TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Daniel Rasmus Põder 193583IABB

Automaatne arvete koostamise ja väljasaatmise rakendus

Bakalaureusetöö

Juhendaja: Liisa Jõgiste
MSc

Tallinn 2024

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: Daniel Rasmus Põder

03.01.2024

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on luua rakendus, mis koostab ja saadab automaatselt välja arveid. Loodava rakendusega on soov lahendada ebaefektiivse ajakasutuse probleem ja automatiseerida manuaalne töölõik.

Töö olulisemad osad on:

- Integratsioon Hubstaff'iga, et rakendus saaks kokku koguda valitud ajaperioodi jooksul tehtud töötunnid
- Rakenduse sisemine loogika, mis redigeerib ajaperioodi jooksul tehtud töötunde vastavalt ettevõttesisestele reeglitele
- Integratsioon Meriti raamatupidamistarkvaraga, mille abil luuakse Meriti portaalis kliendile väljastatav arve

Bakalaureusetöö on kirjutatud eesti keeles ning sisaldab teksti 60 leheküljel, 7 peatükki, 33 joonist, 8 tabelit.

Abstract

Automated invoice creation and sending application.

The goal of this thesis is to create an application that will automatically create and send out invoice. The solution aims to reduce project managers work by creating a comfortable way to create and send out invoice.

The main parts of the thesis:

- Integration with Hubstaff to collect the time periods development hours.
- Applications internal logic which would modify the development hours based on a set of company rules.
- Integration with accounting platform Merit to send out the created invoices.

The thesis is in Estonian and contains 60 pages of text, 7 chapters, 33 figures, 8 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface,* API on määratletud reeglite kogum, mis võimaldab erinevatel rakendustel omavahel suhelda. [1]
DevSecOps	*development, security, and operations*, arendus, turvalisus ja operatsioonid. [2]
HTML	*Hyper Text Markup Language,* HTML on veebilehtede loomise standardne märgistuskeel. [3]
HTTP	*Hyper Text Transfer Protocol*, Hüperteksti edastusprotokoll. [4]
Hubstaff	Hubstaff on tootlikkuse ja tööjõu haldamise tarkvara, mis võimaldab jälgida kui palju on töötajad valitud perioodi jooksul tööd teinud. [5]
IDE	*Integrated Development Environment*, Integreeritud arenduskeskkond. [6]
JavaScript	JavaScript on veebirakenduste programmeerimiskeel, mis võimaldab muuta HTML ja CSS. [7]
JSON	*JavaScript Object Notation,* JSON on vorming andmete edastamiseks ja salvestamiseks. [8]
Laracast	Õppekeskkond, kus on õppevideod Laraveli ja teiste raamistike kohta. [9]
Laravel	Laravel on veebirakenduse raamistik, mis põhineb PHP arenduskeelel. [10]
OAuth 2.0	„Open Authorization“, standard, mis on loodud selleks, et kasutajatel oleks autentimise tulemusena võimalik ligi pääseda kolmandate osapoolte veebirakenduste ressurssidele. [11]
PHP	*PHP: Hypertext Preprocessor*, PHP on üldotstarbeline skriptikeel, mis sobib hästi veebirakenduste arenduseks. [12]
REST	*Representational State Transfer*, REST on disainiprintsiip, mida saab kasutada API’de loomisel. [13]
SPA	*Single-page application,* SPA on veebirakendus, mis laeb ainult ühe lehe ja uuendab selle sisu JavaScripti kaudu. [14]
TypeScript	TypeScript on JavaScripti süntaktiline superkomplekt. [15]
WSL	*Windows Subsystem for Linux*, WSL on Windowsi lahendus, mis lubab jooksutada Linux’it Windowsi masinate peal. [16]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Ettevõtetes kulutatakse igakuiselt tunde arvete koostamisele. Kuigi arvete koostamiseks on olemas mitmeid tarkvarasid, siis paljudel juhtudel tehakse neid ikkagi käsitsi, sest pakutavad tarkvarad on tasulised ja/või ei võta arvesse ettevõtte eripärasid. Autor ise on projektijuht ja tal kulub ühe arve koostamiseks umbes 45-75 minutit. Igakuiselt peab ta koostama 2 arvet. Autori ettevõttes koostatakse igakuiselt umbes 10 arvet. Arvutuskäigu tulemusena selgub, et igakuiselt kulub kokku 7,5-12,5 tundi arvete koostamisele.

Sellest tulenevalt leidis autor, et mõttekas on arvete koostamiseks ja väljastamiseks luua ise ettevõtte vajadustel põhinev rakendus. Sel viisil on võimalik keskenduda ainult vajalikele funktsionaalsustele, samuti on rakenduse töökäik ettevõtte töötajatele teada. Selle seisukohaga nõustus ka ettevõtte tegevjuht, kus autor töötab. Mõlemad leidsid, et ettevõttele oleks kasulikum kui arvete koostamisele kulutatavat aega kasutatakse teiste, kõrgema lisandväärtusega tööülesannete täitmiseks, näiteks: tulemusraportite koostamine, projektihaldussüsteemi parendamine ja tulevaste arendustööde analüüsimine. Sellest tulenevalt tellis ettevõtte tegevjuht autorilt vastava rakenduse arenduse, sest ettevõtte vaatepunktist on kasulik arvete koostamise ja väljastamise protsess automatiseerida.

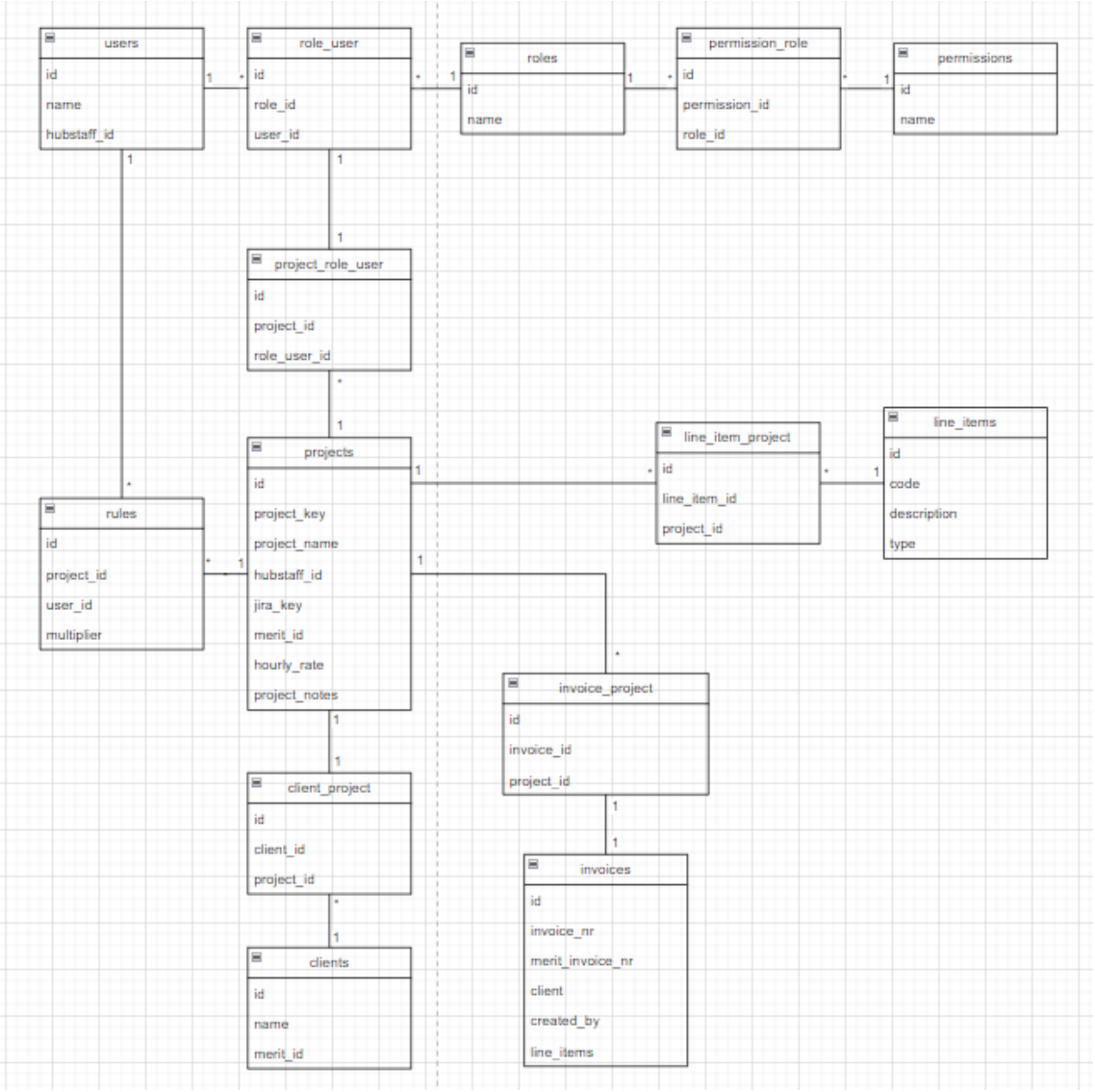
Antud töö eesmärgiks on luua rakendus, mida saavad kasutada projektijuhid, et mugavalt koostada ja väljastada arveid. Esmalt peab rakendus integratsiooni abil ajajälgimistarkvarast kokku koguma kindla projekti raames määratud perioodi jooksul tehtud arendustunnid. Seejärel neid vastavalt ettevõttesisestele reeglitele redigeerima, mille järel peab rakendus läbi integratsiooni edastama koostatud arve raamatupidamistarkvarasse, kust see saadetakse kliendile.

Järgnevalt toob autor välja rakenduse loomise protsessi, kasutatud tööriistad, rakenduse dokumentatsiooni ja analüüsi.

2 Metoodika

2.1 Protsess

Esmalt koostas töö autor rakendusele funktsionaalsed ja mittefunktsionaalsed nõuded. Nõuete koostamiseks viis autor läbi arutelu ettevõttes töötavate projektijuhtide ja tegevjuhiga. Arutelu käigus võeti arvesse varasemat kogemust ja probleeme protsessides. Järgnevalt tuli autoril koostada rakenduse spetsifikatsioon ja valida kõige sobilikumad tehnoloogiad rakenduse loomiseks. Selle tegemiseks andsid autorile sisendit ettevõtte arendajad ja tegevjuht, mille tulemusel koostati spetsifikatsioon ja valiti tehnoloogiad, mis vastavad paika pandud nõuetele. Seejärel tegi töö autor esimese andmebaasi mudeli ja koostas tehnilise dokumentatsiooni. Peale dokumentatsiooni koostamist tegi autor projektihaldustarkvaras JIRA valmis tööülesanded, mida hakkas kahenädalaste iteratsioonide käigus arendama. Kokku koostas autor 27 tööülesannet, millest 25 sai lõputöö käigus tehtud. Kui konkreetse ülesande raames vajalik arendus oli autori poolt valmis tehtud, siis testis ta loodud lahendust. Kui autor leidis, et tööülesanne on valmis, siis edastas ta selle ettevõttesisesele juhendajale, kes vaatas kirjutatud koodi üle ja testis lahendust. Kui lahendus oli piisavalt hea, siis juhendaja sulges tööülesande. Kui mingi tööülesanne vajas korrigeerimist, siis sai autor juhendajalt vastava tagasiside, mida ja miks on vaja muuta.



Joonis 1. Algne andmebaasimudel

2.2 Kasutatud tööriistad

Rakenduse loomiseks on mõistlik kasutada erinevaid tööriistasid ja tarkvarasid. Esiteks pidi autor välja valima projektihalduse tööriista, millega saab luua tööülesandeid, jälgida tööprotsessi, koostada dokumentatsioone ja talletada kõike rakendusega seonduv. Teiseks valis autor versioonihalduse tarkvara, kus koodibaasi talletada.

2.2.1 Projektihaldus

Autor leidis, et lõputöö tegemiseks on mõistlik kasutada projektijuhtimise tööriista, et planeerida vajalikud arendustööd läbi tööülesannete loomise. Lisaks on projektijuhtimise tööriistas hea jälgida arendustöö progressi ja näha tööülesannete raames tehtud koodi osi.

Veel oli autori arvates vajalik virtuaalne tööruum, kuhu talletada rakendusega seotud dokumente nagu tehniline dokumentatsioon, nõuded rakendusele ja spetsifikatsioon. Hea, kui rakendusega seotud dokumendid on lihtsasti jagatavad ja teiste poolt muudetavad.

Mõned tuntumad projektijuhtimise tööriistad on:

- ClickUp - tegemist on projektihaldussüsteemiga, mis on sobilik erineva suurusega ettevõtetele ja meeskondadele. Selle abil saab parandada meeskonna koostööd, säästa aega ja rohkem saavutada. [17]

- GitHub Issues - aitab planeerida arendustöid, jälgida töövoogu ja luua tööülesandeid. Seda on mugav kasutada, sest see paikneb otse versioonihalduse tarkvara GitHub keskmes. [18]

- JIRA - on juhtiv agiilne projektihalduse tööriist. Seda tööriista kasutavad tiimid planeerimiseks, jälgimiseks ja arendustöö toetamiseks. JIRA pakub mugavat integratsiooni teiste Atlassian'i toodetega. [19]

Lõputöö tegemiseks valis töö autor JIRA projektihaldussüsteemi, sest see on varasemalt olnud ettevõttes kasutusel ja ta on sellega juba tuttav. JIRA't on väga lihtne integreerida teiste Atlassian'i toodetega, mis võimaldab tööülesandeid, dokumentatsiooni ja diagramme lihtsalt ühendada.

Dokumentatsiooni halduseks kasutas töö autor Confluence'i, mis on samuti Atlassian'i toode ja tulenevalt lihtsast integratsioonist JIRA'ga tundus see autorile ainus sobiv valik.

2.2.2 Versioonihaldus

Lihtsa kättesaadavuse ja ligipääsetavuse eesmärgil leidis autor, et koodi tuleb hoiustada versioonihalduse keskkonnas.

Kolm populaarsemat versioonihaldustarkvara on:

- GitHub - 2008. aastal loodud vabavaraline rakendus, mis aitab kasutajatel oma koodibaasi talletada Git majutusteenuse abil. GitHub on üks populaarseim versioonihalduse tarkvara. [20]

- GitLab - 2014. aastal alguse saanud DevSecOps platvorm, mis pakub versioonihalduse, planeerimise ja projektihalduse lahendusi. GitLab on viimaste aastate jooksul populaarsust kogunud ja neil on juba üle 30 miljoni kasutaja. [21]

- Bitbucket - Atlassian'i poolt pakutav versioonihaldustarkvara, mis baseerub Git majutusteenusel. Bitbucket pakub kiiret ja mugavat integratsiooni Atlassian'i teiste toodetega nagu näiteks JIRA ja Confluence. [22]

Ettevõttes, kus autor töötab, on kasutusel mitmed erinevad Atlassian'i tooted ja integratsioon nende vahel on väga mugav ning loob lisandväärtust. Sellest tulenevalt valis töö autor koodibaasi talletamiseks versioonihaldustarkvara Bitbucket.

2.2.3 Muud tööriistad

Autor leidis, et arendustöö käigus on mõistlik lisaks eelpool valitud tööriistadele kasutada veel järgnevaid tööriistu, mis lihtsustavad rakenduse loomist:

- Postman - platvorm, mis võimaldab lihtsalt luua, testida ja kasutada API päringuid. Neid on platvormil võimalik hõlpsasti teiste arendajatega jagada ja samuti töötab see API dokumentatsioonina. [23]

- Ubuntu - käsurida või konsool, mis võimaldab läbi tekstiliidese suhelda arvutiga. Ubuntu põhineb Linuxi operatsioonisüsteemil, millest tulenevalt on Windowsi peal kasutamiseks vajalik WSL. [24]

- Confluence – Atlassian'i poolt pakutav virtuaalne meeskonnatööruum, kus on võimalik arendada ideid, luua projekti dokumentatsiooni ja organiseerida tegevusi. [25]

- PHPStorm - PHP jaoks loodud IDE, mis toetab PHP arenduskeeles programmeerimist enam kui konkureerivad IDE'd. [26]

- Slack - sõnumside rakendus, mis on peamiselt mõeldud ettevõtetele. Läbi Slack'i on võimalik töötada ühtse meeskonnana ja luua ettevõtete vahelist koostööd. [27]

3 Rakenduse kirjeldus

Antud peatükis räägib autor rakenduse nõuetest, kasutajatest, kasutusjuhtudest ja andmebaasist.

3.1 Kasutajad

Rakendusel on neli erinevat kasutaja rolli. Nendest kolm on otseselt seotud projektidega. Neljandaks kasutaja rolliks on administraator, kellel peab olema ligipääs kogu rakendusele ja selle sisule.

Need neli kasutaja rolli on:

- Administraator - rakenduse administraator. Kõige rohkemate õigustega kasutaja, kes tegeleb keskkonna haldusega ja peab ligi pääsema tervele rakenduse sisule.
- Mänedžer - projekti mänedžer. Projekti tasemel kõige rohkemate õigustega kasutaja. Ta näeb kogu projektiga seotud informatsiooni. Peamine rakenduse kasutus on arvete koostamine ja projekti info haldamine.
- Vaataja - projekti vaataja. Projekti tasemel mänedžerist järgmise taseme õigustega kasutaja. Ta näeb kõike projektiga seonduvat kuid ise ei saa midagi muuta.
- Arendaja - projekti arendaja. Projekti tasemel madalaimate õigustega kasutaja. Eesmärgiks on olla seotud projektidega, kus ollakse arendaja.

3.2 Funktsionaalsed nõuded

Rakenduse funktsionaalsed nõuded, mille autor koostas koostöös ettevõttes töötavate projektijuhtide, arendajate ja tegevjuhiga, on järgnevad:

1. Kasutaja peab saama süsteemi sisse logida.

2. Kasutaja saab rakenduse funktsionaalsusi kasutada ainult siis, kui ta on sisse loginud.

3. Kasutaja saab rakenduses hallata ainult andmeid, mille haldamiseks tal on luba, mis tuleneb kasutaja rollist.

4. Süsteemis kustutatud andmed jäävad andmebaasi alles, aga ei ole enam kasutajatele nähtavad.

5. Arve koostamisel peab süsteem kokku koguma valitud ajaperioodil tehtud tunnid ajajälgimistarkvarast Hubstaff.

6. Rakendus muudab arve koostamisel valitud ajaperioodil tehtud töötunde vastavalt ettevõttesisestele reeglitele.

7. Administraator peab saama luua, vaadata, muuta ja kustutada kõiki rakenduses olevaid andmeid.

8. Administraator peab saama luua, vaadata, muuta ja kustutada kliente, projekte ja kasutajaid.

9. Ainult administraator peab saama kustutada arveid.

10. Ainult mänedžer ja administraator peavad saama luua rakenduses arveid.

11. Ainult mänedžer ja administraator peavad saama läbi rakenduses luua arveid raamatupidamistarkvaras Merit.

12. Mänedžer ja administraator peavad saama luua, vaadata, muuta ja kustutada ettevõttesiseseid reegleid.

13. Mänedžer peab saama luua, vaadata, muuta ja kustutada kõiki oma projektidega seotud andmeid.

14. Mänedžer peab saama muuta arveid, juhul kui need ei ole veel edasi saadetud raamatupidamistarkvarasse Merit.

15. Vaataja peab saama rakenduses vaadata kõiki oma projektidega seotud andmeid.

16. Arendaja peab saama rakenduses vaadata ainul oma kasutajaga seotud andmeid ja projekte, millega ta on seotud.

3.3 Mittefunktsionaalsed nõuded

Rakenduse mittefunktsionaalsed nõuded on järgnevad:

1. Tagaosa peale on võimalik ehitada kasutajaliides.
2. Päringud peavad olema kiired ja võimalikult optimeeritud.
3. Koodi peab olema võimalikult palju taaskasutatud.
4. Rakendust peab olema mugav kasutada.
5. Rakendust peab saama mitu kasutajat kasutada samal ajal.
6. Rakendus on tõrkekindel, aga tõrke korral peab vea põhjus olema kasutaja jaoks selge.
7. Rakendus peab olema võimalikult põhjalikult dokumenteeritud, et uue arendaja liitumisel või arendaja vahetusel oleks võimalik kiirest aru saada, mida rakendus teeb, millest koosneb ja mis ülesannet mingi osa täidab. Lisaks sellele peab kasutajaliidese loomiseks vajalik informatsioon olema ühes kohas ja piisavalt detailselt kirjeldatud, et ei oleks vaja tagaosa koodi uurida.
8. Rakendus ise ja rakendusega seotud informatsioon peab olema inglise keeles.

3.4 Raamistik

Enne arendustööga alustamist valis autor välja sobiva raamistiku, millele rakendus ülesse ehitada. Raamistik peab kokku sobima rakenduse nõuete ja spetsifikatsiooniga. Lõputöös keskendub autor rakenduse tagaosa arendusele, aga tulevikus on planeeritud rakendusele luua ka kasutajaliides ja sellest tulenevalt on üheks kriteeriumiks lihtsasti loodavad HTTP päringud. Raamistik peab võimaldama lihtsasti luua rakenduse vundamenti, et autor ei peaks aega kulutama vähe väärtus andvale tegevusele.

Valikus on mitmeid raamistikke, mida on võimalik rakenduse loomiseks kasutada. Mõned neist on:

- Symfony - universaalne veebiarendusplatvorm, mis põhineb PHP arenduskeelel. Symfony võimaldab hõlpsasti kasutada juba valmis PHP komponente ja vajadusel neid ise luua. [28]

- Laravel - PHP arenduskeelel põhinev veebirakendusraamistik. Niinimetatud karbist väljas lahendusena pakub Laravel juba mitmeid sisseehitatud funktsioone, mida ühel veebirakendusel on vaja. Sellest tulenevalt pole vaja kulutada aega ratta uuesti leiutamisele. Kõigele muule lisaks pakub Laravel veel sisseehitatud testimisvõimalust. [10]

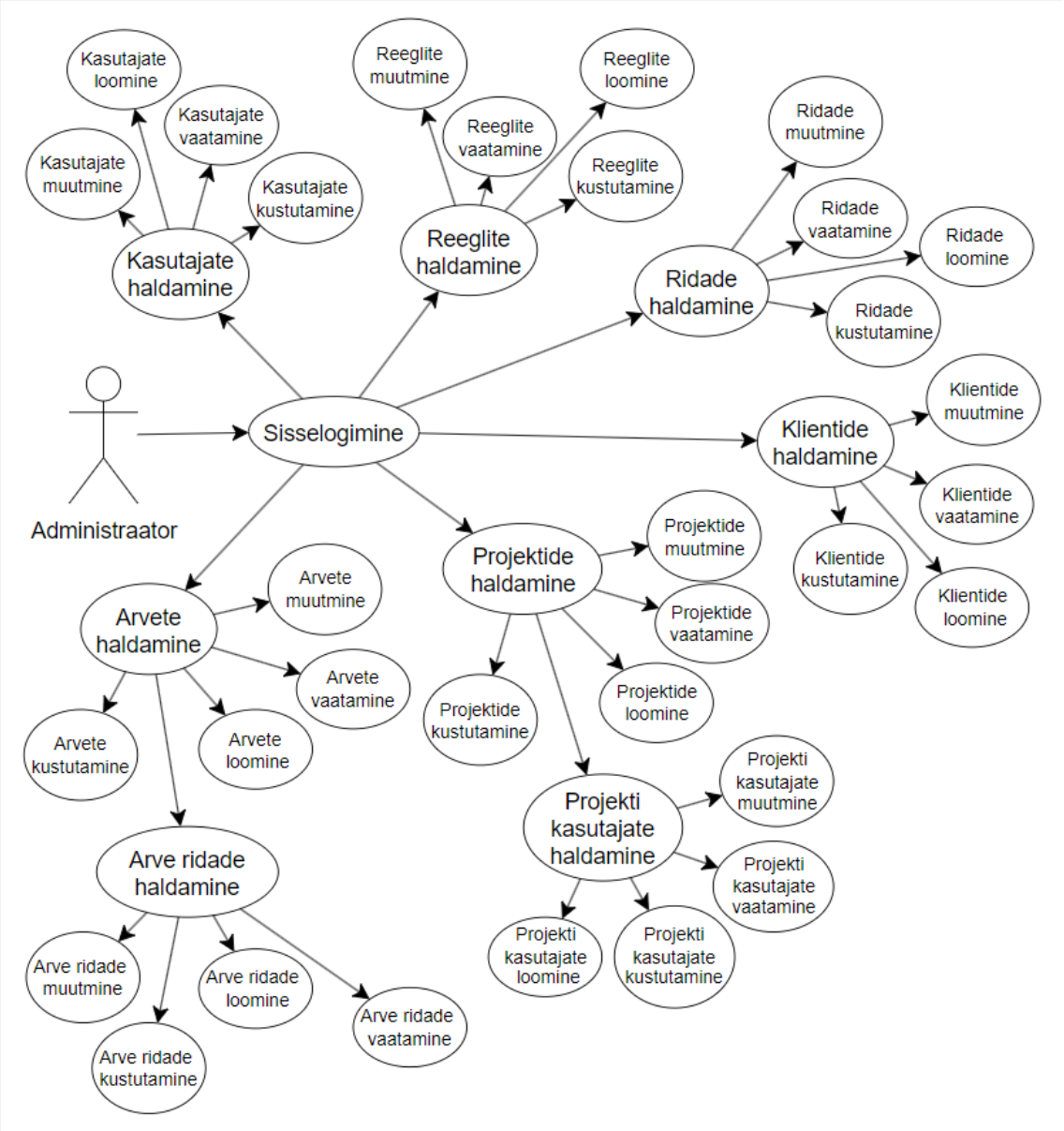
- Angular - raamistik, mis võimaldab ehitada SPA rakendusi, kasutades HTML'i ja TypeScripti. Angular võimaldab rakenduse ehitamisel kasutada juba valmis ehitatud Angulari komponente. [29]

Laravel pakub mitmed sisseehitatud lahendusi, mida on ühe rakenduse vundamendi loomiseks vaja. Tänu sellele ei pea rakendust nullist ehitama hakkama. Üldjuhul peab rakenduses ühe mudeliga seotud klassid ühendama manuaalselt, aga Laravel'is seda tegema ei pea ja raamistik saab ise aru, millised klassid peavad omavahel suhtlema. Lisaks on Laravel'il väga põhjalik dokumentatsioon ja läbi ettevõtte on autoril võimalus saada ligipääs Laracast'i tasulisele versioonile, kus on võimalik vaadata Laravel'i arendusega seotud õpetusi. Viimaks pakub antud raamistik autorile huvi ja sellest tulenevalt valis ta lõputöö tegemiseks Laravel'i raamistiku.

3.5 Kasutusjuhud

3.5.1 Administraatori kasutusjuhud

Administraator on kõige kõrgema taseme kasutaja, kes peab saama rakenduses teha kõike.



Joonis 2. Administraatori kasutusjuhtude mudel

Kasutusjuht	Kasutajate loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada kasutajaid.

Kasutusjuht	Projekti kasutajate loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada projekti kasutajaid.

Kasutusjuht	Reeglite loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada reegleid.

Kasutusjuht	Projektide loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada projekte.

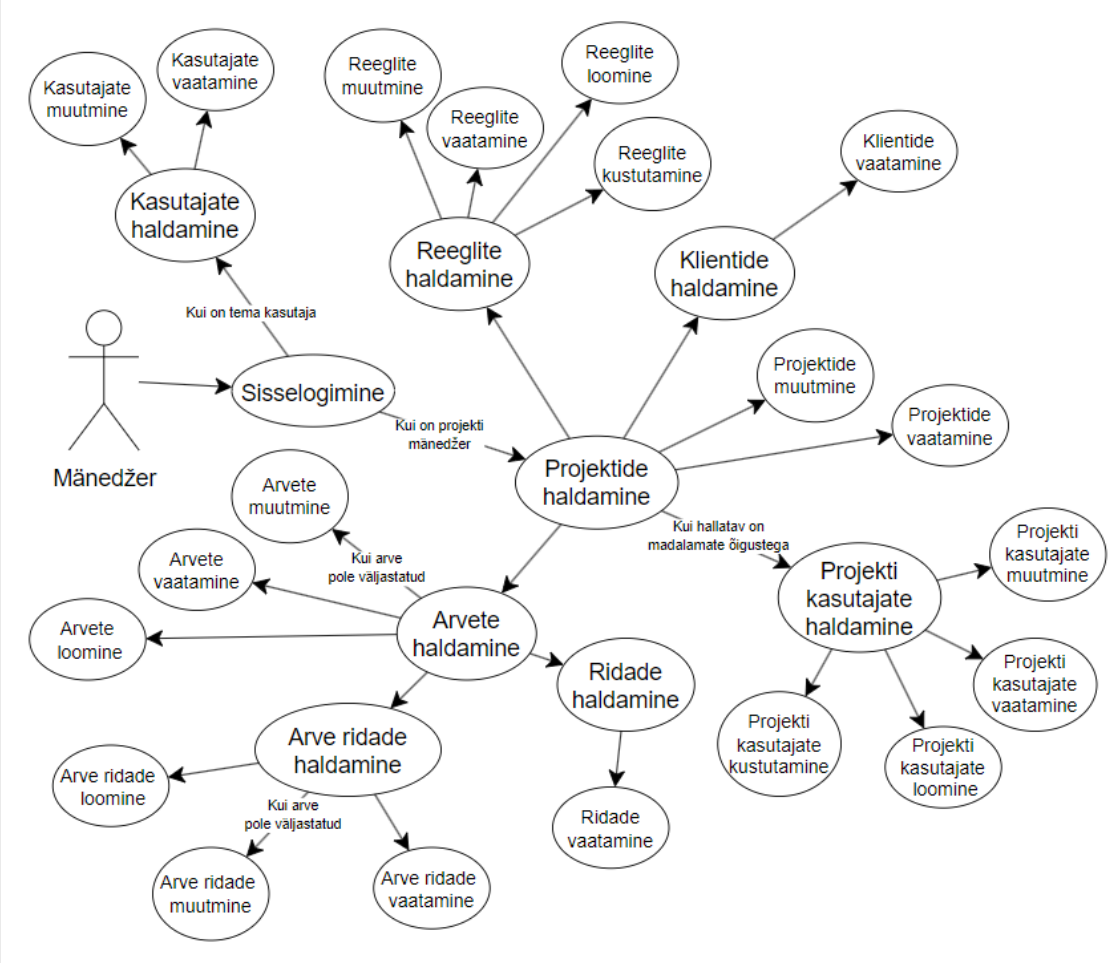
Kasutusjuht	Klientide loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada kliente.

Kasutusjuht	Arvete loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada arveid.

Kasutusjuht	Arve rea artiklite loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada arve rea artikleid.

Kasutusjuht	Rea artiklite loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Administraator peab saama luua, vaadata, redigeerida ja kustutada rea artikleid.

3.5.2 Mänedžeri kasutusjuhud



Joonis 3. Mänedžeri kasutusjuhtude mudel

Kasutusjuht	Projektide vaatamine
Kirjeldus	Mänedžer peab nägema projekte, kus ta on mänedžer, et saada ülevaade enda projektidest.

Kasutusjuht	Kliendi vaatamine
Kirjeldus	Mänedžer peab nägema kliente, kelle projektidel ta on mänedžer, et näha kliendiga seotud informatsiooni.

Kasutusjuht	Projekti kasutajate vaatamine

Kirjeldus	Mänedžer peab nägema kõiki oma projektidega seotud kasutajaid, et teada, kes projektiga töötavad.

Kasutusjuht	Projekti kasutajate loomine
Kirjeldus	Mänedžer peab saama luua seoseid kasutajate ja projektide vahel, kus ta on mänedžer, aga seda ainult juhul kui loodava sideme tase on väiksem kui mänedžeri oma (vaataja ja arendaja).

Kasutusjuht	Projekti kasutajate redigeerimine
Kirjeldus	Mänedžer peab saama redigeerida seoseid kasutajate ja projektide vahel, kus ta on mänedžer. Seda ainult juhul kui redigeeritava sideme algtase ja lõpptase on väiksemad kui mänedžeri oma (vaataja ja arendaja).

Kasutusjuht	Projekti kasutajate kustutamine
Kirjeldus	Mänedžer peab saama kustutada seoseid kasutajate ja projektide vahel, kus ta on mänedžer, aga seda ainult juhul kui kustutatava sideme tase on väiksem kui mänedžeri oma (vaataja ja arendaja).

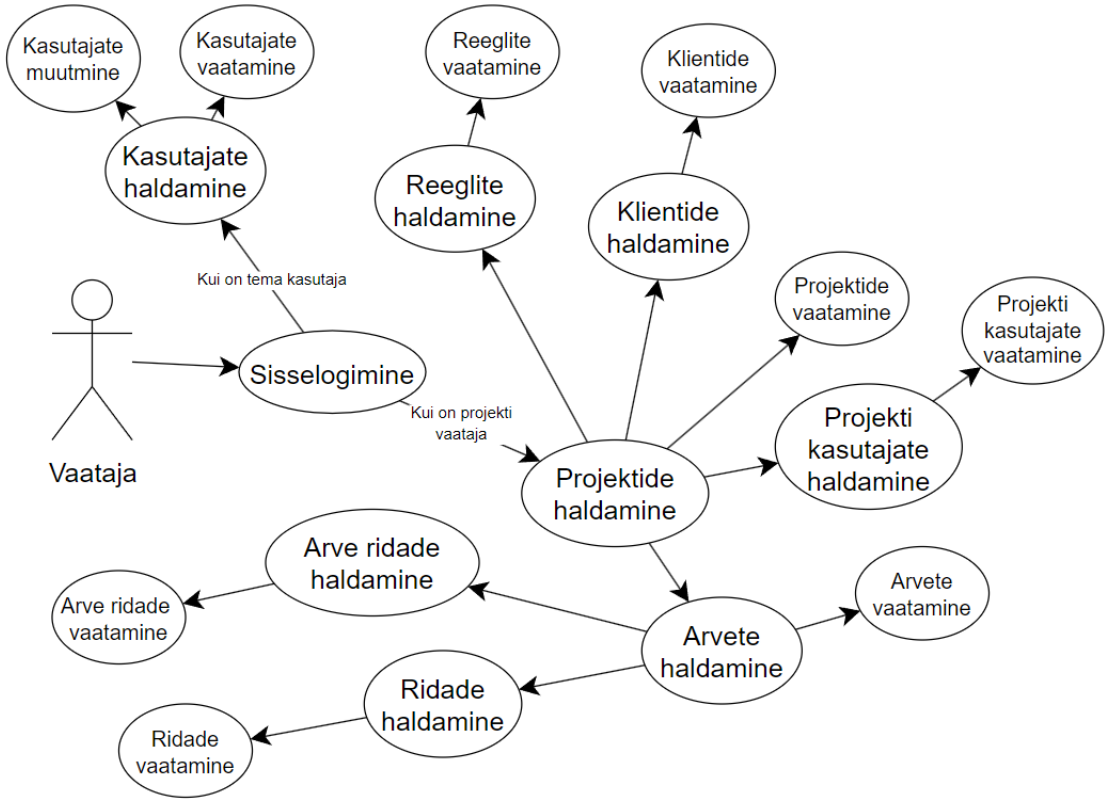
Kasutusjuht	Reeglite loomine, vaatamine, redigeerimine ja kustutamine
Kirjeldus	Mänedžer peab nägema kõiki oma projektidega seotud reegleid ja saama neid luua, vaadata, redigeerida ning kustutada, et need vastaksid ettevõtte nõuetele.

Kasutusjuht	Kasutajate vaatamine
Kirjeldus	Mänedžer peab nägema kasutajaid.

Kasutusjuht	Arve loomine
Kirjeldus	Mänedžer peab saama luua arveid projektidele, kus ta on mänedžer, et igakuiselt luua arveid ja need saata raamatupidamistarkvarasse.

Kasutusjuht	Arvete vaatamine
Kirjeldus	Mänedžer peab nägema kõikide projektide arveid, kus ta on mänedžer, et saada ülevaadet tehtud töödest ja väljasaadetud arvetest.

3.5.3 Vaataja kasutusjuhud



Joonis 4. Vaataja kasutusjuhtude mudel

Kasutusjuht	Projektide vaatamine
Kirjeldus	Vaataja peab nägema projekte, kus ta on vaataja, et saada ülevaade projektidest ja näha projektiga seotud informatsiooni.

Kasutusjuht	Kliendi vaatamine
Kirjeldus	Vaataja peab nägema kliente, kelle projektides ta on vaataja, et näha klientidega seotud infot.

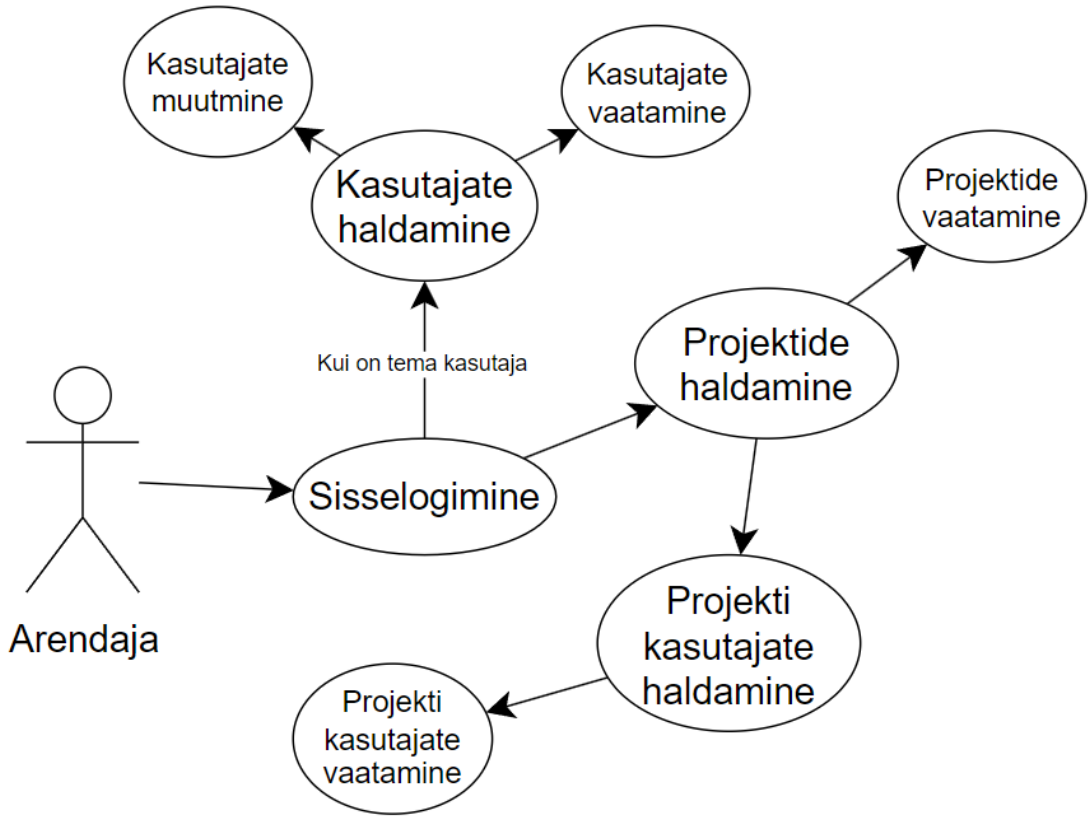
Kasutusjuht	Projekti kasutajate vaatamine
Kirjeldus	Vaataja peab nägema projekti kasutajaid nendel projektidel, kus ta on vaataja.

Kasutusjuht	Arvete vaatamine
Kirjeldus	Vaataja peab nägema nende projektide arveid, kus ta on vaataja, et saada ülevaadet tehtud töödest ja väljasaadetud arvetest.

Kasutusjuht	Reeglite vaatamine
Kirjeldus	Vaataja peab nägema projektide reegleid, kus ta on vaataja, et saada ülevaade ettevõttesisestest reeglitest.

Kasutusjuht	Kasutajate vaatamine
Kirjeldus	Vaataja peab nägema kasutajaid.

3.5.4 Arendaja kasutusjuhud



Joonis 5. Arendaja kasutusjuhtude mudel

Kasutusjuht	Kasutajate vaatamine
Kirjeldus	Arendaja peab nägema kasutajaid.

Kasutusjuht	Projekti vaatamine
Kirjeldus	Arendaja peab nägema projekte, kus ta on arendaja.

Kasutusjuht	Projekti kasutaja vaatamine
Kirjeldus	Arendaja peab nägema projekti kasutajaid nendel projektidel, kus ta on arendaja.

4 Rakenduse loomine

4.1 Arvete koostamine ja väljastamine

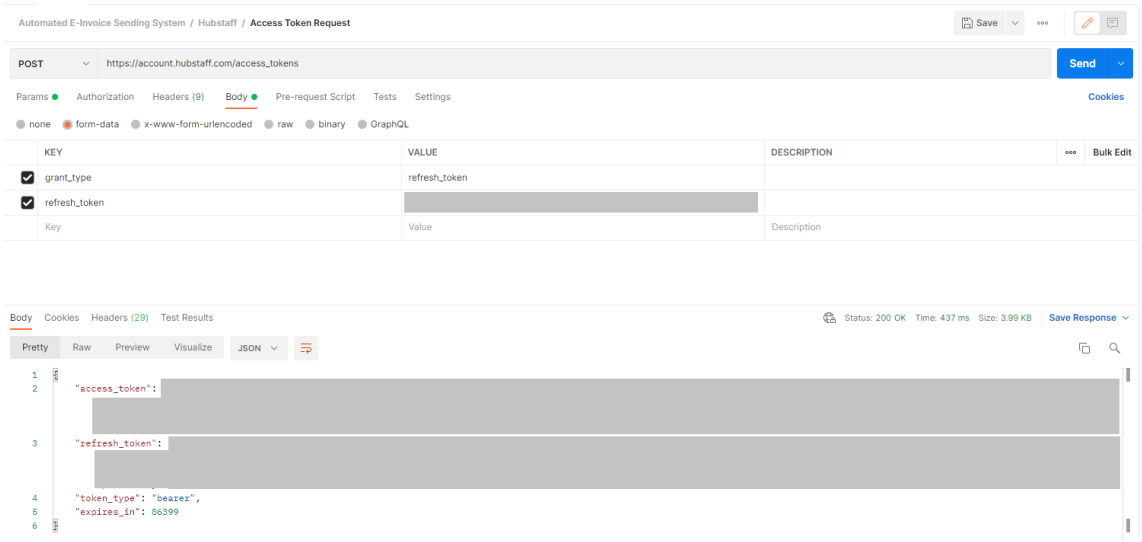
Autori poolt lõputöö raames loodava rakenduse peamiseks eesmärgiks on arvete automaatne koostamine ja nende väljastamine klientidele. Tavapäraselt koostavad autori ettevõttes töötavad projektijuhid arveid järgnevalt: 1) koguvad kokku tunnid Hubstaff'ist, 2) redigeerivad tunde ettevõttesiseste reeglitega, 3) sisestavad info Google Forms'i. Pärast seda sisestab tegevjuht informatsiooni manuaalselt raamatupidamistarkvarasse Merit.

Autor otsustas, et selle protsessi automatiseerimiseks on vaja luua neli erineva funktsionaalsusega osa. Rakenduse nõudest number 12 tulenevalt otsustas autor, et on vaja luua funktsionaalsus, mis võimaldab reegleid luua, vaadata, muuta ja kustutada. Teiseks on vaja luua integratsioon Hubstaff'iga, et rakendusel oleks võimalik pärida tehtud töötunde, mis katab omakorda funktsionaalse nõude number 5. Järgnevalt on vajalik teha lahendus, mis kogutud tunde vastavalt ettevõttesisestele reeglitele muudaks ja arveid koostaks. Viimase osana on vajalik integratsioon Meritiga, et nende portaalis oleks läbi API päringu võimalik arvet luua, see funktsionaalsus vastab nõudele number 11.

Esimesena lõi autor funktsionaalsuse, mille tulemusena on rakenduses võimalik talletada ettevõttesiseseid reegleid. Selleks lõi ta mudeli *rule* ja sellele omakorda loomise, vaatamise, muutmise ja kustutamise lõpp-punktid. Järgnevalt ühendas ta selle mudelitega *project* ja *user*, sest reegli kasutamiseks on vajalik teada, millise kasutaja ja projekti kohta see käib.

Järgmiseks tuli luua integratsiooni Hubstaff'iga. Hubstaff kasutab oma API jaoks tarkvara arhitektuuristiili REST. Selleks, et Hubstaff'i lõpp-punkte kasutada pidi autor looma autentimise loogika, et Hubstaff rakendusele andmeid tagastaks. Hubstaff kasutab autentimiseks *OAuth 2.0* autentimise süsteemi. Autentimise alguses tuleb Hubstaff'ist manuaalselt võtta personaalne värskendusmärk, millega tuleb teha päring https://account.hubstaff.com/access_tokens lõpp-punkti pihta. Päring saab vastuseks JSON'i, mille sees on: 1) juurdepääsumärk, mille abil saab teha päringuid teiste lõpp-punktide pihta, 2) uus värskendusmärk, mis on vajalik järgmise juurdepääsumärgi

päringuks, 3) kasutusaeg, kui kaua on juurdepääsumärk aktiivne ja 4) märgi tüüp, milleks on *bearer*. [30]



Joonis 6. Hubstaffi juurdepääsumärgi päringu näide Postmanis

Autor lõi Hubstaff'i autentimiseks lahenduse, mis kontrollib automaatselt, kas juurdepääsumärk on olemas ja kas see on veel aktiivne. Kui juurdepääsumärk puudub või on aegunud, siis genereeritakse uus märk. Loodud lahenduse eesmärgiks on autentimise täisautomatiseerimine nii, et midagi ei pea manuaalselt sisestama.

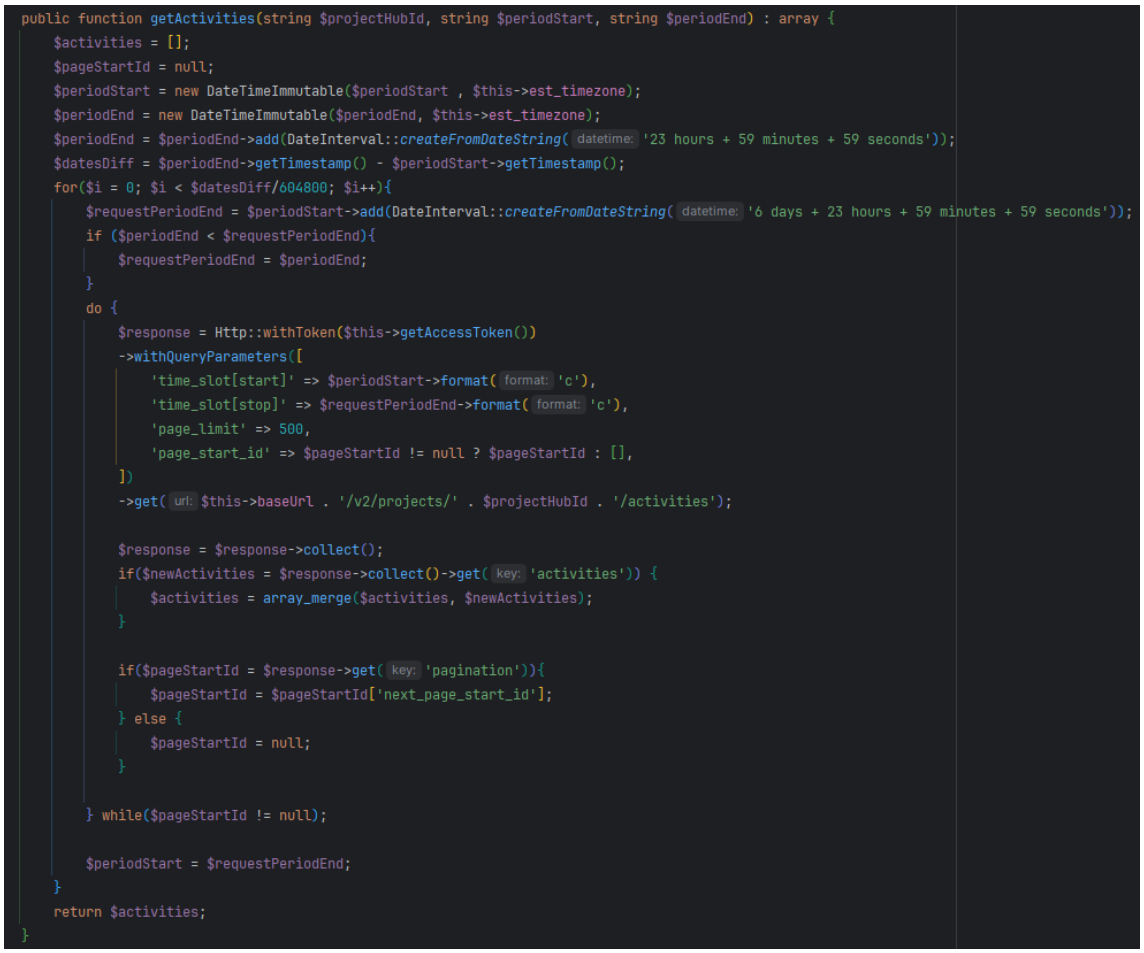
```
1 usage   Daniel Rasmus Põder +1
private function checkIfTokenNeedsToBeRefreshed() : bool {
    $hubstaffToken = HubstaffTokens::where('id', 1)-> first();
    if (empty($hubstaffToken->access_token)){
        return true;
    }
    $updateTime = $hubstaffToken->updated_at->timestamp + $hubstaffToken->expires_in;
    if ($updateTime <= time()-1800){
        return true;
    }
    return false;
}
```

Joonis 7. Hubstaff'i juurdepääsumärgi uuendamise kontrollfunktsioon

```
private function refreshAccessToken() : void {
    $response = Http::asForm()->post($this->tokenUrl, [
        'grant_type' => 'refresh_token',
        'refresh_token' => $this->refreshToken
    ]);
    $response = $response->collect();
    HubstaffTokens::where('id', 1)
        ->update(['access_token' => $response->get( key: 'access_token'),
                  'refresh_token' => $response->get( key: 'refresh_token'),
                  'expires_in' => $response->get( key: 'expires_in')]);
    $this->setVariablesValues();
}
```

Joonis 8. Hubstaff'i juurdepääsu- ja värskendusmärgi uuendamise funktsioon

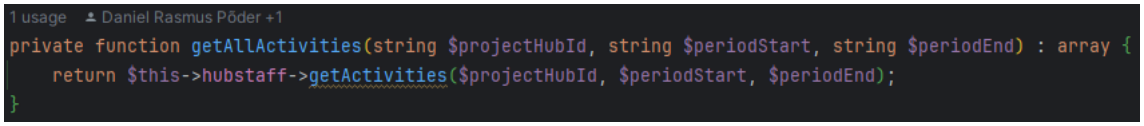
Arvete koostamiseks on rakendusel vaja teada kes ja kui palju on valitud perioodi jooksul erinevate ülesannete raames tööd teinud. Selle jaoks pidi autor looma funktsiooni, mis pärib Hubstaff'i API tegevuste lõpp-punktist kindlal ajavahemikul tehtud tegevusi. Antud funktsiooni sisenditeks on projekti Hubstaff'i identifikaator, perioodi algus- ja lõppkuupäevad. Hubstaff'i eripärast tulenevalt on tegevuste lõpp-punktist maksimaalselt võimalik pärida tegevusi ainult 7-päevase ajavahemiku kohta. Tavaliselt koostatakse arveid ühe kuu kohta ehk 7 päeva on liiga lühike ajavahemik. Antud probleemi lahendamiseks lõi autor funktsionaalsuse, mis vastavalt funktsiooni sisendi ajavahemikule teeb funktsioonis ühe või rohkem päringut, et kätte saada kõik sisendi ajavahemiku tegevused.



```
public function getActivities(string $projectHubId, string $periodStart, string $periodEnd) : array {
    $activities = [];
    $pageStartId = null;
    $periodStart = new DateTimeImmutable($periodStart , $this->est_timezone);
    $periodEnd = new DateTimeImmutable($periodEnd, $this->est_timezone);
    $periodEnd = $periodEnd->add(DateInterval::createFromDateString( datetime: '23 hours + 59 minutes + 59 seconds'));
    $datesDiff = $periodEnd->getTimestamp() - $periodStart->getTimestamp();
    for($i = 0; $i < $datesDiff/604800; $i++){
        $requestPeriodEnd = $periodStart->add(DateInterval::createFromDateString( datetime: '6 days + 23 hours + 59 minutes + 59 seconds'));
        if ($periodEnd < $requestPeriodEnd){
            $requestPeriodEnd = $periodEnd;
        }
        do {
            $response = Http::withToken($this->getAccessToken())
            ->withQueryParameters([
                'time_slot[start]' => $periodStart->format( format: 'c'),
                'time_slot[stop]' => $requestPeriodEnd->format( format: 'c'),
                'page_limit' => 500,
                'page_start_id' => $pageStartId != null ? $pageStartId : [],
            ])
            ->get( url: $this->baseUrl . '/v2/projects/' . $projectHubId . '/activities');

            $response = $response->collect();
            if($newActivities = $response->collect()->get( key: 'activities')) {
                $activities = array_merge($activities, $newActivities);
            }

            if($pageStartId = $response->get( key: 'pagination')){
                $pageStartId = $pageStartId['next_page_start_id'];
            } else {
                $pageStartId = null;
            }

        } while($pageStartId != null);

        $periodStart = $requestPeriodEnd;
    }
    return $activities;
}
```

Joonis 9. Funktsioon, mis pärib Hubstaff'ist kindlal ajavahemikul tehtud tegevusi

Kolmandaks tuli autoril luua lahendus, mis koostaks arveid ja neid vastavalt ettevõttesisestele reeglitele muudaks. Antud eesmärgi täitmiseks lõi ta eraldi klassi *InvoiceGenerator*. Esimesena lõi ta klassi funktsiooni, mis pärib läbi Hubstaff'i integratsiooni arve koostamiseks vajalikud tegevused. Selleks, et funktsioonis tehtav päring tagastaks rakendusele õige projekti tegevused, tuleb päringule kaasa panna projekti Hubstaff'i identifikaator, mis vastab *projects* tabelis väärtusele *hubstaff_id*.



```
1 usage   Daniel Rasmus Põder +1
private function getAllActivities(string $projectHubId, string $periodStart, string $periodEnd) : array {
    return $this->hubstaff->getActivities($projectHubId, $periodStart, $periodEnd);
}
```

Joonis 10. Kuvatõmmis funktsioonist, mis kogub kokku kindla perioodi jooksul tehtud töötunnid

Tulenevalt Hubstaff eripärast, mis tagastab maksimaalselt ühe töötaja ühe tööülesande kohta 10-minutilise tegevuse, pidi autor looma lahenduse, mis arvutaks kokku ühe tööülesande raames ühe töötaja poolt tehtud töö. Töötajaid kaardistatakse vastavalt JSON vastuses olevale *userId*, mis vastab *users* tabelis olevale *hubstaff_id* väärtusele.

```php
private function sumActivities(array $allActivities) : array {
    $summedActivities = [];
    $found = false;

    foreach ($allActivities as $timeSlot) {
        foreach ($summedActivities as &$sumActivity) {
            if(($timeSlot['task_id'] === $sumActivity['task_id'])
                && ($timeSlot['user_id'] === $sumActivity['user_id'])
                && ($timeSlot['project_id'] === $sumActivity['project_id'])) {
                $sumActivity['tracked_seconds'] += $timeSlot['tracked'];
                $found = true;
            }
        }

        if (!$found) {
            $summedActivities[] = [
                'task_id' => $timeSlot['task_id'],
                'user_id' => $timeSlot['user_id'],
                'project_id' => $timeSlot['project_id'],
                'tracked_seconds' => $timeSlot['tracked']
            ];
        }

        $found = false;
    }
    return $summedActivities;
}
```

Joonis 11. Kuvatõmmis funktsioonist, mis arvutab kokku ühe töötaja ühe tööülesande jaoks kulunud aja

Kõik arve jaoks vajalik informatsioon oli nüüd olemas ja autor sai järgnevalt luua lahenduse, mis redigeerib kogutud tunde vastavalt ettevõttesisestele reeglitele. Ettevõttesisesed reeglid on leitavad varem loodud tabelist *rules*. Loodud lahendus käib kõik projektiga seotud reeglid läbi ja vajaduse korral redigeerib valitud perioodi jooksul tehtud töötunde.

```php
private function applyRules(string $projectHubId, array $tasks) : array {
    $projectRules = Rule::with(['project' => function($query) use($projectHubId) {
        $query->where('hubstaff_id', $projectHubId );
    }, 'user'])->get()->whereNotNull( key: 'project');

    foreach ($tasks as &$task) {
        foreach ($projectRules as $rule) {
            if ($task['user_id'] == $rule['user']['hubstaff_id']){
                $task['tracked_seconds'] *= $rule['multiplier'];
            }
        }
    }

    return $tasks;
}
```

Joonis 12. Kuvatõmmis funktsioonist, mis rakendab ettevõttesiseseid reegleid

Kõige viimasena tegi lõputöö autor lahenduse, mis võimaldab saata arveid raamatupidamistarkvarasse Merit. Sarnaselt Hubstaff'ile pidi autor esimesena looma autentimise, et rakendusel oleks võimalik teha päringuid Meriti API pihta.

Meriti autentimiseks on vaja API identifikaatorit, ajatemplit ja signatuuri, mis luuakse API võtme krüpteerimisel HMAC-SHA-256 loogikaga, mille krüpteerimisevõti koosneb API identifikaatoril, ajatemplil ja päringu JSON kombinatsioonist. [31]

Ajatempli ja API identifikaatori sai autor kätte Meriti portaalist kuid signatuuri koostamise funktsionaalsus tuli tal eraldi ehitada. Selle jaoks lõi ta rakendusse funktsiooni, mis vahetult enne päringu tegemist arvutab kokku signatuuri. Signatuur luuakse API identifikaatori, ajatempli, API võtme ja päringu JSON osa põhjal.

```
private function getSignature(string $apiId, string $apiKey, string $timestamp, string $json) : string{
    $signable = $apiId.$timestamp.$json;
    $rawSig = hash_hmac( algo: 'sha256', $signable, $apiKey, binary: true);
    $base64Sig = base64_encode($rawSig);
    return $base64Sig;
}
```

Joonis 13. Merit'i signatuuri genereerimise funktsioon

Järgnevalt oli autoril vaja luua funktsioon, mis võimaldab rakendusel luua arveid Meriti portaalis. Selle jaoks on Meriti API's olemas arve loomise lõpp-punkt, mis võimaldab arve loomiseks kõik vajaliku päringule kaasa panna JSON vormis. [32]

Autor lõi rakenduses arve välja saatmiseks eraldi funktsiooni, mille sisendiks on arve identifikaator, mis vastab ühele arvele rakenduse andmebaasi tabelis *invoice*s. Funktsioon paneb vastavalt andmebaasis olevale infole arve JSON formaadis kokku, loob signatuuri ja teeb POST päringu Meriti API arve loomise lõpp-punkti pihta.

```php
public function createInvoice(int $invoiceId) {
    $apiId = 'd09745ce-e705-46bc-a28f-f18effc8c521';
    $apiKey = '2kvPataóXlnwrBx2ZO3dZwQyAclz/m1SYN7QCNOtLpY=';
    $timestamp = date( format: 'YmdHis');
    $CurTime = date( format: "Ymd");
    $customerId = 'd408fff0-424f-4c6d-9108-4cf26da2ea8c';

    $jsonBody = [
        'Customer' => [
            'Id' => $customerId,
        ],
        'DocDate' => $CurTime,
        'DueDate' => $CurTime,
        'TransactionDate' => $CurTime,
        'InvoiceNo' => "TEST123456",
        'RefNo' => null,
        'CurrencyCode' => "EUR",
        'InvoiceRow' => $this->getInvoiceLineItems($invoiceId),
        'RoundingAmount' => 5,
        'TotalAmount' => (string) $this->getLineItemsTotal($invoiceId),
        'TaxAmount' => [
            [
                'TaxId' => "b9b25735-6a15-4d4e-8720-25b254ae3d21",
                'Amount' => "75.00"
            ]
        ],
        'HComment' => '',
        'FComment' => '',
    ];

    $signature = $this->getSignature($apiId, $apiKey, $timestamp, json_encode($jsonBody));
    $response = Http::withBody(
        json_encode($jsonBody), contentType: 'application/json'
    )->post( url: $this->baseUrl.'/sendinvoice'.'?ApiId='.$apiId.'&timestamp='.$timestamp.'&signature='.$signature);
    return $response;
}
```

Joonis 14. Funktsioon, mis loob arve Merit'i raamatupidamistarkvaras

4.2 Autentimine

Rakenduse funktsionaalsetest nõuetest number 1 ja 2 tulenevalt otsustas autor rakendusele ehitada sisselogimise loogika. Sisselogimise ehk autentimise eesmärgiks on tagada veebirakenduse turvalisus. Selle tulemusena ei saa iga inimene rakenduse lõpp-punktidele ja sealt päritavale infole ligi kui selleks puudub vajalik luba. Paljudel juhtudel on autentimise arendamine keeruline, ajakulukas ja riskantne. Autentimise loomiseks pakub Laravel mugavaid tööriistu, mis teevad autentimise kiireks ja turvaliseks. Laraveli autentimise loogika põhineb peamiselt valvuritel ja pakkujatel. Valvurid määravad kuidas kasutajaid päringute puhul autenditakse. Näiteks on Laravel'is olemas seansivalvur, mis säilitab kasutaja oleku seanssi salvestuse ja küpsiste abil. [33]

Autor otsustas kasutada autentimiseks *Basic Authentication'it*, sest rakendus on veel arendusfaasis ja ei vaja kõige turvalisemat autentimise viisi. Laravel'is on selle

rakendamine väga lihtne. Autor pidi ainult lõpp-punktide deklareerimisele lisama käsu, mis lisab autentimise.

```
Route::middleware( middleware: 'auth:sanctum')->get( uri: '/user', function (Request $request) {
    return $request->user();
});

Route::group(['prefix' => 'v1', 'namespace' => 'App\Http\Controllers\Api\V1'], function() {
    Route::apiResource( name: 'projects', controller: ProjectController::class)->middleware( middleware: 'auth.basic:web');
    Route::apiResource( name: 'clients', controller: ClientController::class)->middleware( middleware: 'auth.basic:web');
    Route::apiResource( name: 'invoices', controller: InvoiceController::class)->middleware( middleware: 'auth.basic:web');
    Route::apiResource( name: 'line_items', controller: LineItemController::class)->middleware( middleware: 'auth.basic:web');
    Route::apiResource( name: 'rules', controller: RuleController::class)->middleware( middleware: 'auth.basic:web');;
    Route::apiResource( name: 'project_users', controller: ProjectUserController::class)->middleware( middleware: 'auth.basic:web');
    Route::apiResource( name: 'users', controller: UserController::class)->middleware( middleware: 'auth.basic:web');
});
```

Joonis 15. Kuvatõmmis autentimise rakendamisest

Peale antud käsu lisamist on lõpp-punktidel autentimine rakendatud. Lõpp-punktidel, millele on rakendatud *Basic Authentication*, ei ole võimalik ilma andmebaasis salvestatud emaili ja salasõna kombinatsioonita ligi pääseda.

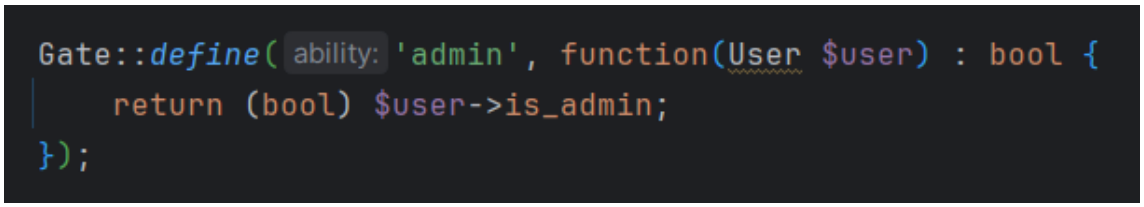
4.3 Autoriseerimine

Rakenduse viiendaks nõudeks on, et igal rakenduse mudelil peavad olema paika pandud õigused, kes neid luua, vaadata, muuta ja kustutada saavad. Sellest lähtuvalt otsustas autor, et antud rakenduses ei piisa ainult kasutaja autentimisest ja vajalik on ehitada autoriseerimine tagamaks, et andmetele pääsevad ligi ainult volitatud isikud.

Autoriseerimiseks pakub Laravel väga lihtsat ja organiseeritud viisi kontrollimaks, millele kasutaja pääseb ligi ja millele mitte. Laravel'i on sisse ehitatud autoriseerimiseks kaks tööriista, milleks on väravad ja poliitikad. Rakenduse loomisel ei pea valima ühe või teise vahel vaid on võimalik kasutada mõlemaid vastavalt vajadusele. [34]

Väravad on hea viis määramaks, kas kasutajal on konkreetse toimingu tegemiseks õigus või mitte. Hea tava kohaselt lisatakse väravad *AuthServiceProvider* kausta, kust neid saab vastavalt vajadusele välja kutsuda. Väravate puhul pannakse alati esimese parameetrina väravale kaasa kasutaja üksus, mille põhjal autoriseerimine teostatakse. [35]

Autor lõi rakenduses autenditud kasutaja autoriseerimiseks kolm väravat. Üks kontrollib, kas tegemist on administraatoriga või mitte. Teine kontrollib, kas autenditud kasutaja on projekti mänedžer või mitte. Kolmas kontrollib, kas autenditud kasutaja on projekti vaataja või mitte.

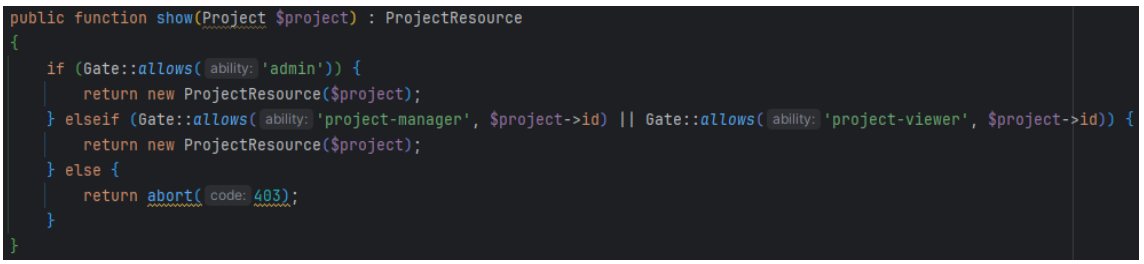


Joonis 16. Administraatori värava näidis

Joonis 17. Projekti mänedžeri värava näidis

Joonis 18. Projekti vaataja värava näidis

Näiteks on autor rakendanud antud väravaid, kui päritakse ühte kindlat projekti. Esimesena kontrollitakse kas kasutaja on administraator, kellel on ligipääs kõikidele projektidele. Teiseks määratakse värava abil kindlaks, kas tegemist on päritava projekti mänedžeri või vaatajaga. Kui kasutaja ei kuulu nendesse gruppidesse, siis antakse päringule vastuseks staatuskood 403, mis tähendab, et neil ei ole sellele üksusele ligipääsuluba.



Joonis 19. Väravate kasutuse näidis

Poliitikad on klassid, mis korraldavad autoriseerimise loogikat kindla mudeli või ressursi ümber. Näiteks on autor rakendanud *UserPolicy*, mis paneb paika *User* mudeli autoriseerimise õigused. Erinevalt väravatest luuakse poliitikate jaoks eraldi klassid, et iga mudeliga seonduv loogika oleks eraldatud. Laravel'i parimate praktikate kohaselt soovitatakse enamik mudelitega seonduv autoriseerimise loogika ülesse ehitada just poliitikate peale. Praktikas kasutatakse poliitikaid pigem suurematel projektidel autoriseerimiseks. [36]

Autor on antud rakenduses peamiselt kasutanud väravaid, sest need rahuldavad kõige paremini rakenduse eripärast tulenevat loogikat, kus erinevate tasemete kasutajatele on ühe lõpp-punkti väljundid erinevad.

Näiteks on lõputöö autor kasutanud poliitikaid projekti ressursi lõpp-punktide ümber, kus ainukene tegija on administraator. Nende lõpp-punktidega seotud autoriseerimise loogika on autor pannud eraldi klassi, *ProjectPolicy*. Selles klassis on autor seadistanud loogika projektide loomise ja uuendamise lõpp-punktide autoriseerimise kohta. Selleks, et eelpoolmainitud poliitikad töötaksid on autor need välja kutsunud vastavate lõpp-punkti funktsioonide sees.

```php
class ProjectPolicy
{
    /**
     * Determine whether the user can create models.
     */
    Daniel Rasmus Põder
    public function create(User $user): bool
    {
        return (bool) $user->is_admin;
    }

    /**
     * Determine whether the user can update the model.
     */
    Daniel Rasmus Põder
    public function update(User $user, Project $project): bool
    {
        return (bool) $user->is_admin;
    }
}
```

Joonis 20. Projekti poliitika näidis

```php
/**
 * Update the specified resource in storage.
 */
Daniel Rasmus Põder
public function update(UpdateProjectRequest $request, Project $project) : void
{
    $this->authorize( ability: 'update', $project);

    $project->update($request->all());
}
```

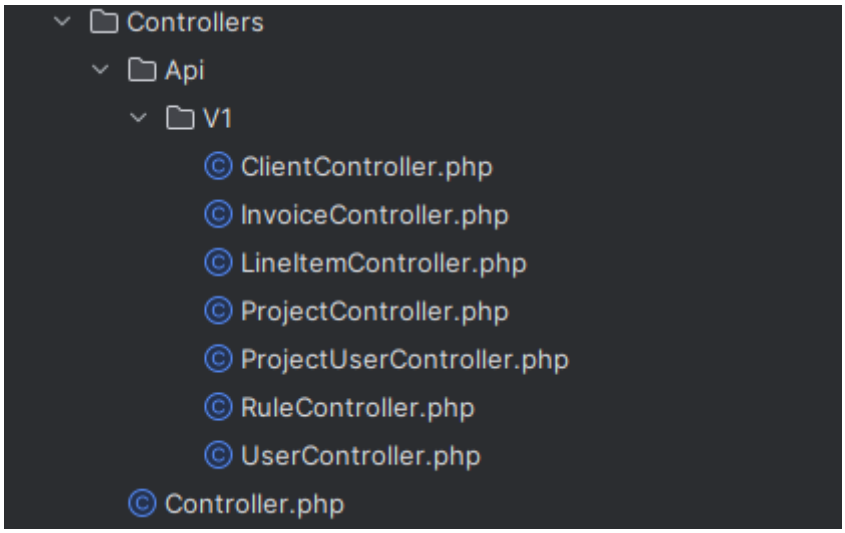
Joonis 21. Projekti uuendamise poliitika rakenduse näidis

5 Koodi analüüs

5.1 Nimetamiskonventsioon

Puhta koodi loomisel on tähtis, et erinevate komponentide nimetamisel tuleb kasutada ettenähtud viise. [37]

Kontrollerite nimetamisel tuleb kasutada *PascalCase*'i. Nimi peab olema ainsuses ja nime lõpus peab olema sõna "*Controller*". Näiteks *ProjectController*. [37]



Joonis 22. Rakenduse HTTP kontrollerid

Andmebaasi tabelite nimetamisel tuleb kasutada *snake_case'i* ja nimetused peavad olema mitmuses. Veergude nimetamisel tuleb samuti kasutada *snake_case'i* ja ei tohi viidata tabeli nimele. Näiteks kasutajaid hoiustava tabeli nimi peab olema *users*. [37]

```php
public function up(): void
{
    Schema::create( table: 'users', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'name');
        $table->string( column: 'email')->unique();
        $table->timestamp( column: 'email_verified_at')->nullable();
        $table->boolean( column: 'is_admin');
        $table->string( column: 'password');
        $table->integer( column: 'hubstaff_id');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Joonis 23. Kasutajate andmebaasi tabeli migratsioon

Ühendustabelite nimetamisel tuleb kasutada *snake_case'i* ja nimi peab koosnema mõlema tabeli nimest ainsuses ja tähestikulises järjekorras. Näiteks tabel *invoice_line_item*, mis loob ühenduse tabelite *invoice* ja *line_item* vahel. [37]

```php
public function up(): void
{
    Schema::create( table: 'invoice_line_item', function (Blueprint $table) {
        $table->id();
        $table->bigInteger( column: 'invoice_id')->unsigned();
        $table->foreign( columns: 'invoice_id')
            ->references( columns: 'id')
            ->on( table: 'invoices')
            ->onDelete( action: 'cascade')
            ->onUpdate( action: 'cascade');
        $table->bigInteger( column: 'line_item_id')->unsigned();
        $table->foreign( columns: 'line_item_id')
            ->references( columns: 'id')
            ->on( table: 'line_items')
            ->onDelete( action: 'cascade')
            ->onUpdate( action: 'cascade');
        $table->string( column: 'code');
        $table->string( column: 'description');
        $table->double( column: 'unit',  total: 6,  places: 2);
        $table->double( column: 'unit_price',  total: 8,  places: 2);
        $table->double( column: 'total',  total: 12,  places: 2);
        $table->timestamps();
    });
}
```

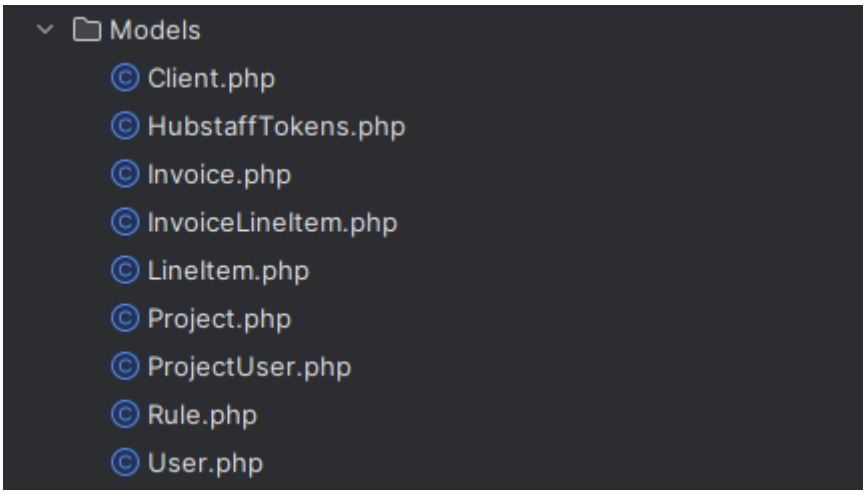
Joonis 24. Arvete ja reaüksuste ühendustabeli migratsioon

Muutujate nimetamisel tuleb kasutada *camelCase'i*, kus esimene täht on väike. Kui tegemist on muutujaga, mis hoiustab mitut väärtust, näiteks *array* või *collection*, siis peab ka nimi olema mitmuses. Näiteks *ClientContoller'is* olev *show* meetod. [37]

```php
public function show(Client $client) : ClientResource
{
    if (Gate::allows( ability: 'admin')) {
        return new ClientResource($client);
    } else {
        $allowedClients = Client::whereUserIsManager()->orWhere->whereUserIsViewer()->get();
        $currentClient = $allowedClients->where('id', $client->id)->first();
        if ($currentClient) {
            return new ClientResource($currentClient);
        }
    }
    return abort( code: 403);
}
```

Joonis 25. Muutujate nimetamise näide

Mudelite nimetamisel tuleb kasutada *PascalCase'i* ja nimetus peab olema ainsuses. Näiteks mudelite nimed. [37]



Joonis 26. Rakenduse mudelite nimetused

Meetodite nimetamisel tuleb kasutada *camelCase'i*, kus esimene täht on väike ja meetodi nimetus peab kirjeldama selle tegevust. Näiteks *Client* mudeli *scopeWhereUserIsManager* meetod. [37]

```
no usages  DanielPoder
public function scopeWhereUserIsManager(Builder $query) : void {
    $allowedProjects = Project::whereUserIsManager()->get();
    if ($allowedProjects) {
        $clientIds = $allowedProjects->pluck('client_id');
        $query->whereIn( column: 'id', $clientIds);
    }
}
```

Joonis 27. Meetodi nimetamise näide

Konstantide nimetamisel tuleb kasutada trükitähti ja sõnad tuleb eraldada alakriipsuga. Näiteks mudelis *ProjectUser* olevad konstandid. [37]

```
class ProjectUser extends Model
{
    5 usages
    const USER_MANAGER = "Manager";
    4 usages
    const USER_VIEWER = "Viewer";
    3 usages
    const USER_DEVELOPER = "Developer";
```

Joonis 28. Konstantide nimetamise näide

5.2 Funktsioonide tagastuse deklaratsioon

Puhta koodi loomisel on tähtis, et funktsioonidel on deklareeritud millist tüüpi väärtust funktsioon tagastab. Tänu sellele teame millist tüüpi on funktsioonist tagasi tulev väärtus ja saame seda kasutada teiste funktsioonide ja rakenduse ülesehitamisel. Vastasel juhul, kui me ei deklareeri tagastustüüpi, võib tekkida olukord, kus ebasobilikku tüüpi väärtuse tagastamisel tekib rakenduses või funktsioonis viga.

```
2 usages   Daniel Rasmus Põder +1
public function getAllowedTypes(String $userType) : array {
    if ($userType == 'Admin'){
        return [self::USER_MANAGER, self::USER_VIEWER, self::USER_DEVELOPER];
    } elseif ($userType == self::USER_MANAGER) {
        return [self::USER_VIEWER, self::USER_DEVELOPER];
    } else {
        return [];
    }
}
```

Joonis 29. Funktsiooni tagastuse deklareerimise näide

5.3 Printsiibid

Selleks, et loodav kood oleks puhas ja tulevikus teiste poolt arusaadav on vajalik jälgida väljatöötatud arendusprintsiipe. Erinevaid printsiipe on olemas mitmeid kuid autor analüüsis koodi kolme tuntuma printsiibi vastu, milleks on SOLID, KISS ja DRY.

5.3.1 SOLID

SOLID on objektorienteeritud arendustöö printsiip. See koosneb viiest osast: ühe vastutuse põhimõte, avatud-suletud põhimõte, Liskov'i asenduspõhimõte, liidese eraldamise põhimõte ja sõltuvuse ümberpööramise põhimõte. SOLID'i eesmärk on luua paremat ja puhast koodi, mida on mugav kasutada ja on arusaadav. Printsiip aitab: 1) ära hoida korduse, 2) kontrollida, kas kood on testitav, 3) luua koodi, millest inimene aru saab ja 4) hoida koodi kompaktsena. [38]

Laravel põhineb PHP arenduskeelel ja PHP on objektorienteeritud arenduskeel, millest tulenevalt on võimalik SOLID printsiibi rakendust analüüsida.

SOLID'i esimene põhimõte on ühe vastutuse põhimõte, mis tähendab, et ühel klassil peab olema üks tegevus ja kui tegevusi on rohkem kui üks, siis tuleb klass muuta kaheks erinevaks klassiks. [38]

Antud rakenduse puhul on hea näide sellest *HubstaffController'i* klass, mille ainukeseks ülesandeks on infot kätte saada Hubsaff'i API'st. Selle eesmärgi saavutamiseks on küll loodud mitu meetodit, aga klassi ülesanne on üks. [38]

Teiseks SOLID'i põhimõtteks on avatud-suletud põhimõte. See tähendab, et ei tohi olla lubatud muuta klassi, aga seda peab olema võimalik pikendada teiste klasside poolt. Antud rakenduse puhul seda põhimõtet vaja ei olnud. [38]

SOLID'i kolmandaks põhimõtteks on Liskov'i asenduspõhimõte. Antud põhimõte tähendab, et vanem klassi objekti on võimalik asendada laps-klassi objektiga nii, et rakendus töötab. Liskov'i asenduspõhimõttest rakenduses head näidet pole. [38]

Neljandaks SOLID'i põhimõtteks on liidese eraldamise põhimõte, millest tulenevalt peaks igal klassil olema liides, mis defineerib klassis kohustuslikud funktsioonid. Antud rakenduses ei ole seda põhimõtet kasutatud, aga on vajalik, et rakendus jälgiks äriloogikat. [38]

Viimaseks SOLID'i põhimõtteks on sõltuvuse ümberpööramise põhimõte. Antud põhimõte tähendab, et kõrgema tasemega klassid peaksid sõltuma liidestest ja abstraktsetest klassidest, mitte konkreetsetest klassidest. [38]

Arendatava rakenduse puhul on antud põhimõtet rakendatud ühe kohas, milleks on HTTP päringute filtreerimine. Antud juhul on loodud üks abstraktne baaskontrolleri klass, mida pikendavad teised HTTP kontrollerid.

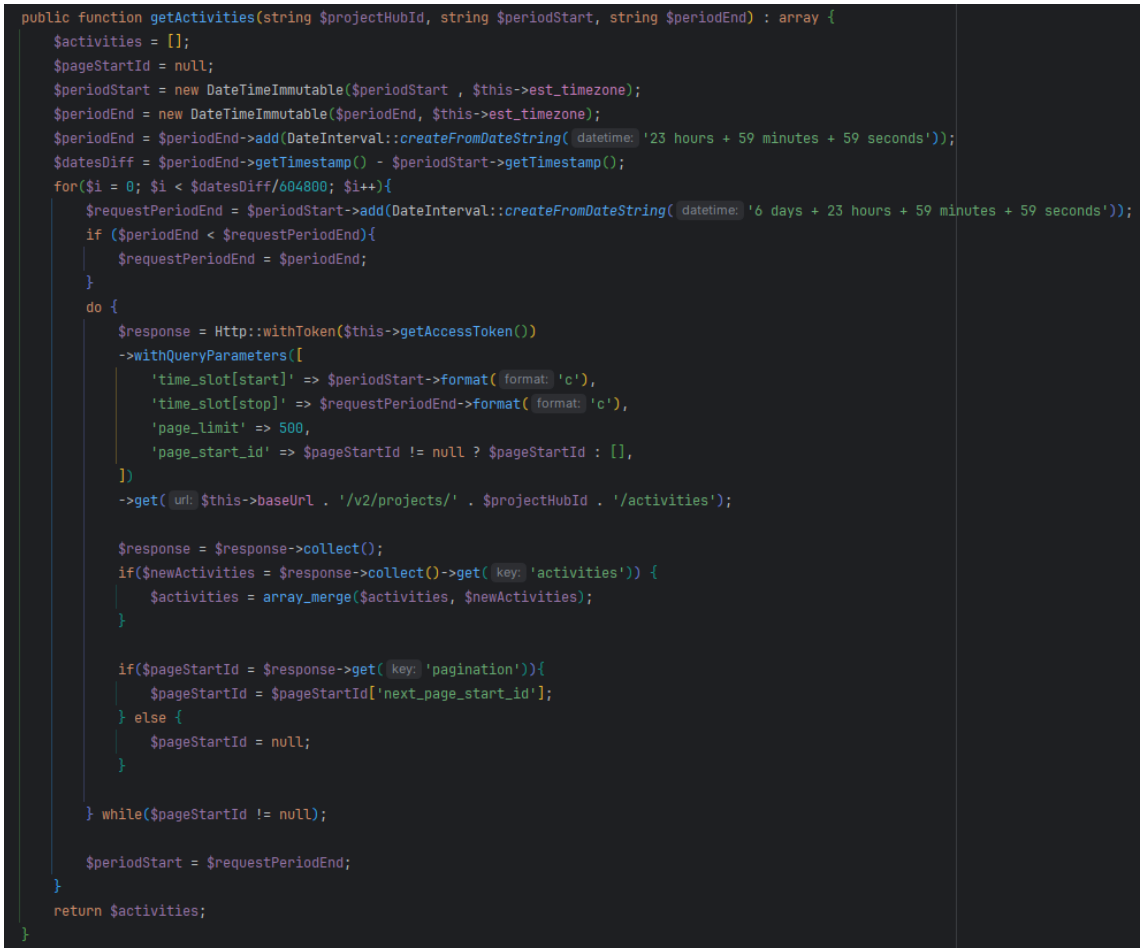
5.3.2 KISS

KISS on printsiip, mille peamine eesmärk on hoida koodi võimalikult lihtsana. KISS tähendab *"Keep it simple, stupid"*. Printsiibi peamine põhimõte on, et probleem tuleb lahendada võimalikult lihtsal viisil. Printsiip on kasutusel kui koodis ei ole meetodeid või üksuseid, mida kordagi ei kasutata; funktsioonid ja meetodid on võimalikult väikesed ja lihtsad; kirjutatud koodist on lihtne aru saada; igal klassil on üks eesmärk ja kasutatakse modulaarset programmeerimist. KISS printsiipi järgides on tulevikus koodi lihtsam

muuta ja hallata, teistel arendajatel on sellest lihtsam aru saada ja lihtsat koodi on kergem testida automaattestidega. [39, 40]

Kuigi rakenduse koodibaasis ei ole kõik nii lihtne kui võiks olla, on siiski üritatud järgida KISS printsiipi. Funktsioonide pikkused varieeruvad kahest reast kuni neljakümne reani. Enamjaolt on siiski meetodite pikkused alla kümne rea, mis lihtsustab koodist arusaamist ja loetavust. Selleks, et KISS põhimõtet veelgi rakendada tuleb pikemad funktsioonid, mis teevad hetkel mitut tegevust, kirjutada lahti väiksemateks funktsioonideks nii, et ühel funktsioonil on ainult üks ülesanne. Veel tuleb veenduda, et poleks loodud funktsioone tegevustele, mis on juba Laravel'i sisseehitatud.

Joonis 30. Kahe realise funktsiooni näide



Joonis 31. Neljakümnerealise funktsiooni näide

5.3.3 DRY

DRY on arendustöö printsiip, mille peamine eesmärk on vähendada koodi duplitseerimist. DRY tähendab *"Don't Repeat Yourself"*. Kui sama ülesandega kood on juba kasutusel kahes kohas, siis printsiibi kohaselt tuleks sellest luua abstraktne versioon, mida saavad kasutada mõlemad kohad, viidades abstraktsele versioonile. Selle tulemusena ei teki olukorda, kus sama eesmärgiga kood on duplitseeritud. Antud printsiibi järgimine aitab kaasa koodi puhtusele, sest ei teki segadust kui mitmes kohas on sama eesmärgiga funktsioon. [41]

Rakenduses on autor järginud DRY printsiipi. Hea näide sellest on HTTP päringute filtreerimine. Filtreerimine oli alguses defineeritud eraldi iga kontrolleri kohta, mille tulemusena oli sama eesmärgiga funktsioone seitse, iga funktsioon kaksteist rida pikk. DRY printsiipi järgides lõi autor sellest funktsioonist abstraktse versiooni, mida iga kontroller oma spetsiifiliste parameetritega välja kutsub. Selle ühe funktsiooni abstraktseks tegemine hoidis kokku viiskümmend kolm rida koodi, sest see on defineeritud ühes kohas, kus selle pikkuseks on kümme rida ja välja kutsutud seitsmes kontrolleris.

```php
protected function filterIfNeeded(Model $model, ResourceCollection $collection, ApiFilter $filter, Request $request) {
    $filterItems = $filter->transform($request);

    if (count($filterItems) === 0) {
        // No filtering needed return normal collection
        return new $collection($model::all());
    } else {
        // Filter by given parameters and return filtered collection.
        $items = $model::where($filterItems)->paginate();

        return new $collection($items->appends($request->query()));
    }
}
```

Joonis 32. Abstraktne filtri klass

```php
Daniel Rasmus Pöder +1
public function index(Request $request) : InvoiceCollection
{
    if (Gate::allows( ability: 'admin')){
        return $this->filterIfNeeded(new Invoice(), new InvoiceCollection(Invoice::all()), new InvoicesFilter(), $request);
    } else {
        return new InvoiceCollection(Invoice::whereUserIsManager()->orWhere->whereUserIsViewer()->get());
    }
}
```

Joonis 33. Abstraktse filtri funktsiooni kasutamine

6 Võimalused edasiseks arenduseks

Lõputöö käigus autori poolt loodud tagaosa rakendus vajab kasutamiseks tehnilisi teadmisi ja lahendust ei ole kõige mugavam kasutada, sest puudub kasutajaliides. Sellest tulenevalt on esimene võimalik edasiarendus rakendusele ehitada kasutajaliides, mille kaudu on mittetehniliste teadmistega töötajatel võimalik rakendust mugavalt kasutada.

Koodi analüüsi peatüki all selgus, et rakenduses ei ole kasutatud SOLID printsiibi liidese eraldamise põhimõtet. Seetõttu on järgmine võimalik edasiarendus antud põhimõtte rakendamine, mille tulemusena saavad arendajad kindlad olla, et rakendus jälgib ettevõtte äriloogikat.

Antud lõputöö raames ei jõudnud autor tulenevalt ajapuudusest rakendusele luua automaatteste, mis kontrolliksid rakenduse põhifunktsionaalsuseid. Testide olemasolu on iga rakenduse tähtis osa veendumaks, kas rakendus töötab korrektselt või mitte. Lisaks aitavad testid ennetada võimalikke vigu rakenduse töös.

Valminud rakendus saab olema autori ettevõtte üheks sisemiseks tööriistaks. Loodud baasile on võimalik ehitada ka teisi rakendusi, mida ettevõttel võib vaja minna. Siinkohal võib välja tuua näiteks puhkuste halduse süsteemi, uute töötajate õppekeskuse ja tööjõu haldamise ning planeerimise süsteemi.

7 Kokkuvõte

Bakalaureusetöö autori ettevõttes puudub arvete automaatseks koostamiseks vajalik tarkvara. Igakuiselt kulutatakse autori ettevõttes 7,5-12,5 tundi arvete koostamisele, samas kui seda aega saaks kasutada suurema lisandväärtusega tööülesannete täitmiseks. Lõputöö eesmärgiks oli luua rakendus, mille abil saaksid projektijuhid automaatselt koostada ja väljastada arveid. Rakenduse loomiseks koostas autor kõigepealt koos kolleegidega rakenduse nõuded ja spetsifikatsiooni ning valis välja sobilikud tööriistad. Järgnevalt lõi tööülesanded, mida hakkas kahenädalaste iteratsioonidega arendama. Lõputöö käigus ehitas autor tagaosa rakenduse, mis koostab ja väljastab arveid. Arve koostamisel kogub rakendus kokku ajajälgimisetarkvarast Hubsaff valitud ajaperioodil tehtud töötunnid. Rakenduse sisemine loogika loob arve ja redigeerib kogutud töötunde vastavalt ettevõttesisestele reeglitele. Kui arve on valmis, väljastab rakendus arve läbi integratsiooni raamatupidamistarkvaraga Merit. Lõputööd saab lugeda õnnestunuks, sest autori poolt loodud rakendus töötab ja täidab püstitatud eesmärki. Töö käigus sai täidetud kuueteistkümnest funktsionaalsest nõudest kolmeteist ja kaheksast mittefunktsionaalsest nõudest seitse.

Kasutatud kirjandus

[1] „What is an API? ” ibm.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.ibm.com/topics/api

[2] „What is DevSecOps? ” ibm.com. Kasutatud: 22 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.ibm.com/topics/devsecops#:~:text=DevSecOps%E2%80%94short%20for%20development%2C%20security,%2C%20deployment%2C%20and%20software%20delivery.

[3] „HTML Introduction” w3schools.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.w3schools.com/html/html_intro.asp

[4] „What is HTTP?” w3schools.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.w3schools.com/whatis/whatis_http.asp

[5] „Clearing the hurdles between you & your goals” hubstaff.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://hubstaff.com/about

[6] „What Is an IDE? ” codecademy.com. Kasutatud: 22 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.codecademy.com/article/what-is-an-ide

[7] „What is JavaScript?” w3schools.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.w3schools.com/whatis/whatis_js.asp

[8] „What is JSON? ” w3schools.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.w3schools.com/whatis/whatis_json.asp

[9] „Your Path to Laravel Mastery Starts Here...” laracast.com. Kasutatud: 10 jaanuar 2023. [Võrgumaterjal.] Saadaval: https://laracasts.com/

[10] „The PHP Framework for Web Artisans” laravel.com. Kasutatud: 16 detsember 2023. [Võrgumaterjal.] Saadaval: https://laravel.com

[11] „What is OAuth 2.0” auth0.com. Kasutatud: 10 jaanuar 2023. [Võrgumaterjal.] Saadaval: https://auth0.com/intro-to-iam/what-is-oauth-2

[12] „What is PHP? ” php.net. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.php.net/manual/en/intro-whatis.php

[13] „What is a REST API?” ibm.com. Kasutatud: 16 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.ibm.com/topics/rest-apis

[14] „SPA (Single-page application)” developer.mozilla.org. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://developer.mozilla.org/en-US/docs/Glossary/SPA

[15] „TypeScript Introduction” w3schools.com. Kasutatud: 16 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.w3schools.com/typescript/typescript_intro.php

[16] „What is the Windows Subsystem for Linux? ” microsoft.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://learn.microsoft.com/en-us/windows/wsl/about

[17] „We are ClickUp.” clickup.com. Kasutatud: 22 detsember 2023. [Võrgumaterjal.] Saadaval: https://clickup.com/about

[18] „About issues” docs.github.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues

[19] „Welcome to Jira Software” atlassian.com. Kasutatud: 22 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software

[20] „About GitHub Desktop” docs.github.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://docs.github.com/en/desktop/overview/about-github-desktop

[21] „Software. Faster.” gitlab.com. Kasutatud: 22 detsember 2023. [Võrgumaterjal.] Saadaval: https://about.gitlab.com

[22] „A brief overview of Bitbucket” bitbucket.org. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket

[23] „What is Postman?” postman.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.postman.com/product/what-is-postman/

[24] „Overview” ubuntu.com. Kasutatud: 14 detsember 2023. [Võrgumaterjal.] Saadaval: https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

[25] „Confluence basics” atlassian.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.atlassian.com/software/confluence/resources/guides/get-started/overview

[26] „Intelligent coding assistance” jetbrains.com. Kasutatud: 14 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.jetbrains.com/phpstorm/features/#:~:text=PhpStorm%20is%20an%20out%2Dof,refactoring%2C%20and%20zero%20configuration%20debugging.

[27] „ What is Slack?” slack.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://slack.com/help/articles/115004071768-What-is-Slack-#:~:text=Slack%20is%20a%20messaging%20app,transforms%20the%20way%20organizations%20communicate.

[28] „Introduction” symfony.com. Kasutatud: 14 detsember 2023. [Võrgumaterjal.] Saadaval: https://symfony.com/doc/current/create_framework/introduction.html#:~:text=Symfony%20is%20a%20reusable%20set,create%20your%20very%20own%20framework.

[29] „Introduction to Angular concepts” angular.io. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://angular.io/guide/architecture

[30] „Authentication” developer.hubstaff.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://developer.hubstaff.com/authentication

[31] „AUTHENTICATION” api.merit.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://api.merit.ee/connecting-robots/reference-manual/authentication/

[32] „CREATE SALES INVOICE” api.merit.com. Kasutatud: 18 detsember 2023. [Võrgumaterjal.] Saadaval: https://api.merit.ee/connecting-robots/reference-manual/sales-invoices/create-sales-invoice/#Endpointsv2

[33] „HTTP Basic Authentication” laravel.com. Kasutatud: 14 detsember 2023. [Võrgumaterjal.] Saadaval: https://laravel.com/docs/10.x/authentication#http-basic-authentication

[34] „Authorization” laravel.com. Kasutatud: 14 detsember 2023. [Võrgumaterjal.] Saadaval: https://laravel.com/docs/10.x/authorization#introduction

[35] „Gates” laravel.com. Kasutatud: 14 detsember 2023. [Võrgumaterjal.] Saadaval: https://laravel.com/docs/10.x/authorization#gates

[36] „Creating Policies” laravel.com. Kasutatud: 14 detsember 2023. [Võrgumaterjal.] Saadaval: https://laravel.com/docs/10.x/authorization#creating-policies

[37] „Laravel Naming Conventions” webdevetc.com. Kasutatud: 18 detsember 2023. [Võrgumaterjal.] Saadaval: https://webdevetc.com/blog/laravel-naming-conventions/

[38] G. K. Arora. „SOLID Principles” freedownloads247.com. Kasutatud: 1 jaanuar 2024. [Võrgumaterjal.] Saadaval: https://www.freedownloads247.com/UploadedFiles/8-2017/4223/solidprinciplessuccinctly.pdf

[39] D. Periwal. „The KISS (Keep It Simple and Straightforward) Principles for OR-Mapping Products from Software Tree, Inc” softwaretree.com. Kasutatud: 27 detsember 2023. [Võrgumaterjal.] Saadaval: https://softwaretree.com/v1/white-paper-pdf/KISSPrinciples.pdf

[40] „KISS Software Design Principle” baeldung.com. Kasutatud: 10 detsember 2023. [Võrgumaterjal.] Saadaval: https://www.baeldung.com/cs/kiss-software-design-principle

[41] „The DRY Principle: Benefits and Costs with Examples” thevaluable.dev. Kasutatud: 27 detsember 2023. [Võrgumaterjal.] Saadaval: https://thevaluable.dev/dry-principle-cost-benefit-example/

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

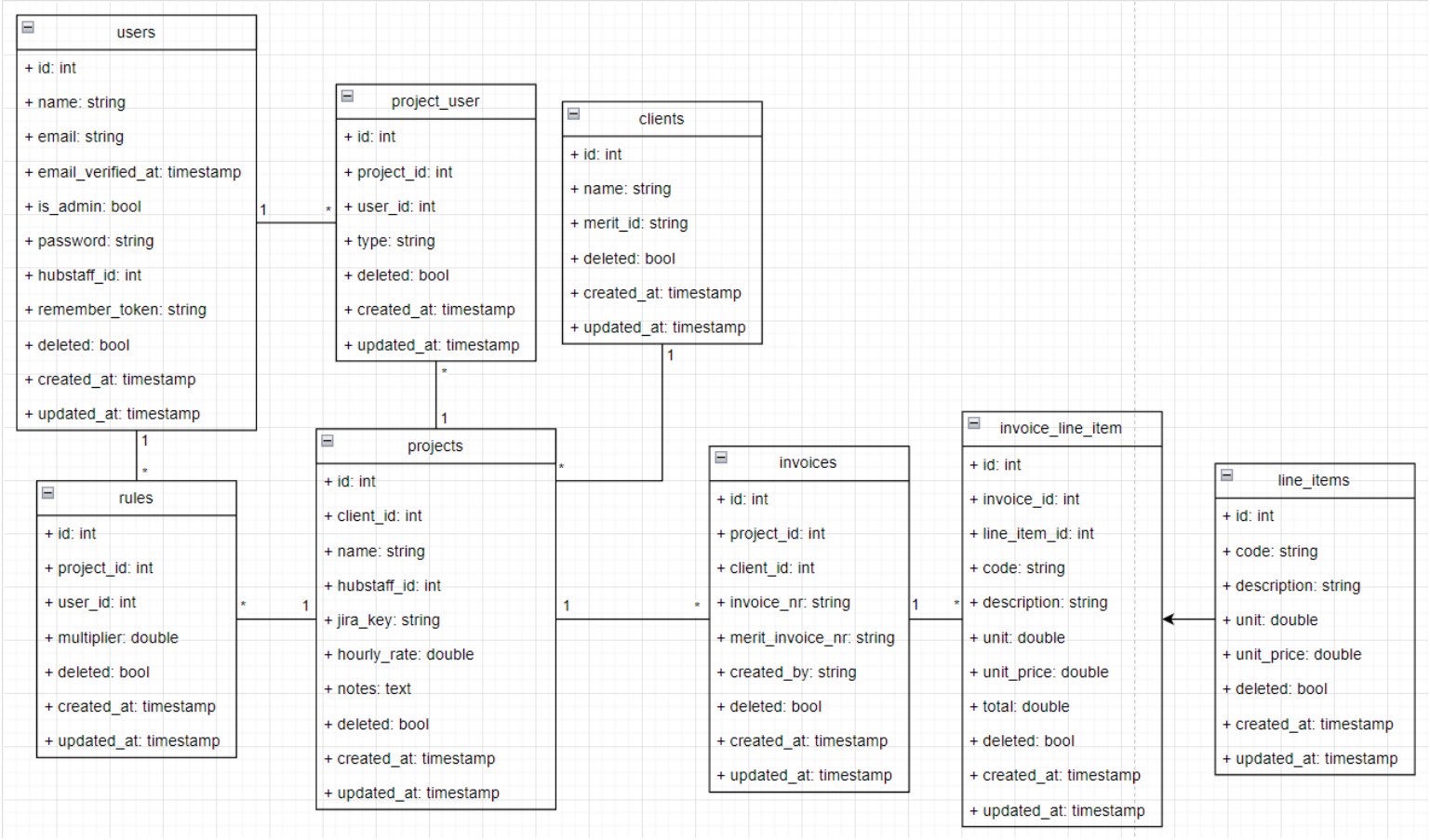
Mina, Daniel Rasmus Põder

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Automaatne arvete koostamise ja väljasaatmise rakendus“, mille juhendaja on Liisa Jõgiste.
   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.

10.01.2024

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 mudel



Lisa 3 – Andmebaasi tabelite täpne ülevaade

Tabel 1. Projektide andmebaasi tabeli kirjeldus

projects				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah
client_id : FK	integer	Võõrvõti, väärtus peab olemas olema clients tabelis.	-	Jah
name	string	Nimi ei tohi koosneda tühikutest.	-	Jah
hubstaff_id	integer	Väärtus Hubstaff'i andmebaasis, mis vastab antud projektile.	-	Jah
jira_key	string	Väärtus JIRA'i andmebaasis, mis vastab antud projektile.	-	Jah
hourly_rate	double	Antud projekti baastunnihind.	-	Jah
notes	text	Projektiga seotud märkmed	-	Ei
deleted	bool	Kas antud projekt on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud projekt loodi.	Hetke timestamp	Jah
updated_at	timestamp	Millal antud projekti viimati uuendati.	Hetke timestamp	Ei

Tabel 2. Klientide andmebaasi tabeli kirjeldus

clients				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah
name	string	Nimi ei tohi koosneda tühikutest.	-	Jah
merit_id	string	Väärtus Meriti andmebaasis, mis vastab antud kliendile.	-	Jah
deleted	bool	Kas antud klient on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud klient loodi.	Hetke timestamp	Jah
updated_at	timestamp	Millal antud klienti viimati uuendati.	Hetke timestamp	Ei

Tabel 3. Kasutajate andmebaasi tabeli kirjeldus

users				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah
name	string	Nimi ei tohi koosneda tühikutest.	-	Jah
email	string	Peab sisaldama @ sümbolit.	-	Jah
email_verified_at	timestamp	Millal kasutaja meili ära kinnitas.	-	Ei
is_admin	bool	Kas kasutaja on administraator või mitte	false	Jah

password	string	-	-	Jah
hubstaff_id	integer	Väärtus Hubstaff'i andmebaasis, mis vastab antud projektile.	-	Jah
remeber_token	string	-	-	Ei
deleted	bool	Kas antud kasutaja on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud kasutaja loodi.	Hetke timestamp	Jah
updated_at	timestamp	Millal antud kasutajat viimati uuendati.	Hetke timestamp	Ei

Tabel 4. Reeglite andmebaasi tabeli kirjeldus

rules				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah
project_id : FK	integer	Võõrvõti, väärtus peab olemas olema projects tabelis.	-	Jah
user_id : FK	integer	Võõrvõti, väärtus peab olemas olema users tabelis.	-	Jah
multiplier	double	Kasutaja ja projekti spetsiifiline baastunnihinna kordaja.	-	Jah
deleted	bool	Kas antud reegel on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud reegel loodi.	Hetke timestamp	Jah

updated_at	timestamp	Millal antud reeglit viimati uuendati.	Hetke timestamp	Ei

Tabel 5. Projekti kasutajate andmebaasi tabeli kirjeldus

project_user				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah
project_id : FK	integer	Võõrvõti, väärtus peab olemas olema projects tabelis.	-	Jah
user_id : FK	integer	Võõrvõti, väärtus peab olemas olema users tabelis.	-	Jah
type	string	Mis tüüpi õigused on antud kasutajal selle projekti raames.	-	Jah
deleted	bool	Kas antud project_user on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud project_user loodi.	Hetke timestamp	Jah
updated_at	timestamp	Millal antud project_userit viimati uuendati.	Hetke timestamp	Ei

Tabel 6. Arvete andmebaasi tabeli kirjeldus

invoices				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah

project_id : FK	integer	Võõrvõti, väärtus peab olemas olema projects tabelis.	-	Jah
client_id : FK	integer	Võõrvõti, väärtus peab olemas olema clients tabelis.	-	Jah
invoice_nr	string	-	Automaatselt genereeritud sisemine arve number	Jah
merit_invoice_nr	string	Peale arve saatmist Meritisse, sealt tagasi saadud Meriti arve number	-	Jah
created_by	string	Kasutaja, kes koostas arve	-	Jah
deleted	bool	Kas antud arve on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud arve loodi.	Hetke timestamp	Jah
updated_at	timestamp	Millal antud arvet viimati uuendati	Hetke timestamp	Ei

Tabel 7. Reaüksuste andmebaasi tabeli kirjeldus

line_items				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah
code	string	Merit tarkvarast tulev rea kood	-	Jah
description	string	Merit tarkvarast tulev rea kirjeldus	-	Jah

unit	double	Merit tarkvarast tulev rea ühik	-	Jah
unit_price	double	-	-	Jah
deleted	bool	Kas antud rida on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud rida loodi.	Hetke timestamp	Jah
updated_at	timestamp	Millal antud rida viimati uuendati	Hetke timestamp	Ei

Tabel 8. Arve reaüksuste andmebaasi tabeli kirjeldus

invoice_line_item				
Veerg	**Tüüp**	**Väärtus**	**Vaikeväärtus**	**Kohustuslik**
id : PK	integer	-	Automaatne increment	Jah
invoice_id : FK	integer	Võõrvõti, väärtus peab olemas olema invoices tabelis	-	Jah
line_item_id : FK	integer	Võõrvõti, väärtus peab olemas olema line_items tabelis	-	Jah
code	string	-	Tuleb tabeli line_items code veerust	Jah
description	string	Arve rea kirjeldus	Tuleb tabeli line_items description veerust	Jah
unit	double	Tund, tükk või kogus	Tuleb tabeli line_items unit veerust	Jah
unit_price	double	-	Tuleb tabeli line_items unit_price veerust	Jah

total	double	Tundi arvu ja unit_price korrutis	-	Jah
deleted	bool	Kas antud arve rida on kustutatud või mitte.	false	Jah
created_at	timestamp	Millal antud arve rida loodi.	Hetke timestamp	Jah
updated_at	timestamp	Millal antud arve rida viimati uuendati	Hetke timestamp	Ei