Rakenduse kasutamiseks tuleb algselt luua töölaud. Seda töölauda on võimalik importida ja eksportida, et loend ühest arvutist teise saada. See teeb rakenduse kasutamise natuke ebameeldivaks, sest koduvõrgus ei pruugi alati ühe arvutiga töötada, kuid komponentide loendit igakord selleks salvestada ja üles laadida on ebameeldiv.

Komponendi lisamiseks on vajalik täita 3 välja: komponendi tüüp, number ja tootja. Samas on võimalik täita ka väljad, mis on juba varasemates lahendustes läbi käidud nagu miinimum kogus, komponendi väärtus, kao protsent, jms.

BOMIST'i kõige tugevam külg on projektide lisamine ning ülevaade. Projekti loomine on lihtne ning selleks on vaja määrata lihtsalt nimi ja valikuliselt ka klient. Komponentide lisamine projektile on samuti väga lihtne ning on olemas ka võimalus CSV[1] tüüpi faili importimiseks. Projekte on võimalik planeerida ja ehitada ning ehitamise ajal on võimalik projekti jaoks vastav kogus komponente laost eemaldada. Ehitamise saab ka pooleli jätta ja hiljem jätkata.

Kuigi rakendus töötab väga hästi on selle kõige suurim miinus kallis kuumakse. Tasuta versioonil on võimalik laos hoida ainult 100 komponenti ning piirangu eemaldamiseks

[1] *Comma-separated values*, piiritletud tekstifail, mis kasutab väärtuste eraldamiseks koma

tuleb ligikaudu 30 eurot kuus maksta. Samuti on jällegi lisatud liiga palju ebavajalikku funktsionaalsust nagu ka eelmainitud lahendustel.

3.1.5 PartsBox

PartsBox on veebirakendus, mis võimaldab kontrollida oma elektrooniliste osade laoseisu, osta ning tellida komponente ja jälgida elektroonikaprojektide kulusid. See jälgib, kus komponente hoitakse, millised on praegused komponentide kogused ja millised komponente millistes projektides kasutatakse [7]. PartsBox on tasuta ehitajatele ning rakendusel on ka tasuline plaan millega kaasnevad mitmed erinevad featuurid nagu lisakasutajad, osade tootmiskao arvutused jms. PartsBoxi kasutamiseks tuleb algselt ennast kasutajaks registreerida.

PartsBoxi on komponente võimalik lisada kolme erinevat viisi:

1. Kasutades komponendi osanumbrit, mis leiab komponendi kasutades Octoparti API'd ning lisab selle lattu täpsete andmetega;
2. Lisades lokaalse komponendi, andes sellele nime. See valik on mõeldud üldistele ja nimetutele komponentidele, PCB[1]'dele, kohandatud osadele, mehaanilistele osadele ja kõigele muule millel pole täpset osanumbrit;
3. Lisades metakomponendi ehk komponendi mis on asendatav või teisega ühesugune.

Komponente saab ka lisada vöötkoode skaneerides kuid mitme komponendi korraga lisamine, kasutades näiteks Exceli faili, puudub. Olemasolevast laost on võimalik eksportida oma andmed JSON[2] failis mida hiljem saab kasutada, et samad andmed importida.

PartsBox võimaldab projektide koostamist ja ehitamist. Projektile komponentide lisamine on kerge ning lisaks on ka võimalus projekti BOM faili importimiseks erinevatest CAD[3]

[1] *Printed circuit board*, trükkplaat

[2] *JavaScript Object Notation*, lihtsustatud andmevahetusvorming

[3] *Computer-aided design*, raalprojekteerimine

programmidest. Projekte on võimalik ka ehitada, mis eemaldab vastavad komponentide kogused laost.

PartsBox on kõige kasutajasõbralikum lahendus kõikide teistega võrreldes. Kasutamine on kerge ja arusaadav ning lihtne. Puuduvad mõned tähtsamad funktsionaalsused nagu hulgi komponentide lisamine.

3.2 Uue lahenduse skoop

Eksisteerivate lahenduste peamine puudus ja põhjus miks kliendile ükski neist ei sobi on, et rakendust oleks vaja lokaalselt jooksutada juba olemasoleval serveril. See tähendab, et on konkreetsed eelistused nii programmeerimiskeele kui andmebaasisüsteemi osas. Niimoodi on võimalik rakendust kasutada erinevates arvutides kohalikus võrgus ning ilma Interneti ühenduseta. Samuti, kuna tegemist on väikesemahulise laosüsteemiga, siis pole vaja enamust keerukamatest võimalustest mida pakuvad olemasolevad lahendused, näiteks nagu laohind, kaoprotsent, jms.

Eelnevalt nimetatud probleemi lahendamiseks pakub lõputöö autor lahendusena välja veebirakenduse, mis eelkõige vastab kliendi tehnilistele nõuetele. Rakendus on suhteliselt väikese elektroonikakomponentide lao organiseerimise süsteem, mis on orienteeritud lihtsusele ja kasutusmugavusele.

Elektroonikakomponendi lisamisel võib komponendil olla ainult nimetus (nagu näiteks mikroskeemidel või transistoridel) aga võib olla ka nimetus ja väärtus (nagu näiteks takistitel ja kondensaatoritel). Kuna tegemist on väikese laoga ei ole vaja vahet teha suure hulga komponentide parameetrite vahel. Näiteks erinevat tüüpi korpustega sama tüüpi mikroskeemid on reeglina samas sahtlis. Vajadusel eristatakse komponente kirjelduse sisuga. Samuti ei ole vaja sahtleid (asukohti) eraldi andmebaasis hoitavate olemitena käsitleda. Asukoht märgitakse lihtsalt komponendi juures teksti kujul.

Rakendus võimaldab ka mitu komponendi ühikut korraga lattu lisada.

Rakendus toetab ka projektipõhiseid tegevusi. Projekt on sisuliselt komponentide komplekt mida on vaja ühe kindla elektroonikaseadme koostamiseks. Projekti saab käsitsi kokku panna, tekitades nimekirja süsteemis olevatest komponentidest aga on ka võimalus CAD tarkvaras koostatud komponentide nimekirja ehk BOM’i üles laadimist.

Lahendus võimaldab ka aruande koostamist, milles on kirjas komponendid mis hakkavad laost otsa saama.

Lõputöö eesmärgiks ei ole raha teenimine ning valminud rakenduse kood jääb vabavaralise tarkvarana saadavaks koodihalduse keskkonnas. Rakenduse juures on ka dokumentatsioon, mis lisaks rakenduse üldisele kirjeldusele sisaldab ka kiirjuhendit paigaldamiseks ja kasutusele võtuks. Igaühel on võimalik valminud veebirakendust tasuta endale paigaldada ning vajadusel edasi arendada.

4 Loodava veebirakenduse analüüs

Järgnevates peatükkides on täpsustatud lahenduse nõudeid, analüüsitud tehnoloogiate ja süsteemide valikud ning kirjeldatud loodava rakenduse disaini ja arhitektuuri.

4.1 Nõuete täpsustamine

Kirjeldatud nõuded katavad antud lõputöös määratud skoopi. Nõuete määramisel võeti kuulda kliendi soovid, mis on järgnevalt tõlgendatud ja kirja pandud vajalikke nõuetena.

4.1.1 Funktsionaalsed nõuded

Süsteemi uue komponendi lisamisel peab saama komponendi kohta märkida järgmised andmed:

- Komponendi nimi;
- Komponendi asukoht (võib ka tühi olla);
- Komponendi väärtus (võib ka tühi olla);
- Komponendi kirjeldus (võib ka tühi olla);
- Lisatavate komponentide hulk (võib ka 0 olla);
- Minimaalne kogus komponente, mis laos peaks olemas olema (võib ka 0 olla);
- URL[1], mis viitab komponendi võimaliku hankimise asukohale (võib ka tühi olla).

Olemasoleva komponendi kõiki andmeid peab saama muuta.

Süsteem peab võimaldama komponente laost otsida, et kindlaks teha nende olemasolu ja asukoht. Komponentide otsing peab olema tõstutundetu ning peab otsima sisestatud fraasi nii komponentide nimedest, kirjeldustest kui ka väärtustest.

[1] *Uniform resource locator*, internetiaadress

Süsteem peab võimaldama komponentide laost väljastamist. Kui komponentide arv väheneb nullini siis ei tohi komponenti automaatselt andmebaasist ära kustutada. Selleks on eraldi komponendi kustutamise funktsionaalsus.

Süsteem peab võimaldama projektide loomist kas alustades tühjast projektist või kopeerides olemasoleva projekti sisu uude projekti. Igal projektil peab olema kasutaja muudetav nimi ja kustutamise võimalus. Projektile saab lisada komponente ühendades need ära laos leiduvate komponentidega. Lisaks saab projektile määratud komponendile sättida ka koguse, mitut ühikut projekti ehitamiseks vaja läheb, ning vajadusel ka kommentaari.

Süsteem peab võimaldama projektile komponente lisada üleslaetud komponentide nimekirjast. Algselt on vaja võimalust EagleCAD tarkvara versioon 7 väljastatava BOM'i CSV formaadis importimist. Importimisel peab süsteem:

- Üritama automaatselt nime ja väärtuse järgi BOM'is olevaid komponente andmebaasis olevatega sobitada;
- Võimaldama kõikide, ka automaatselt valitud, komponentide koguseid käsitsi korrigeerida;
- Juhuks kui komponenti pole laosüsteemis, võimaldama luua uus komponent ilma importimisfunktsiooni katkestamata.

Süsteem peab võimaldama luua aruannet komponentidest mida on laos vähem või sama palju kui minimaalne nõutud kogus.

4.1.2 Mittefunktsionaalsed nõuded

Tehnilised nõuded tulenevad suuresti vajadusest kasutada rakenduse jaoks juba olemasolevat serverit. Kuna süsteemi peab olema võimaik kasutada erinevatelt arvutitelt millel võib olla ka erinev operatsioonisüsteem, siis on veebipõhine rakendus loogiline valik. Järgnevalt on loetletud peamised mittefunktsionaalsed nõuded:

- Rakendus peab olema veebipõhine;
- Rakendus peab olema kirjutatud Pythonis;

- Andmebaasimootoriks tuleb kasutada MariaDB SQL serverit.

Süsteemi loomisel tuleb arvestada tööprotsessidega, et kasutamine oleks mugav ja kiire. Väikese ekraaniga mobiiliseadmetega ei ole vaja arvestada. Süsteemi kasutatakse süle- või lauaarvutitelt ja oluline on, et kasutajaliides neil hästi töötaks.

Tooteühiku ehk projekti komplekteerimisel on vaja laost korraga välja võtta ühe või mitme tooteühiku jagu erinevaid komponente.

Ühe komponendi väljastamine või lattu lisamine peaks olema hästi lihtne. Näiteks võiks iga komponendi juures olla nupp, mis kogust ühe võrra suurendab või vähendab.

4.2 Tehnoloogiate valik

Veebirakenduste arendamiseks on tehnoloogiate valik aastatega aina suurenenud [8]. Käesoleva lõputöö rakenduse tehnoloogiate valikute juures on arvestatud sellega, et tegemist on populaarsete ja enim kasutatavate tehnoloogiatega, mis võiks mõjuda positiivselt süsteemi edasi arendamisele teiste poolt.

4.2.1 Programmeerimiskeele valik

Kuigi kliendil on nõue kasutada just Pythoni programmeerimiskeelt, siis on kasulik analüüsida ning võrrelda erinevaid valikuid, et näha mis eelised erinevatel keeltel on. Järgnevalt on toodud välja kõik programmeerimiskeeled mida lõputöö autor oma õpingute ajal käsitlenud on ja mis on populaarsed veebirakenduste loomiseks [9]:

- C# – mitmeotstarbeline keel, mida saab kasutada konsooli, Windowsi, veebi- ja mobiilirakenduste arendamiseks. C# on programmeerimiskeel, mis töötab koos .NET raamistikuga. .NET raamistik on levinud keskkond ning tööriistakomplekt Windowsi rakenduste, Windowsi teenuste, veebirakenduste ja veebiteenuste loomiseks, ehitamiseks ja jooksutamiseks [10].

- Java – programmeerimiskeel, mida kasutatakse erinevat tüüpi tarkvarade arendamiseks ning mille suurimaks plussiks on ühilduvus mitme operatsioonisüsteemiga. Tegemist on kaheastmelise keelega, mis tähendab, et kirjutatud kood kompileeritakse enne jooksmist masinkoodiks. Java on

populaarne tänu pikaajalisele ja tõhusale testimisele, uuendamisele ning järjepidevale kohaletoimetamisele [11] [12].

- JavaScript – *lightweight*[1] programmeerimiskeel mida veebiarendajad kasutavad tavaliselt dünaamiliste interaktsioonide loomiseks veebilehtede, rakenduste, serverite või isegi mängude arendamisel. Algselt oli JavaScript üldotstarbeline skriptikeel, et tagada veebilehtede koostalitlusvõime erinevates brauserites ja seadmetes. JavaScript on arenenud koos brauseritega ning tänaseks on olemas ka esimene moderne JavaScripti mootor, V8, mis kompileerib baitkoodi masinkoodiks [13].

- PHP – laialdaselt kasutatav avatud lähtekoodiga üldotstarbeline skriptikeel, mis sobib eriti hästi veebiarenduseks ja mida on võimalik kinnitada HTML[2]'i külge. PHP'd kasutatakse näiteks komplekse sisu haldamiseks, andmebaaside ja seansside logimiseks kui ka tervete e-kaubandus lehtede loomiseks. PHP on väga ühilduv nii mitmete operatsioonisüsteemidega kui ka erinevate andmebaasisüsteemidega [14] [15].

- Python – programmeerimiskeel, mida sageli kasutatakse veebisaitide ja rakenduste ehitamiseks, ülesannete automatiseerimiseks ja andmete analüüsimiseks. Python on tuntud tänu oma lihtsale süntaksile, mis näeb välja nagu inimkeel. Samuti on Python on väga algajasõbralik ning see koos selle mitmekülgsusega on teinud sellest ühe kõige enimkasutatava programmeerimiskeele [16].

Kõik nendest keeltest on laialdaselt kasutatud veebiarenduses ning igaühel neist on olemas põhjalik dokumentatsioon. Järgnevalt on välja toodud autori kogemus ning keele õppimise keerukus eelnimetatud programmeerimiskeelte vahel (Tabel 1).

[1] Efektiivne, vähese keerukusega

[2] *HyperText Markup Language*, veebilehtede märgendamise keel

Tabel 1. Programmeerimiskeelte võrdlus

Keel	**Kogemus**	**Õppimise keerukus**
C#	Hea	Madal [17]
Java	Hea	Keskmine [18]
JavaScript	Keskmine	Keskmine [19]
PHP	Nõrk	Madal [20]
Python	Keskmine	Madal [21]

Nagu varem mainitud on olemasoleva serveri jaoks vajalik, et rakendus oleks kirjutatud kasutades Pythoni programmeerimiskeelt. Python on lihtne, võimas ja hästi disainitud keel ning üks kõige populaarsemaid veebirakenduste arendamiseks. Plussid Pythoni kasutamise juures on näiteks see, et seda on kerge õppida, sellel on kõrge loetavus, palju liideseid ja olemas on hea raamistike valik, mis teeb arendusprotsessi lihtsamaks. Miinusteks on näiteks kõrge mälu kasutatavus, kiiruse piirangud ja ebapopulaarsus mobiilirakendustega [22]. Õnneks ei mõjuta need miinused antud lõputöö lahendust. Mõned näited firmadest, kes kasutavad oma veebirakendustes Pythoni on näiteks Netflix, Instagram ja Reddit [23].

Kuigi lõputöö autoril pole sama palju kogemusi veebirakenduste kirjutamisega kasutades Pythoni programmeerimiskeelt kui teiste keeltega, ei ole selle õppimine ja kasutusele võtmine väga aeganõudev tegevus.

4.2.2 Raamistiku valik

Veebirakenduse ehitamiseks ja arendusprotsessi kiirendamiseks on mõistlik kasutada Pythonile ehitatud veebiraamistikku. Raamistik on pakettide või moodulite kogu, mis võimaldab arendajatel kirjutada veebirakendusi või -teenuseid. Tänu sellele ei pea arendajad käsitlema madala taseme üksikasju nagu protokollid, protsesside haldus, jms. Raamistikud on abiks päringute tõlkimiseks, vastuste loomiseks ja andmete hoiustamiseks [24]. Järgnevalt on välja toodud populaarsemad Pythoni veebiraamistikud:

- Django – Pythoni põhine veebirakenduste raamistik, mis on tasuta ja avatud lähtekoodiga. Django teeb veebirakenduste arendamise lihtsamaks ja kiiremaks vähema koodiga, et arendajatel oleks rohkem aega keskenduda rakenduse arendamisele. Selle kõige suuremad plussid on turvalisus, mitmekülgsus, kiirus, kättesaadavus, suur kasutajaskond ning põhjalik dokumentatsioon. Django sobib suurepäraselt keerukate andmepõhiste veebirakenduste loomiseks [25].

- Flask – väiksemat tüüpi veebiraamistik, mida vahest kutsutakse ka mikroraamistikuks. Flask on kavandatud algusest peale laiendatava raamistikuna. See pakub põhiteenustega tugeva baasi ning lisamoodulitega on võimalik juurde lisada erinevaid teenuseid. Kuna on võimalus valida endale soovitud lisamooduleid, siis lõpptulemuseks on pakett, mis sisaldab ainult vajalikku ilma millegi liigseta. Flask on suurepärane valik algajatele ning väikeste ja keskmiste suurustega veebirakenduste jaoks, mis ei vaja keerukaid funktsioone [26].

- Pyramid – väike, kiire ja maalähedane Pythoni veebiraamistik. Raamistikul on palju sisseehitatud funktsionaalsusi kuid samas on ka väga kohandatav rakenduse vajadustele. Pyramid sobib väikeste rakenduste arendamisele, mis aja jooksul kasvavad suuremaks ning vajavad palju paindlikust ja kohandamist [27].

- Bottle – kiire ja väike mikroraamistik mida on kerge ja tõhus kasutada. Raamistiku jaoks pole vaja muid sõltuvusi peale Pythoni standardteegi. Sellel puuduvad mitmed keerukad võimalused ja kasutusvalmis lahendused nagu teistes raamistikes. Bottle sobib väiksemate veebirakenduste, prototüüpide ja API'de arendamiseks [28].

- CherryPy – objektorienteeritud Pythoni raamistik veebirakenduste arendamiseks. CherryPy on loodud *multithreading*[1] kontseptsiooni alusel, mis annab raamistikule eelise mitme ülesande korraga käsitlemisel. Raamistikku on lihtne õppida ning sellel on hea paindlikkus ja mitmekülgsus. CherryPy sobib väikeste ja keskmiste, suurt jõudlust vajavate, veebirakenduste arendamiseks [29].

[1] Hargtöötlus

Lõputöö autoril puudub kogemus iga eelnimetatud raamistikuga, seega on mõttekas valida suure kasutajaskonna ja põhjaliku dokumentatsiooniga raamistik. Järgnevalt on võrreldud kahte kõige sobilikumat valikut eelnimetatud loendist: Django ja Flask.

Django raamistik on Pythonil põhinev veebirakenduse raamistik, mis aitab arendajatel kergelt ehitada keerulisi andmebaasiga toetatud veebisaite ja -rakendusi. Raamistik katab kõiki veebirakenduse tehnoologiakihte ja laseb arendajal keskenduda oma veebirakenduse loomisele. Djangot eelistatakse agiilse arengu jaoks. Djangot on lihtne ja kiire tööle saada, see on väga turvaline ja skaleeritav, olemas on kergesti jälgitav dokumentatsioon, suurepärane sisseehitatud mallikujundus ja veel. Samas võib raamistiku stiil olla keeruline, sellel on järsk õppimiskõver ja liiga palju võimalusi lihtsatele projektidele [30].

Flaski raamistik on Pythonil põhinev mikroraamistik, millel on minimaalne sõltuvus väliste teekide jaoks. See tähendab, et arendajatel on võimalus valida endale eelistatud kujundusmuster, andmebaas, lisamoodulid, jms. Flask on laiendatav ja kohandatav uute tehnoloogiatega, sobib hästi väiksemate rakenduste jaoks, saab kiiresti prototüüpida ja lihtne on integreerida andmebaase. Sellest eest võib rakenduse hooldus olla raskem, puuduvad turva- ja autoriseerimisfeatuurid ning toetus mitmelehelistele rakendustele [30].

Nende kahe raamistiku seast on lõputöö autor otsustanud kasutada Flask raamistikku. Peamiseks põhjuseks on see, et kuna autoril puudub kogemus mõlema raamistikuga, siis analüüsi käigus on leitud, et Flask raamistikku on kergem õppida ja kasutusele võtta ning see sobib paremini väiksematele rakendustele.

4.2.3 Kasutajaliidese keele valik

Flask võimaldab MVC[1] mustri kasutamist mis tähendab, et puudub vajadus eraldi kliendipoolseks rakenduseks. Flask kasutab Jinja mallide teeki mallide renderdamiseks. Mallid on failid, mis sisaldavad staatilisi andmeid ning ka kohatäiteid dünaamiliste andmete jaoks. Malli rendertakse spetsiifiliste andmetega, et koostada lõppdokument. Flaski rakenduses kasutatakse malle, et renderdada HTML'i, mis kuvatakse kliendi

[1] *Model-View-Controller*, tarkvara arhitektuuri muster

brauseris. Jinja näeb välja ja töötab peamiselt nagu Python. Jinja süntaksi eristamiseks malli staatilistest andmetest kasutatakse spetsiaalseid eraldajaid [31].

4.2.4 Andmebaasi valik

Aastate jooksul on arenenud mitmeid erinevaid andmebaase ja andmebaasi tüüpe. Andmebaasi tüübi valikul võib olla suur mõju sellele, milliseid toiminguid rakendus saab sooritada ja millised funktsioone andmebaasi haldussüsteem arenduse ning käitluse ajal pakub. Üks kõige populaarsemaid andmebaasi tüüpe on relatsiooniline mudel, kus objektid hoitakse andmebaasis ja nendevahelised seosed on esitatud tabelite kujul [32]. Lõputöö raames on analüüsitud ainult relatsioonilisi andmebaase, mis on eelkõige populaarsed ning mille kasutamisega lõputöö autoril kogemusi on. Järgnevalt on toodud välja ja analüüsitud andmebaasisüsteemid:

- MySQL – MySQL on avatud lähtekoodiga, Orcale poolt omatud, relatsiooniline andmebaasi juhtimissüsteem. See põhineb struktureeritud päringukeelel SQL ja töötab nii töölauarakenduste kui ka andmebaasipõhiste veebirakenduste jaoks. Süsteemi tasuta kasutamine on piiratud ning äriliselt kasutamiseks on vaja litsentsi [33].

- MariaDB – MySQL'ist arenenud haru, mis on ka tagasiühilduv MySQL süsteemiga. MariaDB on rohkemate featuuridega, parema jõudlusega ning täiustatud turvalisusega [34].

- PostgreSQL – avatud lähtekoodiga, väga stabiilne andmebaasisüsteem, mis pakub tuge SQL'i erinevatele funktsioonidele. PostegreSQL täiendab veelgi SQL keelt, pakkudes mitmeid võimalusi mis töökoormust paremini skaleerivad ja ressursse reserveerivad. Seda kasutatakse peamiselt paljude mobiili-, veebi- ja analüüsirakenduste andmete salvestamiseks [35].

- Oracle – Mitme mudeliga relatsioonilise andmebaasi haldussüsteem. Oracle süsteem suudab kiirelt tegeleda suurte andmekogustega, kuid andmebaasi litsentsi hinnad on kõrged. Tuntud oma skaleeritavuse ja töökindluse poolest, Oracle on laialdaselt kasutustes suurte ettevõtete rakendustes [36] [37].

- SQLite – faili-põhine relatsioonilise andmebaasi juhtimissüsteem. SQLite on mälu põhine avatud lähtekoodiga teek, mis ei vaja konfigureerimist ega serverit.

Tuntud oma lihtsusele ja väiksele suurusele, SQLite sobib suurepäraselt väiksematele projektidele ja mobiilirakendustele [38].

Nagu ka programmeerimisekeelega oli, on kliendil soov, et rakendus kasutaks MariaDB andmebaasi. MariaDB andmebaasi kasutamisel pole mingit probleemi, kuid igal kasutajal ei pruugi olla serverit kus andmebaasi jooksutada. Selleks, et rakendust oleks võimalik jooksutada ka lokaalselt, enda arvutis, on vajalik ka, et rakendus oleks konfigureeritav kasutama SQLite andmebaasi. See tähendab, et lõputöö lahendus kasutaks nii MariaDB kui ka SQLite andmebaasi vastavalt kasutaja konfiguratsioonile.

4.2.5 Arenduskeskkonna valik

Arenduskeskkonna valikud on jagatud kaheks: koodihalduskeskkonna valik ja koodi arenduskeskkonna valik.

Järgnevalt on analüüsitud koodi- ja versioonihalduskeskkonnad mida lõputöö autor oma õpingute ajal kasutanud on:

- GitHub – veebisait ja teenus, mis hõlbsustab tarkvaraarendust võimaldades koodi salvestada repositooriumitesse ning näha koodis tehtud muudatusi. Lisaks pakub see hostimisteenust ja tööriistu koodi kirjutamiseks, testimiseks ja rakendamiseks. GitHub kasutab projektide haldamiseks versioonikontrolli arendustööriista Git, mis jälgib failide muudatusi ja võimaldab mitmel inimesel sama projekti kallal töötada [39].

- BitBucket – Giti repositooriumite halduslahendus, mis on loodud professionaalsetele meeskondadele. See pakub keskset kohta, kus saab hallata Giti repositooriume, teha koostööd koodi arenduses ja juhendada kasutajaid arendusvoos. Peamine erinevus GitHubi ja Bitbucketi vahel on see, et GitHub on mõeldud rohkem avatud koodile ja BitBucket privaatsele [40] [41].

- GitLab – visuaalne Giti repositooriumite haldussüsteem, mis võimaldab kasutajatel sirvida, auditeerida, liita koodi ja täita muid igapäevaseid toiminguid, mis muidu vajaksid käsurea liidest. GitLab on unikaalne selle poolest, et see pakub isehostitavat toodet GitLab CE, samas kui GitHub on täielikult SaaS teenus [42].

Koodi- ja versioonihaldus keskkonna valikuks sellele lõputööle on GitHub, peamiselt sellepärast, et see on kõige populaarsem ning sobib antud lõputöö lahendusele kõige paremini.

Koodi arenduskeskkonna valikuteks on JetBrainsi tööriist PyCharm ja Microsofti Visual Studio Code. PyCharm on eksklusiivselt disainitud Pythoni, tema pinude, liideste ja lisafeatuuride kasutamiseks, mis teeb koodi arenduse efektiivsemaks. VS Code on tasuta, *lightweight* aga võimas lähtekoodi redaktor, mis jookseb töölaua rakendusena ja on olemas ka veebirakendusena. PyCharm on IDE[1] aga VS Code on koodi toimetaja, mis tänu mitmetele liidestele saavutab IDE'ga samad funktsionaalsused [43] [44].

Kuigi lõputöö autor õppis Pythoni kirjutama kasutades PyCharmi, siis on tal rohkem kogemusi Visual Studio Code arenduskeskkonna kasutamisega. Lisaks, kuna Visual Studio Code pole ainult Pythoni kirjutamiseks, on selle õppimine ja kasutamine kasulik ka tulevikuks teiste keeltega ning seetõttu on antud lahenduse koodi arenduskeskkonnaks valitud Visual Studio Code.

4.3 Arhitektuur

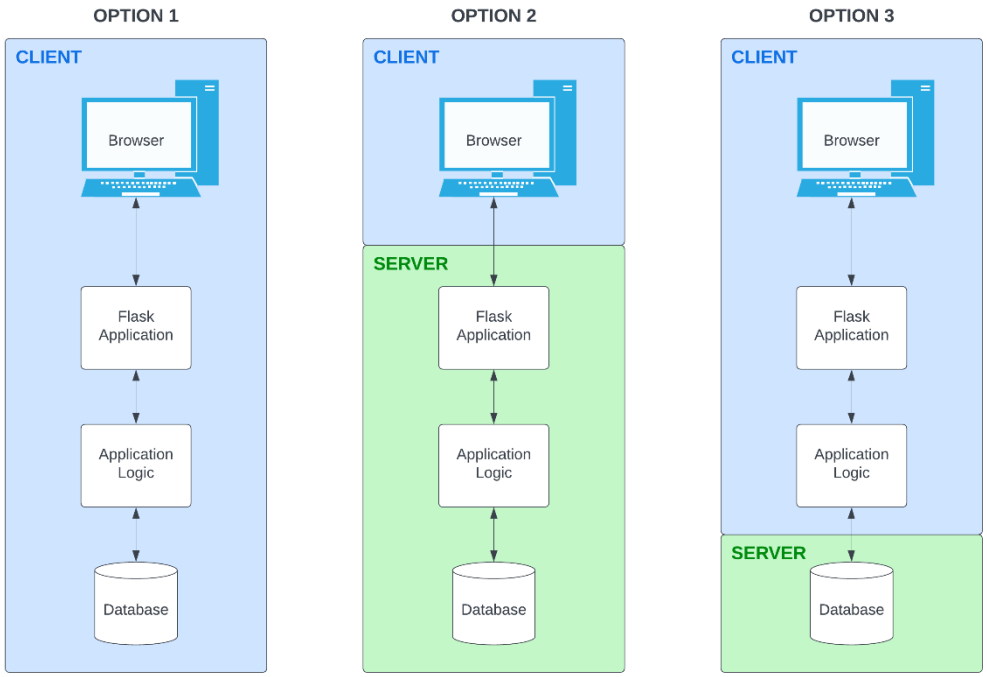
Antud lõputöö rakenduse oluline eesmärk on teha lihtsaks ja mugavaks nii kasutaja kui ka arendaja töövoog. Järgnevates alamosades on kirjeldatud ja analüüsitud loodava rakenduse disaini ja arhitektuuri puudutavaid osasid. Koodi struktuuri kirjeldamine aitab mõista milliseid komponente ja funktsioone rakenduse ehitamiseks vaja on. Kasutajakogemuse ja andmebaasiskeemi kirjeldamine annab ülevaate vajalikest funktsionaalsustest ja vajadustest loodavale rakendusele. Koodi struktuuri ning kasutajakogemuse kui ka andmebaasi disaini kirjeldamiseks on loodud joonis, et teha selle arusaamine lihtsamaks.

4.3.1 Rakenduse üldine arhitektuur

Rakendus on üles ehitatud klient-server arhitektuuri kasutades. Klient-server tarkvaraarhitektuur on mudel, milles tarkvararakendus on jagatud kaheks eraldi osaks, kliendiks ja serveriks, mis suhtlevad omavahel arvutivõrgu kaudu või samas arvutis.

[1] *Integrated development environment*, integreeritud programmeerimiskeskkond

Klient on rakendus või programm mida kasutavad lõppkasutajad ning server on arvutiprogramm, mis pakub teenuseid kliendile [45]. Kuna rakendus on mõeldud kohalikku serverisse paigaldamiseks või oma masinas jooksutamiseks ning kasutajal on ka võimalus määrata millist andmebaasi lahendust rakendusel kasutada, on rakenduse paigaldamiseks mitu erinevat viisi. Järgnevalt on Joonisel 1 välja toodud erinevad rakenduse paigaldamise võimalused ning seletatud nende erinevused. Joonise joonistamiseks on kasutatud Lucidchart veebirakendust.



Joonis 1. Rakenduse paigaldamise alternatiivid

Esimene variant on, et kasutaja jooksutab tervet rakendust oma enda masinas. Niimoodi on kasutuses SQLite andmebaasisüsteem ning kellelgi teisel pole rakendusele ega selle andmetele ligipääsu.

Teine alternatiiv on, et kogu rakendus on paigaldatud võrku. Iga masin kohalikus võrgus saab vastavale aadressile minnes rakendusele ligi ning kõik andmed on kohalikus võrgus jagatud. Andmebaasimootoriks saab olla nii SQLite kui ka MariaDB vastavalt kasutaja konfiguratsioonile.

Kolmas valik on, et ainult andmebaas asub võrgus. See on arvatavasti kõige ebapopulaarsem paigaldamisviis, sest väga ebatõenäoline on, et kellelgi on server koos

andmebaasiga aga rakendust jooksutab enda masinas. Sel juhul kui kellelgi peaks sama andmebaasiühendus olema, siis on andmed jagatud.

4.3.2 Koodi struktuur

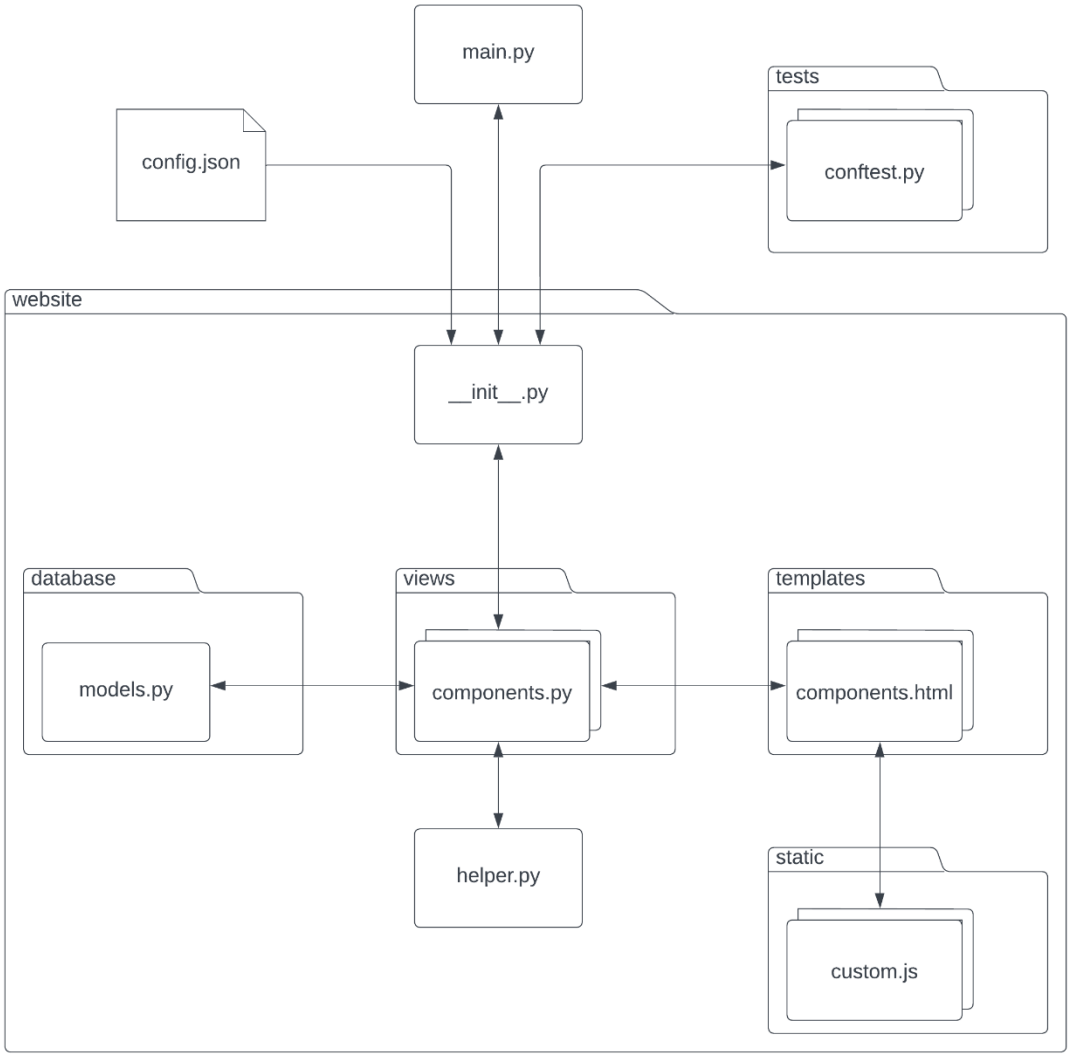
Nagu juba eelnevalt mainitud, on rakenduse arendamiseks kasutatud MVC mustrit. MVC on arhitektuuriline muster, mis jagab rakenduse kolmeks loogiliseks osaks: mudeli osa, vaadete osa ja kontroller. MVC mustri kasutamine kergendab koodi taaskasutamist ja aitab luua stabiilse veebirakenduse struktuuri. Järgnevalt on toodud välja ning kirjeldatud kõik mustri osad [46]:

- Mudel – kõige madalam osa mustrist, mis tegeleb andmete hoiustamisega. Mudel on ühendatud andmebaasiga. Andmete lisamine või pärimine käib läbi mudeli. Kuulab kontrolleri osalt tulevaid päringuid ja saadab vastuseid ainult kontrollerile.

- Vaade – andmete esitamine on läbi vaate komponendi. See osa mustrist loob liidese mida kasutaja näeb. Vaated on loodud andmetega, mis tulevad mudelist aga vaade ja mudel omavahel kunagi ei suhtle.

- Kontroller – mustri peaosa, mis tegeleb mudeli ja vaate vahelise suhtlusega. Kontroller ei pea muretsema andmete käsitlemise loogikaga, see lihtsalt ütleb mudelile mida vaja. Pärast andmete saabumist töötleb kontroller need andmed ära ja annab kõik informatsiooni edasi vaate osale.

Järgnevalt on Joonisel 2 kirjeldatud antud lõputöö veebirakenduse koodi struktuuri ning iga MVC komponendi asukohta. Joonise joonistamiseks on kasutatud Lucidchart veebirakendust.

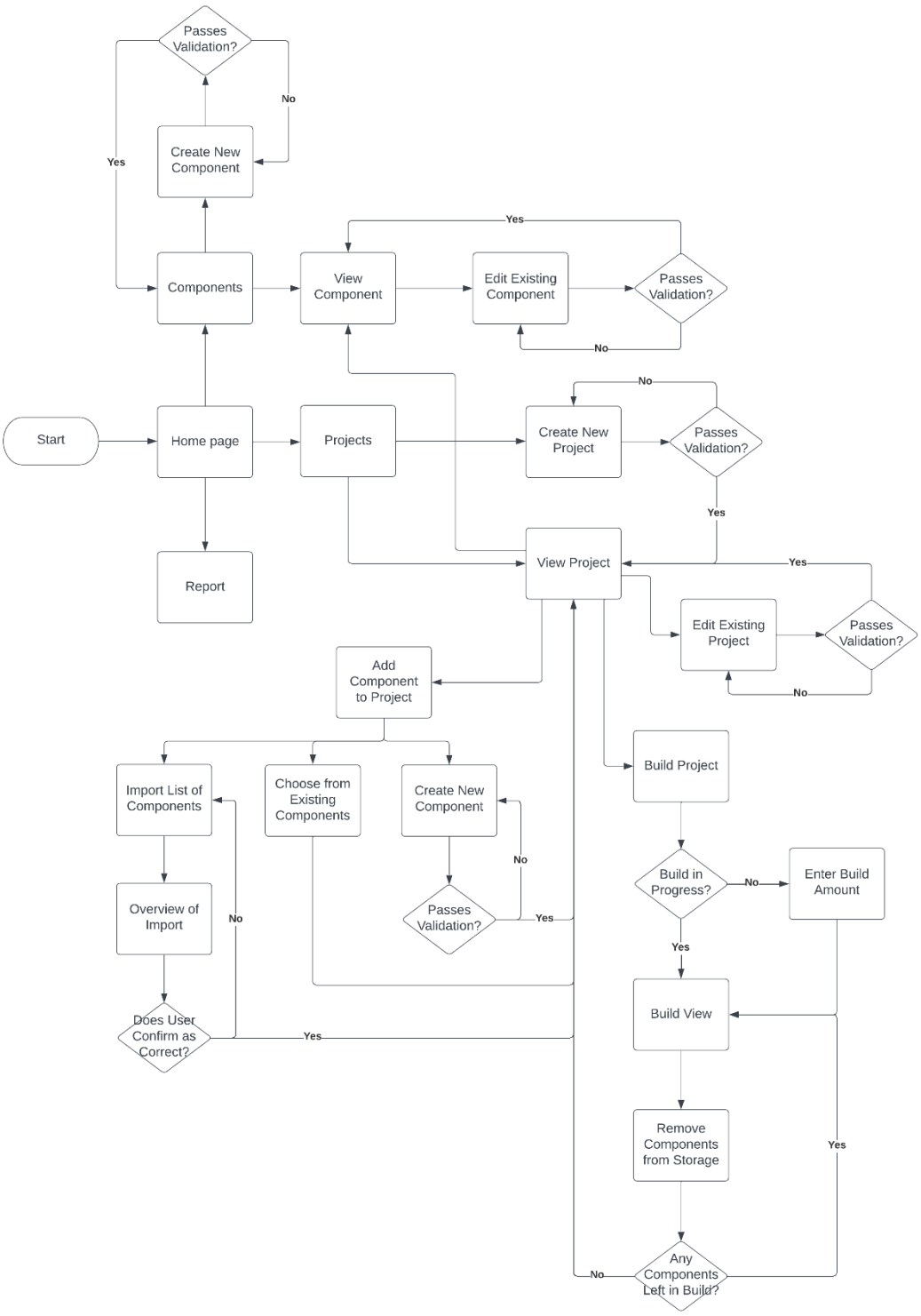


Joonis 2. Veebirakenduse koodi struktuur

Veebirakenduse server käivitatakse kasutades *main.py* faili, mis loob veebirakenduse. *Views* osa veebirakendusest on kontrollerid, mis tegelevad päringute ja vastustega. *Database* osa on mudeli osa, kus päritakse või tehakse muudatusi andmebaasi. *Templates* on vaadete osa rakendusest, kus genereeritakse vaated mida kliendile näidatakse.

4.3.3 Kasutajakogemuse disain

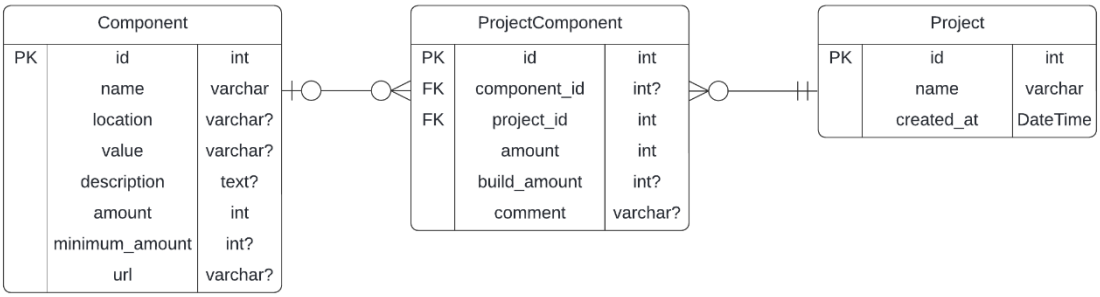
Järgnevalt on joonisena välja toodud rakenduse kasutajakogemuse diagramm (Joonis 3). Joonise loomiseks on kasutatud Lucidchart veebirakendust.



Joonis 3. Rakenduse kasutajakogemuse diagramm

4.3.4 Andmebaasi disain

Lähtuvalt funktsionaalsetest nõuetest on võimalik kujundada andmebaasi ERD[1]. Olemi-suhte diagrammi joonistamiseks on kasutatud Lucidchart veebirakendust. Joonisel 4 on lõputöö lahenduse andmebaasi skeem. Kuna veebirakendusel puudub vajadus tegeleda autoriseerimisega ja kasutajatega, piisab väga lihtsast andmebaasi struktuurist. Küsimärgiga on märgitud andmebaasi väljad, mis võivad tühjad olla.



Joonis 4. Olemi-suhte diagramm

4.4 Analüüsi kokkuvõte

Analüüsi käigus kirjeldati kliendi soovid ning pandi kirja rakendusele vajalikud funktsionaalsed ja mittefunktsionaalsed nõuded.

Tehnoloogiate analüüsi käigus selgus, et Python on tõhus ja lihtne lahendus veebirakenduste ehitamiseks. Pythoni veebirammistike seast otsustus kõige sobivamaks valikuks Flask tänu oma lihtsusele ja kergele õpitavusele. Rakendus kasutab MVC mustrit mis tähendab, et puudub vajadus eraldi *frontend*[2] rakendusele.

Andmebaasi lahenduseks toetab loodud rakendus kahte süsteemi: MariaDB ja SQLite. Esimene nendest on kliendi nõue olemasoleva serveri jaoks. SQLite kasutamine annab võimaluse rakendust kasutada ilma seda serverisse paigaldamata, jooksutades kogu süsteemi oma masinas. Kuna rakendus on konfigureeritav nii andmebaasi kui ka

[1] *Entity Relationship Diagram*, olemi-suhte diagramm

[2] Kasutajaliides, graafiline osa rakendusest mida lõppkasutaja näeb

paigaldamise poolest, siis annab see kasutajale kolm erinevat rakenduse paigaldamise viisi.

Rakendust arendatakse kasutades Visual Studio Code arenduskeskkonda ning valmis rakendus publitseeritakse kasutades GitHub koodi- ja versioonihalduskeskkonda.

5 Veebirakenduse arendus

Veebirakenduse arendus on jaotatud nelja alampunkti: veebiteenuse lahendus, selle testimine, dokumenteerimine ja publitseerimine.

5.1 Veebiteenuse lahendus

Rakenduse arenduse alustamiseks on mõttekas algselt luua Pythoni virtuaalne keskkond. Virtuaalne keskkond on koopia Pythoni tõlgendajast, mis laseb raamistikke paigaldada ilma globaalse Pythoni tõlgendaja mõjutamiseta. Flaski veebiraamistiku paigaldamiseks saab kasutada Pythoni raamistike juhtimissüsteemi pip. Kasutades macOS või Linux süsteemide käsurida tuleb virtuaalkeskkonna loomiseks, selle aktiveerimiseks ning selle sisse Flaski paigaldamiseks sisestada Joonises 5 kujutatud käsud [47].

```
$ python3 -m venv venv
$ . venv/bin/activate
(venv) $ pip install flask
```

Joonis 5. Virtuaalkeskkonna loomine, aktiveerimine ja Flaski paigaldamine

Kõik Flaski kasutavad rakendused peavad looma Flaski rakenduse instantsi. Veebiserver saadab kõik klientidelt sissetulevad päringud sellele objektile töötlemiseks kasutades WSGI[1] protokolli. WSGI kirjeldab kuidas veebiserver veebirakendustega suhtleb ning kuidas saab neid rakendusi ühe päringu töötlemiseks kokku aheldada [48]. Rakenduse instants on Flask klassi objekt (Joonis 6) [26].

~/website/__init__.py
```
from flask import Flask

def create_app():
   app = Flask(__name__)
   return app
```

Joonis 6. Veebirakenduse instantsi loomine

Nüüd on võimalik hakata rakendusele lisama komponente kuni see vastab antud lõputöö lahenduse skoobile. Nagu analüüsi käigus otsustati, on veebirakenduse arendamiseks mõttekas kasutada MVC mustrit. Rakenduse failide jaotus ja struktuur on seletatud antud

[1] *Web Server Gateway Interface*, veebiserveri lüüsi liides

lõputöö punktis 4.3 Arhitektuur. Järgnevad alampunktid kirjeldavad iga mustri osa teostamist koos näidetega.

5.1.1 Kontroller

Klient saadab päringuid veebiserverisse, mis omakorda saadab need edasi Flaski rakenduse instantsile. Flaski rakenduse instants peab teadma, millist koodi käivitada igale URL'ile saadetud päringule. Seega on rakendusel kaardistatud URL'ide ja Pythoni funktsioonide vaheline suhe. Kõige lihtsam viis sellise suhte loomiseks on läbi rakenduse instantsi dekoraatori (Joonis 7) [49].

~/website/views/views.py
```
from .. import app

@app.route('/')
def home():
   return '<h1>Hello World!</h1>'
```

Joonis 7. Rakendusele marsruudi lisamine

Antud lõputöö rakendusele on vaja teha päris mitmed vaateid ning ühte faili kõike vaateid paigutada on halb tava. Selle jaoks on mõttekas kasutele võtta Flaski *Blueprint* objekt. *Blueprint* on funktsionaalsuste kogumik mis sisaldab kontrollereid, vaateid, jms [50]. Joonis 8 kujutab *Blueprint* objekti loomist.

~/website/views/views.py
```
from flask import Blueprint

views = Blueprint('views', __name__)

@views.route('/')
def home():
   return '<h1>Hello World!</h1>'
```

Joonis 8. *Blueprint* objekti loomine

Blueprint objektide kasutamiseks tuleb need ka rakenduse instantsi loomisel ära registreerida (Joonis 9).

```
~/website/__init__.py
from flask import Flask

def create_app():
   app = Flask(__name__)

   from .views.views import views
   app.register_blueprint(views, url_prefix="/")

   return app
```

Joonis 9. *Blueprint* objekti registreerimine

Blueprint'ide kasutamine teeb ilusamaks rakenduse arhitektuuri, parandab loetavust ja aitab koodi taaskasutamisega. Antud lõputöö rakenduse jaoks on mõttekas luua kolm *Blueprint* objekti: *components*, *projects* ja *views*. Esimesed kaks tegelevad komponentide ja projektide päringutega ning kolmas on üldiste päringute nagu koduvaate või aruande lehe jaoks.

5.1.2 Vaade

Eelnevas punktis loodud kodulehe kontroller tagastab ainult staatilist lehekülge. Dünaamiliste andmete esitamiseks on vaja kasutada Flaski sisseehitatud Jinja2 mallide teeki. Mall on fail, mis sisaldab vastuse teksti koos kohta täitvate muutujatega kuhu läheb dünaamiline osa [26]. Joonises 10 on näha kuidas kontroller tagastab malli kliendile.

```
~/website/views/views.py
from flask import Blueprint, render_template

views = Blueprint('views', __name__)

@views.route('/')
def home():
   return render_template('home.html')
```

Joonis 10. Veebirakenduses malli kasutamine

Vaikimisi otsib Flaski rakendus malle *templates* nimelisest kaustast.

Mõttekas on siinkohal luua ka baasmall. Baasmall sisaldab veebilehe osasid, mis on igal vaatel nähtavad nagu navigeerimisriba, jalus, jms. Baasmalli on mõttekas ka lisada igasugune rakenduse CSS[1] ja JavaScript. Antud rakendus kasutab CSS'i ja eesliidese raamistikku Bootstrap. Bootstrap on suur kogumik kasulikke ja taaskasutatavaid HTML,

[1] *Cascading Style Sheets*, stiililehe keel

CSS ja JavaScripti koodijuppe [51]. Bootstrap lisatakse baasfaili, et see oleks saadaval igal baasfaili laiendataval rakenduse vaatel. Järgnevalt on kujutatud antud lõputöö veebirakenduse baasmall ja seda laiendav koduleht (Joonis 11 ja 12).

~/website/templates/base.html

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <title>{% block title %}{% endblock %}</title>
      …
   </head>
   <body>
       <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
           …
        </nav>
       <main role="main" class="py-5 container">
          <div class="body-content">{% block content %}{% endblock %}</div>
       </main>
       <footer class="container">
          …
       </footer>
       …
   </body>
</html>
```

Joonis 11. Veebirakenduse baasmall

~/website/templates/home.html

```
{% extends "base.html" %}
{% block title %}
Home
{% endblock %}
{% block content %}
<h1>Home</h1>
…
{% endblock %}
```

Joonis 12. Veebirakenduse kodulehe mall

Lisaks dünaamiliste andmete esitamisele, aitavad mallid parandada koodi taaskasutamist ning loovad parema struktuuri veebirakendusele.

Dünaamiliste andmetega vaadete renderdamist käsitletakse järgnevas osas.

5.1.3 Andmebaas

Flask ei sea piiranguid andmebaasi pakettidele mida rakendus saab kasutada. Samuti on olemas mitmeid erinevaid andmebaasi abstraktsioonikihtide teeke, mis lasevad töötada kõrgema taseme Pythoni objektidega selle asemel, et käsitleda andmebaasi tabeleid,

veerge, jms. Sellist tarkvara tehnikat nimetatakse ORM[1] disainiks. Antud lõputöö rakendus kasutab Flask-SQLAlchemy andmebaasi käsitlemisteeki. SQLAlchemy on võimas relatsioonilise andmebaasi teek, mis toetab mitmeid andmebaasi süsteeme [26]. SQLAlchemy teegi paigaldamiseks tuleb virtuaalkeskkonna käsureal sisestada Joonises 13 kujutatud käsk.

```
(venv) $ pip install flask-sqlalchemy
```

Joonis 13. SQLAlchemy teegi paigaldamine virtuaalkeskkonda

Pärast teegi installeerimist virtuaalkeskkonda tuleb rakenduse instantsi loomise faili importida SQLAlchemy klass. Rakenduse konfiguratsioonil on vaja ka võti-väärtus paare salavõtme ja andmebaasi aadressi jaoks. Importida tuleb ka andmebaasi mudeli klassid mida SQLAlchemy teek rakenduse kontekstiga luua üritab [52]. Joonis 14 kujutab uuendatud rakenduse instantsi loomis faili.

~/website/__init__.py

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

def create_app():
   app = Flask(__name__)
   app.config['SECRET_KEY'] = 'secret'
   app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'

   from .views.views import views
   app.register_blueprint(views, url_prefix="/")


   from .database.models import Component
   with app.app_context():
       db.create_all()

   return app
```

Joonis 14. Andmebaasi ühenduse ja mudeli loomine

Andmebaasi mudeli defineerimiseks tuleb importida rakenduse instantsi loomisel tehtud SQLAlchemy objekt. Joonisel 15 on kujutatud andmebaasi mudeli kood.

[1] *Object-relational mapping*, objektide ja relatsioonilise andmete tõlkimine

~/website/database/models.py

```
from .. import db

class Component(db.Model):
   id = db.Column(db.Integer, primary_key=True)
   name = db.Column(db.String, nullable=False)
   location = db.Column(db.String)
   value = db.Column(db.String)
   amount = db.Column(db.Integer, default=0, nullable=False)
   minimum_amount = db.Column(db.Integer)
   url = db.Column(db.String)
```

Joonis 15. Komponendi mudel

Mudeli kasutamiseks tuleb rakenduse instantsi loomisel loodud SQLAlchemy klass ja ka kasutatav andmebaasi mudel importida vastavasse kontrollerisse. Kuna *Component* klass on *db.Model* alamklass, siis on olemas kõrgemast klassist päritud meetodid nagu päringud ja kirjete lisamine. Järgnevad koodinäited on aruande komponentide päring ning edastamine kontrolleris ja dünaamiliste andmete kuvamine aruande mallis (Joonis 16 ja 17).

~/website/views/views.py

```
from flask import Blueprint, render_template
from ..database.models import Component
from .. import db

views = Blueprint('views', __name__)

@views.route('/')
def home():
   return render_template('home.html')

@views.route('/report')
def report():
   query = Component.query.filter(
            Component.amount <= Component.minimum_amount)
   return render_template('report.html', components=query)
```

Joonis 16. Andmebaasist andmete pärimine ja nende kasutamine mallis

```
~/website/templates/report.html
{% extends “base.html“ %}
{% block title %}
Report
{% endblock %}
{% block content %}
<h1>Report</h1>
<table class=“…“>
   <thead>
      <tr>
         <th scope=“col“>Name</th>
         …
      </tr>
   </thead>
   <tbody>
      {% for component in components %}
      <tr>
         <td>{{component.name}}</td>
         …
      </tr>
      {% endfor %}
   </tbody>
</table>
{% endblock %}
```

Joonis 17. Dünaamiliste andmete esitamine mallis

Kõik eelmainitud tehnikad on kasutusel, et ehitada antud lõputöö skoobile vastav veebirakendus.

5.2 Testimine

Käesoleva lõputöö raames mõeldakse testimise all automaattestimist. Automaattestide kirjutamiseks ja jooksutamiseks kasutatakse pytest raamistikku. Pytest on Pythoni testimisraamistik mida saab kasutada erinevate tarkvaratestide organiseerimiseks ja jooksutamiseks. Pytest raamistiku kasutamiseks tuleb see paigaldada enda Pythoni virtuaalmasinasse (Joonis 18).

```
(venv) $ pip install pytest
```

Joonis 18. Testimisraamistiku pytest paigaldamine

Testide identifitseerimiseks on tavaliselt failid paigaldatud *tests* nimelisse kausta. Testid on funktsioonid, mis algavad nimega *test_** [53].

Pytestile omaseid kinnitusi või *fixture* saab kasutada, et kirjutada koodiosasid mis on taaskasutatavad testide vahel. Antud lõputöö rakendus loob rakenduse instantsi eraldi

funktsiooni välja kutsudes, seega on mõttekas luua kinnitus rakenduse ja test kliendi jaoks (Joonis 19).

~/tests/conftest.py

```
import pytest
from website import create_app

@pytest.fixture()
def app():
   app = create_app()
   app.config.update({
      “TESTING“: True,
   })
   yield app

@pytest.fixture()
def client(app):
   return app.test_client()
```

Joonis 19. Rakenduse ja test kliendi loomine

Test kliendil pole vaja reaalajas serverit, et teha päringuid rakendusele. Järgnevalt on Joonises 20 kujutatud kuidas kasutada test klienti, et kontrollida koduvaate sisu.

~/tests/test_views.py

```
def test_home(client):
   response = client.get(’/’)
   assert b“Navigation“ in response.data
```

Joonis 20. Rakenduse koduvaate sisu kontrollimine

MVC arhitektuuril on andmete edastamine ja kasutajaliides ühendatud, mis teeb testimise raskeks. Sellel põhjusel on kirjutatud antud veebirakendusele ainult põhilised CRUD[1] testid komponentide ja projektide jaoks ning lisaks kontrollitakse ka testidega lihtsamate vaadete sisusid.

5.3 Dokumenteerimine

Antud lõputöö lahenduse üheks eesmärgiks on luua arendatav veebirakendus. Selleks, et tulevikus oleks kasutajatel või arendajatel kergem rakendust arendada ning uuendada on rakenduse kõik funktsioonid ja klassid dokumenteeritud kasutades koodi kommentaare ning vajadusel ka funktsoonide tüübi vihjeid (Joonised 21 ja 22).

[1] *Create-read-update-delete*, neli põhilist funktsiooni andmete hoiustamiseks

~/website/views/views.py

```
from flask import Blueprint, render_template
from ..database.models import Component
from .. import db

# Creating Blueprint, URL prefix is '/'
views = Blueprint('views', __name__)

# Mapping home() function as handler for address '/'
# Function renders the home page using template from ./templates/home.html
@views.route('/')
def home():
   return render_template('home.html')

# Mapping report() function as handler for address '/report'
# Queries components where the amounts are smaller or equal to minimum amount
# Function renders the report page using query result and
# template from ./templates/home.html
@views.route('/report')
def report():
   query = Component.query.filter(
            Component.amount <= Component.minimum_amount)
   return render_template('report.html', components=query)
```

Joonis 21. Dokumenteeritud vaadete kontroller

~/website/helper.py

```
# A helper function used to check if file is allowed
# Uses the file name and a list of allowed extension i.e. ['json']
# Used when validating imported files
# Returns True if allowed, otherwise False
def allowedFile(fileName: str, allowed: list[str]) -> bool:
   return '.' in fileName and \
          fileName.rsplit('.', 1)[1].lower() in allowed
```

Joonis 22. Dokumenteeritud abifunktsioon

5.4 Publitseerimine

Antud lõputöö veebirakenduse kood on publitseeritud kasutades GitHub versioonihalduskeskkonda. Flaski sisseehitatud serverit ei ole soovitatud kasutada väljaspool arendust, selle asemel tuleks kasutada eraldi rakenduse serveri raamistikku [54]. Antud lahenduse jaoks on otsustatud kasutada Waitress serveri raamistikku, sest seda on lihtne ja kiire konfigureerida. Selleks, et kasutajal oleks rakenduse paigaldamine kiire ja mugav on selle jaoks kirjutatud kiirjuhend. Kuna rakenduse käivitamiseks on vaja päris mitu teeki paigaldada siis on kasulik luua vajalikke teekide fail ning see koos rakendusega publitseerida. Rakenduse versioonihalduse aadressi leiab Lisast 2.

6 Hinnang loodud veebirakendusele

Lõputöö raames loodud veebirakendust saab hinnata saavutatud funktsionaalsuste alusel. Samuti on võimalik hinnata tehnoloogiate valikut, mis on oluline tuleviku edasi arenduseks. Tähtis on ka kliendi hinnang, sest autoril endal puudub pikem kogemus elektroonikakomponentide ja -projektide majandamisega (Lisa 3).

6.1 Saavutatud kasutatavus

Loodud rakendus saavutas kõik kirjapandud funktsionaalsed ja mittefunktsionaalsed nõuded. Rakenduse arendamine oli lõputöö autorile suureks üleskutseks, sest enamuste tehnoloogiatega polnud varem kokku puututud. Töö kirjutamise käigus omandati lisaks ka teadmisi erinevate serverilahenduste ja struktuuride kohta.

Rakendus on olemas ühe tervikuna ning kuulub ühte ainsasse avalikku GitHubi repositooriumisse. Seal on ka kirjas juhend rakenduse paigaldamiseks ning võimalus esitada probleemide või vigade teavitusi. Hetkel on rakendus konfigureeritud kasutama kas arenduseks sobilikku sisseehitatud serveri või Waitress raamistiku serverit. Rakenduse paigaldamiseks kohalikku serverisse tuleb selleks installeerida rakenduse kood, konfigureerida andmebaasiühendus ja sätted, paigaldada vajalikud teegid ja siis käivitada veebirakendus.

6.2 Kasutatavad tehnoloogiad

Rakenduse loomiseks kasutati Pythoni kõige uuemat stabiilset versiooni 3.11 ning kõige uuemat Flask raamistiku versiooni 2.2. Andmebaasisüsteemi jaoks oli nõue kasutada MariaDB süsteemi, kuid lahenduse skoopi mahtus ka SQLite paigaldamine, et rakendus oleks kasutatav ka kohalikus masinas.

Rakenduse testimise jaoks kasutati pytest raamistikku. WSGI serveri käivitamiseks saab kasutada Flaski sisseehitatud serveri või Waitress raamistiku serveri mis on töökindlam ja parema jõudlusega.

Tehnoloogiate valimisel peeti peamiselt silmas kasutatava tehnoloogia dokumentatsiooni ning aktiivest kasutajaskonda. Tänu sellele oli võimalik autoril, kes varem ei ole Pythoni

ega selle raamistikega veebirakendusi kirjutanud, teha valmis nõuetele vastav, kiire ja tõhus lahendus laosüsteemi probleemile. Kasutatavad tehnoloogiad on veel arenduses ning saavad uuendusi aastatega juurde.

6.3 Võimalused edasiarenduseks

Rakenduse edasiarenduse võimalusi on palju. Olemasolevate lahenduste analüüsi käigus selgus palju funktsionaalsusi mille peale probleemi püstitamise ajal ei tuldud. Kuigi nende funktsionaalsuste lisamine jäi antud lõputöö skoobist välja, on võimalus igaühel rakendust edasi arendada.

Üks olulisemaid edasiarenguid oleks kindlasti komponentide nimekirja importimine ja toetamine erinevatest olemasolevatest lahendustest. Praegusel versioonil on küll võimalus komponente JSON formaadis failist hulgi importida aga mitmetel teistel lahendustel on võimalus komponente otse ühest rakendusest teise tuua. Selline toetus oleks suureks abiks nendele kellel juba mingisugune laosüsteem on kasutusel, kuid otsivad endale midagi uut ja paremat. See teeks antud lõputöö rakenduse kasutusele võtmise ahvatlemaks ning rohkematelt kasutajatelt oleks võimalik saada rohkem tagasisidet.

Hetkel toetab rakendus ainult inglise keelt. Kindlasti oleks kasutajaskonda võimalik suurendada erinevate keelte toetamisega, kuid selliste arengute jaoks puudub hetkel arendatud rakendusele toetus. Selle jaoks peaks mõtlema ka andmebaasi lahenduse muutmise peale, sest vaja oleks kasutajapõhiseid erisusi.

Võimalik oleks ka rakendusele ehitada eraldiseisev klientrakendus. Kuna Jinja2 ei ole väga populaarne ja palju kasutatav, siis on sellel palju puudujääke. Kui ehitada rakendus ümber MVC mustri lahendusest, oleks võimalik kasutajaliides luua näiteks React *frontend* raamistikuga, mis on palju populaarsem ja paremate funktsionaalsustega. See annaks ka võimaluse luua ka mobiilirakenduse, kuid ka sellega kaasnevad omad probleemid, sest hetkel on rakendus suunatud suure ekraaniga seadmetel kohalikus võrgus või masinas kasutamiseks, sellel puudavad kasutajate manageerimine ja autentimise võimalused.

Antud lõputöö laosüsteemi veebirakendus on avatud lähtekoodiga mis tähendab, et selle täiendamisega võivad tegeleda kõik kes soovivad.

7 Kokkuvõte

Antud bakalaureusetöö käigus loodi lahendus väikesemahulise lao haldamise probleemile. Rakenduse skoopi kuulub: komponendid haldus ja sellega seonduvad funktsioonid, projektide haldus ja sellega seonduv ning aruande koostamine. Komponente on võimalik lisada, muuta, uuendada, kustutada ja hulgi importida. Projekte on võimalik luua, muuta, kustutada, projekti saab lisada olemasolevaid komponente ning projekte saab ehitada. Koostatud aruandes on näha milliseid komponendid hakkavad laost otsa saama. Rakendus on mõeldud enda masinas käivitamiseks või kohalikus võrgus olevasse serverisse paigaldamiseks.

Loodud rakendusega on saavutatud töö alguses püstitatud eesmärk. Rakendus on vabavaraline ning saadaval kõigile. Loodud lahendus on suureks abiks väikesemahulise elektroonikakomponentide lao jälgimiseks ning organiseerimiseks. Rakendus aitab kasutajal oma komponente leida ning kiiresti tuvastada mida on vaja juurde tellida. See hoiab kokku raha ja aega, mis on tänapäeval väga tähtis. Rakendus on loodud silmas pidades edasiarendamise võimalikkust, seda toetavad põhjalikult dokumenteeritud kood ja automaattestid. Klient on loodud lahendusega rahul ning rakendus on kasutusele võetud.

Kasutatud kirjandus

[1] Mouser Electronics, Inc., „About Mouser Electronics - Electronics Component Distributor,“ Mouser Electronics, Inc., veebruar 2023. [Võrgumaterjal]. Available: https://www.mouser.ee/aboutus/. [Kasutatud 26 veebruar 2023].

[2] Mouser Electronics, Inc., „Inventory Management by Mouser,“ Mouser Electronics, Inc., veebruar 2023. [Võrgumaterjal]. Available: https://www.mouser.ee/online-inventory-help/. [Kasutatud 26 veebruar 2023].

[3] Digi-Key Electronics, „About DigiKey,“ Digi-Key Electronics, veebruar 2023. [Võrgumaterjal]. Available: https://www.digikey.ee/en/resources/about-digikey. [Kasutatud 26 veebruar 2023].

[4] Digi-Key Electronics, „myLists - Parts List Management,“ Digi-Key Electronics, veebruar 2023. [Võrgumaterjal]. Available: https://www.digikey.ee/en/mylists/. [Kasutatud 26 veebruar 2023].

[5] M. Brown, „GitHub · Binner Wiki,“ Refactor Software, veebruar 2023. [Võrgumaterjal]. Available: https://github.com/replaysMike/Binner/wiki. [Kasutatud 26 veebruar 2023].

[6] BOMIST, „Documentation,“ BOMIST, veebruar 2023. [Võrgumaterjal]. Available: https://docs.bomist.com/. [Kasutatud 26 veebruar 2023].

[7] J. Rychter, „PartsBox User's Guide (PartsBox),“ PartsBox, veebruar 2023. [Võrgumaterjal]. Available: https://partsbox.com/users-guide.html. [Kasutatud 26 veebruar 2023].

[8] R. Belfiore, „The Most Cutting-Edge Web Development Technologies | Blog,“ BairesDev, märts 2023. [Võrgumaterjal]. Available: https://www.bairesdev.com/blog/cutting-edge-web-development-technologies/. [Kasutatud 10 märts 2023].

[9] L. d. Alba, „A Guide to the Best Programming Language for Web Development 2023,“ SitePoint Pty. Ltd., november 2022. [Võrgumaterjal]. Available: https://www.sitepoint.com/best-programming-language-for-web-development/. [Kasutatud 10 märts 2023].

[10] P. McGee, C#, New York: McGraw-Hill Education, 2014.

[11] A. Ozanich, „What Is Java: The Beginner's Guide to the Java Programming Language,“ HubSpot, Inc., oktoober 2022. [Võrgumaterjal]. Available: https://blog.hubspot.com/website/what-is-java. [Kasutatud 12 märts 2023].

[12] R. Toal, „What is Java and What is it Used For?,“ Code Institute Global, märts 2023. [Võrgumaterjal]. Available: https://codeinstitute.net/global/blog/what-is-java/. [Kasutatud 12 märts 2023].

[13] A. Jordana, „What is JavaScript? A Basic Introduction to JS for Beginners,“ Hostinger, jaanuar 2023. [Võrgumaterjal]. Available: https://www.hostinger.com/tutorials/what-is-javascript. [Kasutatud 12 märts 2023].

[14] PHP Group, „PHP: What is PHP? - Manual,“ PHP Group, märts 2023. [Võrgumaterjal]. Available: https://www.php.net/manual/en/intro-whatis.php. [Kasutatud 12 märts 2023].

[15] K. Ashwani, „What is PHP? and How PHP works?,“ DevOpsSchool, mai 2021. [Võrgumaterjal]. Available: https://www.devopsschool.com/blog/what-is-php-and-how-php-works/. [Kasutatud 12 märts 2023].

[16] Coursera, „What Is Python Used For? A Beginner's Guide,“ Coursera, aprill 2023. [Võrgumaterjal]. Available: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python. [Kasutatud 12 märts 2023].

[17] M. Aina, „Is C# Hard to Learn?,“ Career Karma, jaanuar 2022. [Võrgumaterjal]. Available: https://careerkarma.com/blog/is-c-sharp-hard-to-learn/. [Kasutatud 12 märts 2023].

[18] T. Statler, „Is Java Hard To Learn For A Beginner? The Hard Facts,“ Compi Sci Central, märts 2023. [Võrgumaterjal]. Available: https://compscicentral.com/is-java-hard-to-learn/. [Kasutatud 12 märts 2023].

[19] S. Morris, „How Hard Is it to Learn JavaScript? The Pros Weigh In - Skullcrush,“ Skillcrush, märts 2023. [Võrgumaterjal]. Available: https://skillcrush.com/blog/how-hard-is-it-to-learn-javascript/. [Kasutatud 12 märts 2023].

[20] T. V. Harz, „How long does it take to learn PHP?,“ TylerTheTech, veebruar 2023. [Võrgumaterjal]. Available: https://tylerthetech.com/how-long-does-it-take-to-learn-php/. [Kasutatud 12 märts 2023].

[21] Thinkful, „How Hard is it to Learn Python?,“ Thinkful, märts 2023. [Võrgumaterjal]. Available: https://www.thinkful.com/blog/how-hard-is-it-to-learn-python/. [Kasutatud 12 märts 2023].

[22] J. Korsun, „The 16 Most Important Pros and Cons of using Python for Web Development,“ Django Stars, veebruar 2023. [Võrgumaterjal]. Available: https://djangostars.com/blog/python-web-development/. [Kasutatud 16 märts 2023].

[23] T. Bąk, „30+ Companies Using Python by Domain - Python Use Cases,“ SoftKraft, märts 2023. [Võrgumaterjal]. Available: https://www.softkraft.co/companies-using-python/. [Kasutatud 16 märts 2023].

[24] R. Gour, „Python Web Framework - A Detailed List of Web Frameworks in Python,“ Towards Data Science, detsember 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/python-web-framework-a-detailed-list-of-web-frameworks-in-python-1916d3c6222d. [Kasutatud 16 märts 2023].

[25] D. Ó. Tuama, „What is Django? An Introduction,“ Code Institute Global, märts 2023. [Võrgumaterjal]. Available: https://codeinstitute.net/global/blog/what-is-django/. [Kasutatud 2023 16 märts].

[26] M. Grinberg, Flask Web Development, Sebastopol: O'Reilly Media, Incorporated, 2018.

[27] Pylons Project, „The Pyramid Web Framework - The Pyramid Web Framework v2.0.1,“ Pylons Project, jaanuar 2023. [Võrgumaterjal]. Available: https://docs.pylonsproject.org/projects/pyramid/en/latest/index.html. [Kasutatud 23 märts 2023].

[28] M. Goyal, „Introduction to Bottle Framework,“ Coding Ninjas, august 2022. [Võrgumaterjal]. Available:

https://www.codingninjas.com/codestudio/library/introduction-to-bottle-framework. [Kasutatud 23 märts 2023].

[29] AppOptics, „What Is CherryPy - Introduction to CherryPy,“ SolarWinds, märts 2023. [Võrgumaterjal]. Available: https://www.appoptics.com/glossary/cherrypy. [Kasutatud 23 märts 2023].

[30] V. S. Khatri, „Flask vs Django: Which Python Web Framework to Use in 2023?,“ Hackr.io, märts 2023. [Võrgumaterjal]. Available: https://hackr.io/blog/flask-vs-django. [Kasutatud 23 märts 2023].

[31] A. Ronacher, „Templates - Flask Documentation (2.2.x),“ Flask, märts 2023. [Võrgumaterjal]. Available: https://flask.palletsprojects.com/en/2.2.x/tutorial/templates/. [Kasutatud 23 märts 2023].

[32] J. Ellingwood, „The Different Types of Databases - Overview with Examples,“ Prisma, märts 2023. [Võrgumaterjal]. Available: https://www.prisma.io/dataguide/intro/comparing-database-types. [Kasutatud 23 märts 2023].

[33] R. Payne, „What is MySQL?,“ DatabaseJournal, detsember 2022. [Võrgumaterjal]. Available: https://www.databasejournal.com/mysql/what-is-mysql/. [Kasutatud 23 märts 2023].

[34] A. Shahzeb, „MariaDB vs MySQL: Key Differeneces & All You Need To Know,“ Cloudways, november 2022. [Võrgumaterjal]. Available: https://www.cloudways.com/blog/mariadb-vs-mysql/. [Kasutatud 23 märts 2023].

[35] Kinsta, „What Is PostgreSQL?,“ Kinsta, veebruar 2023. [Võrgumaterjal]. Available: https://kinsta.com/knowledgebase/what-is-postgresql/. [Kasutatud 23 märts 2023].

[36] P. Pedamkar, „What is Oracle? | Features, Importance and Benefits of Oracle,“ EDUCBA, märts 2023. [Võrgumaterjal]. Available: https://www.educba.com/what-is-oracle/. [Kasutatud 23 märts 2023].

[37] F. Filipsson, „Six Oracle Database Licensing Models and Costs - 2023,“ Redress Compliance, märts 2023. [Võrgumaterjal]. Available: https://redresscompliance.com/common-oracle-database-licensing-models/. [Kasutatud 23 märts 2023].

[38] A. S. Ravikiran, „What is SQLite? And When to Use It?,“ Simplilearn, veebruar 2023. [Võrgumaterjal]. Available: https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sqlite. [Kasutatud 23 märts 2023].

[39] Codecademy, „What is GitHub? | Codecademy,“ Codecademy, märts 2023. [Võrgumaterjal]. Available: https://www.codecademy.com/article/what-is-github.

[40] Atlassian, „Bitbucket: What is Bitbucket? | Evaluator Resources | Atlassian Documentation,“ Atlassian, märts 2018. [Võrgumaterjal]. Available: https://confluence.atlassian.com/confeval/development-tools-evaluator-resources/bitbucket/bitbucket-what-is-bitbucket. [Kasutatud 2 aprill 2023].

[41] A. T. Tunggal, „Bitbucket vs GitHub (Updated for 2023),“ UpGuard, jaanuar 2023. [Võrgumaterjal]. Available: https://www.upguard.com/blog/bitbucket-vs-github. [Kasutatud 2 aprill 2023].

[42] Incredibuild Software Ltd., „What is GitLab?,“ Incredibuild Software Ltd., märts 2023. [Võrgumaterjal]. Available: https://www.incredibuild.com/integrations/gitlab. [Kasutatud 2 aprill 2023].

[43] Atatus, „PyCharm vs. VS Code: A Detailed Comparison for Choosing the Best Python IDE,“ NamLabs Technologies Pvt Ltd., november 2022. [Võrgumaterjal]. Available: https://www.atatus.com/blog/pycharm-vs-vscode/. [Kasutatud 2 aprill 2023].

[44] M. Heller, „What is Vusial Studio Code? Microsoft's extensible code editor,“ InfoWorld, juuli 2022. [Võrgumaterjal]. Available: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html. [Kasutatud 2 aprill 2023].

[45] L. Alegsa, „Klient-server,“ Alegsaonline, juuni 2021. [Võrgumaterjal]. Available: https://et.alegsaonline.com/art/20972. [Kasutatud 3 aprill 2023].

[46] Z. Svirca, „Everything you need to know about MVC architecture,“ Towards Data Science, mai 2020. [Võrgumaterjal]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1. [Kasutatud 3 aprill 2023].

[47] A. Ronacher, „Installation - Flask Documentation (2.2.x),“ Flask, märts 2023. [Võrgumaterjal]. Available: Ronacher, Armin. [Kasutatud 9 aprill 2023].

[48] Analytics Vidhya, „What is WSGI (Web Server Gateway Interface)?,“ Medium, september 2020. [Võrgumaterjal]. Available: https://medium.com/analytics-vidhya/what-is-wsgi-web-server-gateway-interface-ed2d290449e. [Kasutatud 9 aprill 2023].

[49] T. Birchard, „The Art of Routing in Flask,“ Hackers and Slackers, september 2018. [Võrgumaterjal]. Available: https://hackersandslackers.com/flask-routes/.

[50] M. Garcia, „Use a Flask Blueprint to Architect Your Applications,“ Real Python, veebruar 2020. [Võrgumaterjal]. Available: https://realpython.com/flask-blueprint/. [Kasutatud 9 aprill 2023].

[51] A. Ouellette, „What is Bootstrap? An Awesome 2023 Beginner's Guide,“ CareerFoundry, veebruar 2023. [Võrgumaterjal]. Available: https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/. [Kasutatud 9 aprill 2023].

[52] A. Ronacher, „Quick Start - Flask-SQLAlchemy Documentation (3.0.x),“ Flask, aprill 2023. [Võrgumaterjal]. Available: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/quickstart/. [Kasutatud 9 aprill 2023].

[53] A. Ronacher, „Testing Flask Applications - Flask Documentation (2.2.x),“ Flask, 2023 aprill. [Võrgumaterjal]. Available: https://flask.palletsprojects.com/en/2.2.x/testing/. [Kasutatud 11 aprill 2023].

[54] A. Ronacher, „Deploying to Production - Flask Documentation (2.2.x),“ Flask, aprill 2023. [Võrgumaterjal]. Available: https://flask.palletsprojects.com/en/2.2.x/deploying/. [Kasutatud 12 aprill 2023].

[55] W. Chai, „TechTarget - What is SaaS (Software as a Service)? Everything You Need to Know,“ TechTarget, Inc., oktoober 2022. [Võrgumaterjal]. Available: https://www.techtarget.com/searchcloudcomputing/definition/Software-as-a-Service. [Kasutatud 23 veebruar 2023].

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

Mina, Joonas Kaal

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Veebipõhise väikesemahulise elektroonikakomponentide lao kasutajarakendus", mille juhendaja on Andres Kütt
   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 – Rakenduse versioonihaldus

https://github.com/jokaal/electronic-component-storage

Lisa 3 – Kliendi tagasiside veebirakendusele

Paigaldasin laosüsteemi paigaldusjuhist üldjoontes järgides olemasolevale Linux serverile kuhu oli juba paigaldatud MariaDB ja Python3 (versioon 3.9.2), tegin laosüsteemi jaoks uue kasutaja. Paigaldamiseks lisasin MariaDB-sse andmebaasi ja kasutaja ning andsin vastloodud kasutajale andmebaasi kasutamise õigused. Seejärel lisasin kasutaja ja andmebaasi andmed konfiguratsioonifaili. Pythonit ei hakanud kohe uuendama, paigaldasin vaid puuduolevad pythoni teegid loodud uue kasutaja jaoks, et mitte süsteemselt paigaldatud teeke muuta. Paigaldasin kasutaja jaoks ka pytest-i, see aitas kiiresti konfiguratsiooni õigsust ja süsteemi elementaarset töövõimet kontrollida.

Laosüsteemi automaatseks käivitamiseks tegin Linuxi systemd konfiguratsioonifaili ja väikese shelliscripti laosüsteemi käivitamiseks waitress serveriga, vastloodud kasutaja õigustes. Andmebaasi varukoopiate tegemise organiseerisin cron-iga, kord ööpäevas tehakse mysqldump-iga andmebaasi sisust tekstikoopia, kuupäev sisaldava nimega faili.

Pärast süsteemi paigaldamist läks kõik valutult, kohe sai hakata komponente sisestama ja neid projektides kasutama. Projektide loomise juures tuli kasutamise juures välja paar pisikest tööprotsessi parendamise võimalust mis said ka lõplikku versiooni sisse viidud.

Pikema kasutusel oleku aja jooksul ilmselt tekib veel mõtteid kuidas asja paremaks teha aga põhifunktsionaalsuse osas ei saa juba praegu millegi üle nuriseda. Eriti tahaks välja tuua selle, et kõik funktsioonid töötavad väga nobedalt ja komponentide leidmine laost on lihtne ning töökindel. Väga tubli tulemus on ka see, et juba esimesel paigaldusel ei tulnud välja ühtegi tõsist süsteemiviga kus näiteks andmebaasis oleks midagi valesti läinud või mingi funktsionaalsuse kasutamisel oleks mingi veateada tulnud.

Komponentide lao põhjalik inventuur seisab alles ees aga olen juba praegu kindel, et valminud süsteem jääb kasutusele ja on suureks abiks.