TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Piret Gorban 185967IADB

Arendustööde planeerimise ja prioriteetide määramise rakenduse analüüs ja väljatöötamine

Bakalaureusetöö

Juhendaja: Nadežda Furs

MBA

Tallinn 2021

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: Piret Gorban

16.05.2021

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on luua tööde planeerimise rakenduse analüüs ja esmane kasutatav versioon rakendusest, mis võimaldab ülesannete prioriteete arvutada vastavalt valitud tehnikale. Loodava rakenduse jaoks sobivaim prioriteetide määramise metoodika on WSJF (*Weighted Shortest Job First*), mis võimaldab töid olulisuse alusel järjestada võttes arvesse töö suurust, ärilist väärtust, ajakriitilisust ning seda, kuivõrd maandatakse sellega riske või luuakse võimalusi tulevikuks.

Töös on kajastatud erinevaid tehnoloogiaid, mida rakenduse loomiseks kaaluti ning analüüsitud nende sobivust valmiva rakenduse jaoks. Oluline rõhk on rakenduse arhitektuuri ja disaini kavandamisel.

Bakalaureusetöö käigus valmib töötav rakendus, mis võimaldab töid sisestada, muuta, kustutada ja kommenteerida ning algus- ja lõppkuupäeva põhjal neid graafilisel ajaskaalal kuvada. Samuti on kasutajatel võimalik näha tööde muutmise ajalugu. Rakendus annab võimaluse töödele prioriteete arvutada mitme kasutaja hinnangu põhjal, kusjuures hindamine toimub asünkroonselt määratud perioodi jooksul. Arenduse käigus teostatakse nii manuaalset testimist kui kasutatakse üksusteste rakenduse korrektses toimimises veendumiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 5 peatükki, 10 joonist, 4 tabelit.

Abstract

Analysis and Development of the Application for Planning and Prioritizing the Development Tasks

The aim of this final thesis is to perform an analysis and to create an initial working version of an application that allows to plan and prioritize development tasks according to chosen prioritization technique. Most suitable technique for this particular application is WSJF (Weighted Shortest Job First) that allows to rank tasks taking into account their size, business value, time criticality and the fact how much are they able to mitigate possible risks or provide opportunities.

The thesis describes different technologies considered for building the application and provides an analyzis of how suitable they are for the particular application. An important emphasis is on the architecture and design of the application. Suitable techniques were chosen to collect and prioritize requirements for the application and visualize main activities performed by users using flow charts.

In the process of writing this thesis one of the outcomes is a working application that can be used to add, edit and delete development tasks, add comments to those tasks and display tasks on graphic timeline based on start and end date. Users can also see the log of changes made by other users for every task. Application allows to calculate priorities for tasks based on several users opinions. The evaluation of tasks is asynchronous providing the users the possibility to grade tasks at the time that suits them best during the period that priority voting is open. The application was tested both manually and using unit tests to confirm expected behaviour.

The thesis is in Estonian and contains 35 pages of text, 5 chapters, 10 figures, 4 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendustarkvara liides
AS	Aktsiaselts
BLL	*Business logic layer*, äriloogika kiht
BPMN	*Business process model and notation* – äriprotsesside mudel ja graafiline märgistik
CSS	*Cascading style sheets*, kujunduskeel
DAL	*Data access layer*, andmetele juurdepääsu kiht
DTO	*Data transfer object*, andmeedastusobjekt
EF	*Entity Framework*, raamistik andmebaasiga suhtlemiseks
FURPS+	*Functionality*, *usability*, *reliability*, *perfrmance*, *supportability* – nõuete klassifitseerimise metoodika
Guid	*Globally unique identifier*, andmebaasi esmase võtme tüüp
HMAC	*Hash-based message authentication code*, info tervikluse kontrolli meetod
HTML	*Hyper text markup language*, hüperteksti märgistuskeel
HTTP	*Hypertext transfer protocol*, meetod andmete edastamiseks veebis
HTTPS	*Hypertext transfer protocol secure*, meetod andmete edastamiseks veebis krüpteeritud kujul
IT	Infotehnoloogia
JSON	*JavaScript Object Notation*, andmete formaat
JWT	*JSON Web Token*, JSON formaadis veebitõend
LINQ	*Language integrated query*, Entity Framework Core päringukeel
LTS	*Long term support*, pikaajalist tuge omav versioon
MoSCoW	*Must have, should have, could have, won't have* – nõuete olulisuse alusel grupeerimise metoodika
MPA	*Multi-page application*, mitmeleheline veebirakendus
MSSQL Server	Microsoft SQL Server, Microsofti loodud relatsiooniline andmebaaside haldussüsteem
MVC	*Model-View-Controller*, mudel-vaade-kontroller disainimuster
QFD	*Quality function deployment*, kliendikeskne tootekavandamine

RESTful	*Representational State Transfer,* tarkvara arhitektuuri stiil
SPA	*Single-page application,* üheleheline veebirakendus
SQL	*Structured Query Language*, andmebaasi päringukeel
SQL/PSM	*Structured Query Languge / Persistent Strored Modules* – andmebaasi päringukeele laiend
UML	*Unified modelling language,* ühtne modelleerimiskeel
UoW	*Unit of work,* tööühik
WSJF	*Weighted Shortest Job First*, prioriteetide määramise tehnika kaalutud lühim töö esimesena
XML	*Extensible markup language,* laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Arendusprotsessi käigus võib sageli kerkida üles küsimus, millised tööd tuleks esmajärjekorras teostada. Ülesannetele prioriteetide seadmine võib osutuda subjektiivseks ning sõltuda sellest, kes neid määrab. Selle vastu aitab koondhinnangu loomine meeskonnaliikmete arvamuste põhjal, ent sel juhul tuleb arvestada, et valdkonnast vähem teadmisi omavatel liikmetel võib olla raske adekvaatset hinnangut anda. Kui aga kasutada prioriteedi arvutamiseks omakorda kindlat tehnikat, võib see lõpptulemust oluliselt parandada.

Käesoleva bakalaureusetöö eesmärk on luua tööde planeerimise rakenduse analüüs ja esmane kasutatav versioon rakendusest, mis võimaldab ülesannetele prioriteete arvutada vastavalt valitud tehnikale. Töö käigus selgitatakse välja, milline tehnika selleks kõige paremini sobib.

Rakenduse loomise motivatsioon tuleb firmalt Infotark AS, kelle vajadustest lähtuvalt analüüs läbi viiakse. Praegu kasutatakse tööde planeerimiseks mitmeid rakendusi paralleelselt, kuna ükski ei sisalda kogu soovitud funktsionaalsust. On tehtud katseid prioriteetide arvutamiseks, ent mugavat lahendust selle jaoks leitud ei ole.

1.1 Metoodika

Bakalaureusetöö esimeses osas kirjeldatakse probleemi, mida lahendada püütakse ning selgitatakse nõuete kogumise ja nende olulisuse määramise käigus välja kriteeriumid, mille alusel olemasolevaid lahendusi võrrelda. Sellele järgneb eksisteerivate rakenduste võrdlus ja sobivuse analüüs probleemi lahendamiseks. Seejärel analüüsitakse arendustööde prioriteetide määramise tehnikaid ning valitakse loodava rakenduse jaoks sobivaim välja.

Järgnev osa hõlmab vahendite võrdlust ja valikut serveripoolse rakenduse, andmebaasi ja klientrakenduse loomiseks. Funktsionaalsuse visualiseerimiseks kasutatakse voodiagramme ja rakenduse prototüüpi. Kirjeldatakse rakenduse loomisel valitud

arhitektuurimustreid ning antakse hinnang valminud rakendusele. Samuti antakse ülevaade rakenduse võimalikest edasiarendustest.

1.2 Töö skoop

Käesoleva töö skoopi kuulub analüüs loodava rakenduse jaoks sobivaima arendustööde prioriteetide määramise tehnika ja tehnoloogiliste lahenduste leidmiseks. Samuti rakenduse esmase versiooni loomine, mis võimaldaks arendustöid planeerida ja valitud prioriteetide määramise tehnikat rakendada ning selle testimine.

Rakenduse koodibaas antakse üle Infotark AS-ile, kes on vastutav rakenduse serverisse paigaldamise ja edasise majutamise eest. Seega jääb antud töö skoobist välja majutamiseks sobiva serverilahenduse valik. Töö hõlmab andmebaasi valikut ja sobivuse analüüsi, ent rakenduse reaalne integratsioon lokaalselt andmebaasilt valitud baasile toimub peale rakenduse koodibaasi Infotark AS-ile üle andmist firma töötajate poolt.

1.3 Autori roll

Töö autor ei ole otseselt seotud ettevõttega Infotark AS, kelle jaoks arendustööde planeerimise ja prioriteetide määramise rakendus luuakse. Seetõttu konsulteerib autor sageli ettevõtte IT-juhiga kindlustamaks, et loodav rakendus vastaks nende poolt esitatud nõuetele.

Autori ülesandeks on esitatud nõudeid arvesse võttes valida sobilikud metoodikad ja tehnoloogiad ning luua rakenduse esmane versioon, mida Infotark AS edaspidi edasi arendada saab. Samuti on autori ülesandeks ka rakenduse funktsionaalsuse testimine ja dokumenteerimine.

2 Probleemi ülevaade

Probleem, mida bakalaureusetöös lahendatakse, tuleb ettevõttest Infotark AS, kus soovitakse leida arendustööde planeerimiseks ja prioriteetide määramiseks sobivat rakendust. Olemas on väga palju projektihalduse tarkvarasid ja nii mõnigi neist võimaldab anda sisestatud ülesannetele prioriteete. Ent rakendusi, mis võimaldavad prioriteete arvutada enda määratud parameetrite alusel, on vähe. Eriti kui on soov arvutada neid meeskonnana ehk nii, et koondhinnang kujuneks mitme individuaalse hinnangu põhjal. Suure osa rakenduste igakuised tasud nende kasutamiseks on suuremad kui firma soovib sellele kulutada. Samuti sisaldavad need funktsioone, mis firmale vajalikud ei ole ning mis lisavad pigem keerukust süsteemi kasutamisel.

Selgitamaks välja, kas on mõistlik ehitada uus rakendus või kasutada juba olemas olevat, on vajalik eksisteerivate lahenduste kaardistamine ja nende funktsionaalsuse analüüs. Käesoleva peatüki käigus selgitatakse välja, millised on nõuded rakendusele, mida soovitakse kasutama hakata ning analüüsitakse olemasolevaid rakendusi neist lähtuvalt.

2.1 Nõuded rakendusele

Käesolevas alapeatükis kirjeldatakse nõuete kogumise ja olulisuse määramise tehnikaid ning esitatakse välja selgitatud nõuded.

2.1.1 Nõuete kogumise tehnika

Üks levinumaid nõuete kogumise metoodikaid on intervjuu [1], mida otsustati ka käesolevas töös kasutada. Intervjuud jagunevad struktureeritud (järgitakse ettevalmistatud küsimusi) ja struktureerimata (avatud arutelu) intervjuudeks [1]. Nende vahele paigutub poolstruktureeritud intervjuu [1], mida selle töö raames nõuete kogumiseks kasutatakse, kuna see võimaldab saada vastused konkreetsetele ettevalmistatud küsimustele, ent jätab ruumi ka jätkuküsimusteks ja põhjalikemateks aruteludeks esile kerkinud teemade üle. Alternatiivina kaalutud küsimustik ei võimalda täpsustavaid küsimusi ja avatud arutelu. Igapäevaste tööprotsesside vaatlus nõuab rohkem aega ja keskendub enam olemasolevale kui sellele, mida saavutada tahetakse. Intervjuu

viidi läbi Infotark AS-i IT-juhiga, kes omab ülevaadet hetkeolukorrast ja töötajate vajadustest. Selle käigus selgunud nõuded on toodud välja peatükis 2.1.4.

2.1.2 Nõuete klassifitseerimine FURPS+ abil

Intervjuu käigus selgunud nõuded struktureeriti kasutajalugude (*user story*) tehnika abil. Kasutajalugu on mitteformaalne kirjeldus tarkvara omaduse kohta, mille eesmärk on sõnastada saadav kasu lõppkasutaja vaatest [2]. Kasutajalood järgivad struktuuri "<kellena> soovin <teha midagi selleks, et> <saavutada midagi>". Leitud nõuetele vastavad kasutajalood on toodud välja lisas 3. Kasutajalugude sõnastamine võimaldab keskenduda saadavale kasule ning jätab avatuks viisid, kuidas seda saavutada. Ühtlasi oli detailselt lahti kirjutatud kasutajalugude põhjal lihtsam rakendusele esitatavaid nõudeid kirja panna.

Nõuete klassifitseerimiseks valiti FURPS+ metoodika, kuna see suunab pöörama tähelepanu erinevatele kategooriatele ning võimaldab eristada funktsionaalseid ja mittefunktsionaalsed nõudeid. FURPS+ on Robert Grady poolt loodud nõuete klassifitseerimise metoodika, mis jaotab nõuded kategooriatesse – funktsionaalsus (*functionality*), kasutatavus (*usability*), töökindlus (*reliability*), jõudlus (*performance*) ja toetatavus (*supportability*) [3]. "+" kategooria sisaldab nõudeid disainile, teostusele, liidestusele teiste süsteemidega ja füüsilisele infrastruktuurile [3]. Klassifitseerimine FURPS+ metoodika alusel on toodud välja lisas 4. Olulisuse alusel grupeeritud nõuded on toodud välja peatükis 2.1.3.

2.1.3 Nõuete grupeerimine MoSCoW metoodika abil

Kogutud nõuded otsustati järjestada olulisuse põhjal tuvastamaks, millele rohkem rõhku pöörata ning mis ei ole otsitava lahenduse puhul esmatähtis. See hõlbustab olemasolevate tarkvarade võrdlust ning aitab piirata uue loodava rakenduse skoopi, kui see vajalikuks peaks osutuma. Nõuete grupeerimiseks olulisuse alusel valiti MoSCoW meetod, mis jagab nõuded nelja kategooriasse – kindlasti vajalik funktsionaalsus (M – *must have*), peaks olema (S – *should have*), võiks olla (C – *could have*) ja mida ei kavatseta vaadeldavas skoobis teostada (W – *won't have*) [4]. Selle kasuks otsustati, kuna see võimaldab selgitada välja olulisemad nõuded, millest ennekõike lähtuda ning kõige vähem tähtsad, mis saab vajadusel kõrvale jätta. Hindamine viidi läbi koos Infotark AS IT-juhiga. Grupeeritud nõuded on toodud välja tabelis 1.

Tabel 1. Nõuetele prioriteetide määramine MoSCoW metoodika abil.

Must have	**Should have**
• Kasutaja saab sisestada, muuta ja kustutada ülesandeid. • Kasutaja saab määrata töödele algus- ja lõppkuupäeva, teostajat ja staatust. • Kasutaja saab arvutada tööde prioriteete erinevate kriteeriumite alusel. • Töö prioriteet arvutatakse mitme inimese hinnangu põhjal. • Peakasutaja saab valida kasutajaid, kes saavad töid hinnata. • Peakasutaja saab valida tööde nimekirja, mida hindama hakatakse. • Peakasutaja saab määrata ja muuta perioodi, mil tööde hindamine kasutajate jaoks võimalik on. • Kasutaja saab näha teiste hindeid alles peale hindamisperioodi lõppu. • Kasutaja saab muuta oma hinnangut vastamise perioodi jooksul. • Peakasutaja saab registreerida uusi kasutajaid. • Kasutaja saab rakenduse veebilehel sisse logida ja parooli muuta. • Rakenduse kasutusvõimalused peavad olema dokumenteeritud. • Rakendus peab toetama versioneerimist • Rakendus peab olema võimeline toetama 50 paralleelset sessiooni.	• Kasutaja saab näha kirjete muutmise ajalugu (algus- ja lõppkuupäeva, staatuse, teostaja, töö kirjelduse muudatusi). • Kasutaja saab sisestatud töid näha graafilisel ajaskaalal. • Kasutaja saab valida perioodi, mille kohta ajaline graafik kuvatakse. • Erineva pikkusega tööd on ajalisel graafikul visuaalselt eristatavad.
Could have	**Won’t have**
• Kasutaja tuleb sisse logides suunata prioriteetide hindamise lehele kui on avatud hindamisi, mis on talle määratud. • Ajalisel graafikul töö peal klõpsates on võimalik näha selle detailvaadet. • Kasutaja saab näha prioriteetide arvutamise ajalugu (hindepunktide muudatusi avatud hindamisperioodi jooksul) kasutajate kaupa.	• Peab saama sisse logida ID-kaardiga, Mobiil-IDga ja Smart-IDga. • Peab toetama erinevaid keeli (eesti keel, inglise keel, vene keel). • Ajalisel graafikul saab töid lohistades muuta nende algus- ja lõppkuupäeva.

MoSCoW meetodit kasutades selgus, et üks kõige olulisemaks peetavaid funktsionaalsusi lisaks tööde sisestamisele ja muutmisele on prioriteetide arvutamise võimalus ning seda

just mitme kasutaja hinnangu põhjal. Soovitud ent mitte esmavajalikku kategooriasse kuuluvad tööde kuvamise võimalus graafilisel ajaskaalal ja kirjete muudatuste ajaloo nägemine. Käesoleva töö skoobis ei peetud vajalikuks erinevate keelte tuge ja rakendusse sisse logimise võimalust ID-kaardi, Mobiil-ID ja Smart-IDga.

2.2 Eksisteerivate lahenduste võrdlus

Võrdlemiseks valiti kümme rakendust, mis on märgitud kui parimad agiilsed projektihaldustarkvarad [5]. Neile lisati neli lahendust, mida Infotark AS-is kaalutud või ka juba kasutatud on (ClickUp, Microsoft Project, Jira, Redmine) ning kolm rakendust, mis on spetsialiseerunud rohkem töödele prioriteetide määramisele (Airfocus, Strategic Quadrant, Hygger). Võrdluse aluseks võeti esmavajalike nõuete kategooriast prioriteetide märkimise ja arvutamise võimalus ning järgmisest kategooriast tööde graafiline kuvamine ajalisel skaalal. Viimasesse kuulus ka kirjete ajaloo kuvamine, ent see otsustati kriteeriumitest välja jätta, kuna Infotark AS-i poolt hinnati selle olulisus võrreldes graafilise kuvaga väiksemaks ning leiti, et see ei ole nende jaoks otsustava kaaluga. Neljandaks kriteeriumiks valiti maksumus, kuna see määrab, kas rakendust on võimalik firmas kasutusele võtta või ei. Nimetatud funktsionaalsuste võrdlus on esitatud tabelis 2. Sümbolid + ja – näitavad, kas funktsionaalsus on olemas või ei ole ning +/– "tasuta" tulbas viitab, et on olemas nii tasuta kui tasuline versioon. Lühiajaline (enamasti üks kuu) tasuta prooviperiood ei ole selle alla arvatud.

Tabeli 2 põhjal on näha, et ülesannetele prioriteetide seadmise võimalus on olemas suuremal osal vaadeldud rakendustest, ent nende arvutamine mitme kriteeriumi põhjal on vaid kahel. Samuti oluliseks peetud tööde graafilist kuvamist ajaskaalal võimaldavad enamus. Mitmed lahendused pakuvad piiratud funktsionaalsusega tasuta versioone. Nutcache puhul ei sisalda tasuta versioon tööde kuvamist graafilisel ajaskaalal. Üks rakendus, mis sisaldab kogu soovitud funktsionaalsust on Airfocus, mille eest tuleb meeskonna puhul tasuda 39–79 dollarit kasutaja kohta [6], olles analüüsitud tarkvaradest kalleim ning ületab hinnapiiri, mida Infotark AS oleks nõus maksma. Sellest oluliselt odavam (7 dollarit kuus kasutaja kohta) on Hygger [7], milles samuti soovitud võimalused olemas on.

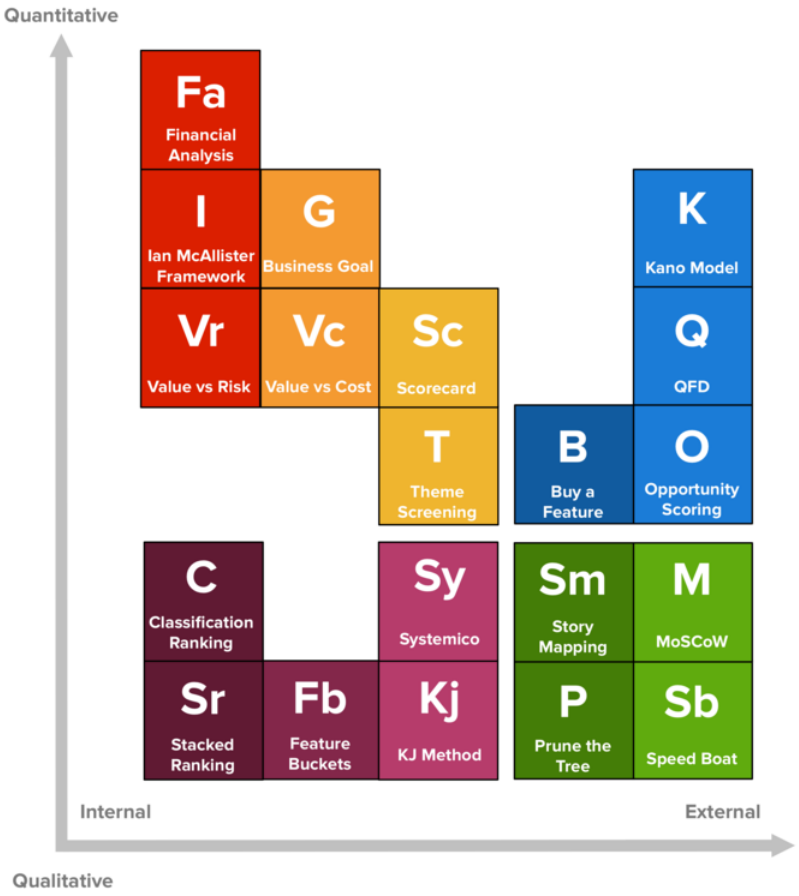
Tabel 2. Olemasolevate projektihaldustarkvarade funktsionaalsuse võrdlus.

Tarkvara	**Prioriteetide märkimine**	**Prioriteetide arvutamine**	**Graafiline kuva**	**Tasuta**
monday.com [8]	+	–	+	–
Smartsheet [9]	+	–	+	–
Clarizen [10]	+	–	+	–
Taiga [11]	–	–	–	+/–
Forecast [12]	–	–	+	–
Wrike [13]	–	–	+	+/–
Hubstaff [14]	–	–	+	+/–
ProjectManager [15]	+	–	+	+/–
Zoho Sprints [16]	+	–	+	+/–
Nutcache [17]	–	–	+/–	+/–
ClickUp [18]	+	–	+	+/–
Microsoft Project [19]	+	–	+	–
Jira [20]	+	–	+	+/–
Redmine [21]	+	–	+	+
Airfocus [22]	+	+	+	–
Strategic Quadrant [23]	+	+	–	–
Hygger [7]	+	+	+	–

Peale olemasolevate rakenduste võrdlust leiti siiski, et on otstarbekas uus sobiva funktsionaalsusega rakendus arendada, kuna sellega ei kaasne igakuiseid kulutusi ning see võimaldab valida ise sobivaim prioriteetide määramise tehnika. Ühtlasi saab loodavat rakendust hiljem edasi arendada soovitud suunas.

3 Prioriteetide määramise tehnika valik

Käesolevas peatükis antakse ülevaade prioriteetide määramise tehnikatest, mida kaaluti loodava rakenduse jaoks ning valitakse välja sobivaim lahendus. Metoodikate valikul lähtuti, et oleks esindatud nii sisemised, välised, kvalitatiivsed kui ka kvantitatiivsed tehnikad. Tehnikate paiknemine nimetatud skaaladel on toodud välja joonisel 1.



Joonis 1. Prioriteetide määramise metoodikad lähtudes kvalitatiivsuse-kvantitatiivsuse ja sisemise-välise skaalast [24]. QFD – *Quality function deployment*, kliendikeskne tootekavandamine. MoSCoW – *Must have, should have, could have, won't have* – nõuete olulisuse alusel grupeerimise metoodika.

Väliste tehnikate korral on lõppkasutajad rohkem kaasatud, sisemiste korral vähem [24]. Kvalitatiivsed metoodikad põhinevad rohkem eksperthinnangutel, kvantitatiivsed mõõdetavatel näitajatel [24]. Lisaks kaasati valikusse tehnika WSJF (*Weighted Shortest Job First*), kuna see võimaldab arvutada prioriteete erinevate kriteeriumite põhjal, mis

läheb kokku sissejuhatuses välja toodud algse ideega prioriteetide määramiseks. Samuti planeerimise pokker kui levinud tehnika meeskonnana tööde hindamiseks.

3.1 Kaalutud prioriteetide määramise tehnikad

Algav peatükk kirjeldab vaadeldud prioriteetide määramise tehnikaid lähtudes loodava rakenduse jaoks seatud nõuetest.

3.1.1 Kano mudel

Kano mudelit kasutatakse lõppkasutaja ootuste määramiseks jagades kavandatavad funktsionaalsused nelja kategooriasse – ühesuunalised (kliendi rahulolu kasvab lisanduva funktsionaalsuse kasvades), kohustuslikud (klient eeldab nende olemasolu), atraktiivsed (neid ei eeldata, aga põhjustavad suurt rahulolu) ja ükskõiksed (ei mõjuta kliendi rahulolu) [24]. Kliendi hinnangu saamiseks kasutatakse küsimustikke, mille põhjal leitakse, millisesse kategooriasse planeeritav funktsionaalsus paigutub. Reeglina lähtutakse tööde planeerimisel järjestusest kohustuslikud → ühesuunalised → atraktiivsed → ükskõiksed [24]. Kuna Kano mudel lähtub ennekõike kliendi hinnangutest, siis võivad ülesanded, mis on tehniliselt olulised, ent mille tulemus ei paista lõppkasutajale välja, põhjendamatult madala prioriteedi saada. Ühtlasi ei võimalda see tehnika ühe kategooria siseselt töödele prioriteete määrata.

3.1.2 Kliendikeskne tootekavandamine – QFD

QFD (*Quality Function Deployment)* tehnika jaoks on vaja tuvastada kliendi nõuded, anda neile kaalud sõltuvalt prioriteedist, panna kirja konkreetsed ülesanded nende saavutamiseks ning määrata nõuete ja ülesannete maatriksis ülesannetele punktid (1, 3 või 9) sõltuvalt sellest, kui tugevalt on ta seotud vastava nõudega [24]. Ülesannete lõppskoor arvutatakse prioriteetide kaalude ja üksikute punktide korrutiste summana, suurima tulemuse saavutanud töö saab suurima prioriteedi [24]. QFD puuduseks on eeldus, et kliendipoolsete nõuete prioriteedid on teada, ent see on osa probleemist, mida käesolevas töös lahendada püütakse.

3.1.3 Võimaluste hindamine

Võimaluste hindamise tehnika (*Opportunity Scoring*) põhineb kliendi hinnangul konkreetsete vajaduste olulisuse osas ning sellel, kui suurel määral see vajadus praegu rahuldatud on [24]. Arvutamiseks kasutatakse järgmist valemit [24]:

$$Olulisus + maksimum\ (\ Olulisus - Rahulolu,\ 0\ ) = Võimalus$$

Mida suurem võimaluse väärtus, seda prioriteetsemad on selle vajaduse rahuldamiseks vajalikud ülesanded. Selle tehnikaga on võimalik prioriteete määrata ennekõike üldisematele kliendipoolsetele soovidele ning lõpptulemusele, mida nad soovivad näha. Ühe konkreetse eesmärgi siseselt lahku löödud väiksemate ülesannete jaoks aga ei ole see parim metoodika.

3.1.4 Osta funktsionaalsus

Funktsionaalsuse ostmise *(Buy a Feature)* puhul on tegu mängulise tehnikaga, mille käigus antakse kliendile kokkulepitud hulk "raha" ja esitatakse nimekiri funktsionaalsustest, mille seast ta peab valida, milliseid ta "osta" soovib [24]. Igale funktsionaalsusele on antud hind mingi kriteeriumi alusel (keerukus, pingutus, mida see nõuab, hind või muu) ning kliendile antav "raha" moodustab kolmandiku kuni pool kõikide toodete koguhinnast [24]. See tehnika aitab selgitada välja kliendi jaoks kõige olulisemad funktsionaalsused, ent nõuab sisendiks tootele määratud hinda, mille ebaadekvaatne määramine võib lõpptulemust mõjutada.

3.1.5 MoSCoW

MoSCoW meetodit on tutvustatud peatükis 2.1.4, kus seda kasutati loodava rakenduse jaoks kirja pandud nõuete liigitamiseks olulisuse alusel. See on hea meetod ülesannete kategoriseerimiseks tuvastamaks neid, mida on kindlasti vaja teha, mida võiks teha ja mis võib esmajärgus kõrvale jätta, ent ühe kategooria siseselt selle abil prioriteete määrata ei saa.

3.1.6 Väärtus vs risk

Väärtuse ja riski põhine metoodika (*Value vs Risk*) määrab ülesannete tegemise järjestuse põhimõttel, et esmalt tuleks teostada kõrge riski ja väärtusega tööd, siis kõrge riski ja madala väärtusega tööd, viimasena madala riski ja väärtusega tööd ning vältida neid, mille puhul mõlemad näitajad on madalad [24]. Riski all võib käsitleda nii ajalist, rahalist

kui ka funktsionaalset riski [24]. Selle tehnika miinuseks on sarnaselt mitme teisega see, et on võimalik töid erinevate prioriteetidega gruppi paigutada, ent grupi siseselt on tööd sama prioriteediga.

3.1.7 Väärtus vs hind

Sarnaselt eelneva tehnikaga võtab väärtus vs hind (*Value vs Cost*) arvesse töö teostamisel loodavat väärtust ning selleks kulunud hinda (see ei pea olema rahaline, vaid võib olla ka näiteks keerukuse hinnang) [24]. Siinkohal aga tekib probleem suure väärtuse ja kuluga tööde ning väikese väärtuse ja kuluga tööde korral, kuna nende suhe on sarnane. Seega positsioneeruvad nad prioriteetide reas samasse piirkonda ning võib juhtuda, et teostatakse rohkem väheolulisi madala kuluga töid ning vähem suurema väärtuse, ent kõrgema kuluga töid. Seega peaks lisaks leidma sobiva tasakaalu omakorda ka nende vahel.

3.1.8 Tulemuste kaart

Tulemuste kaardi tehnika (*Scorecard*) käigus valitakse kriteeriumid, mille abil funktsionaalsusi hindama hakatakse ning määratakse neile kaalud, seejärel koostatakse maatriks funktsionaalsustest ja kriteeriumitest ning arvutatakse neile summaarsed skoorid [24]. See tehnika võimaldab määrata kõigile ülesannetele täpse järjestuse (mitte neid ainult kategoriseerida). Seevastu on kriteeriumite valik ja neile kaalude määramine küllaltki subjektiivne ning võib olla juba kallutatud selles suunas, mida nende määraja oluliseks peab.

3.1.9 Teemade sõelumine

Teemade sõelumine (*Theme Screening*) sarnaneb tulemuste kaardile, ent kriteeriumitele kaalude määramise asemel valitakse üks ülesanne lähtejooneks ning hinnatakse teisi töid selle suhtes [24]. See eemaldab eelnevast meetodist kaalude määramise faasi subjektiivse aspekti.

3.1.10 Funktsionaalsuste grupeerimine

Metoodika looja Adam Nashi järgi jagatakse funktsionaalsuste grupeerimise (*Feature Buckets*) korral kavandatavad tööd kolme gruppi - toote ja äriliste näitajate osas olulisi muudatusi kaasa toovad tööd (*metric movers*), kliendi soovid ja boonusülesanded, mida otseselt nõutud pole [25]. Sellele on lisatud veel strateegiliste tööde grupp, mis koondab

tuleviku vaates olulisi ülesandeid [24]. Suurima prioriteediga on esimene kategooria, kui oluline on leida tasakaal, valides töid kõigist. Sarnaselt mitme teise tehnikaga ei võimalda see ülesannetele prioriteete määrata ühe kategooria siseselt.

3.1.11 Kaalutud lühim töö esimesena – WSJF

WSJF (*Weighted Shortest Job First*) on tehnika, mille abil määratakse töödele prioriteete majanduslikust kasust lähtudes [24]. Nimelt saadakse vastav näitaja jagades tööga viivitamise hind (*cost of delay*) töö suuruse ehk selle teostamiseks kuluva ajaga [26]:

$$WSJF = \frac{\text{viivitamise hind}}{t\ddot{o}\ddot{o}\ suurus}$$

Viivitamise hind saadakse summeerides suhtelised väärtused, mis on määratud vastava töö teostamisest saadavale ärilisele kasule, ajakriitilisusele ning sellele, kuivõrd maandatakse sellega riske või luuakse võimalusi tulevikuks [26]:

Viivitamise hind = äriline väärtus + ajakriitilisus + riski vähendamine või võimalused

Viivitamise hinna komponentide väärtused ja töö suurus tuleks määrata kasutades Fibonacci numbreid (1, 2, 3, 5, 8, 13, 20), kuna see võimaldab paremini väljendada määramatust, mis suuremad tööd endaga kaasa toovad [26]. Oluline ei ole konkreetsed väärtused, vaid nende omavaheline suhe.

3.1.12 Planeerimise pokker

Planeerimise pokker (*Planning Poker*) on populaarne agiilne hindamistehnika, mida kasutatakse nii tööde suuruse kui prioriteetide määramiseks [27]. Igale vaadeldavale ülesandele annavad kõik liikmed hinde Fibonacci numbri näol ning suurte erinevuste korral räägitakse hinnangute põhjused läbi ja hääletakse uuesti [27]. See tehnika võimaldab meeskonnana ühiselt hinnanguid anda, ent arutelu käigus võib domineerivate inimeste arvamus peale jääda. Kuna hinded koosnevad vaid ühest komponendist ega too erinevaid aspekte selgelt välja, ei pruugi need alati tegelikkust peegeldada. Kombineerituna teiste tehnikatega võib anda häid tulemusi.

3.2 Loodava rakenduse jaoks valitud prioriteetide määramise tehnika

Põhjalikumast analüüsist jäeti kõrvale meetodid, mis põhinesid liialt personaalsetel või grupiviisilise diskussiooni käigus selguvatel hinnangutel, mida püütakse otsitava lahendusega asendada. Samuti need, mis võimaldavad otsustada, milliseid töid kõrvale

jätta, ent mis ei aita alles jäänutele prioriteete seada. Ühtlasi ei peetud sobivaks ka ainult finantsanalüüsil põhinevaid tehnikaid.

Väliste tehnikate skaala suuremas otsas paiknevate ehk suure kliendi osalusega tehnikate rakendamine Infotark AS-i puhul on keeruline, kuna sisend töödeks tuleb mitmest allikast – süsteemide välistelt kasutajatelt (ennekõike veebipoe pool), seadusest tulenevatest nõuetest ja asutusest seestpoolt. Seega muutub tööde lõikes klient, kelle vajadustest ja soovidest lähtuma peaks ning nende vahelised skaalad ei pruugi üheselt võrreldavad olla.

Mitmed tehnikad (Kano mudel, MoSCoW, väärtus vs risk, väärtus vs hind, funktsionaalsuste grupeerimine) võimaldavad töid grupeerida prioriteetide alusel, ent ühe grupi siseselt neid järjestada ei aita. Käesoleva probleemi lahendamiseks sobivaimateks metoodikateks on teemade sõelumine ja WSJF, kuna need võimaldavad mitme kriteeriumi põhjal arvutada summaarseid hindeid, mis annab võimaluse võtta hindamisel arvesse planeeritavate tööde erinevaid aspekte. Teemade sõelumise miinuseks võrreldes WSJF-ga on see, et kriteeriumid, mis on tööde hindamisel aluseks, tuleb ise määrata ning ebaõnnestunud valiku korral ei pruugi tulemus reaalset olukorda kajastada. Samuti tuleb alati valida üks ülesanne, mis võetakse mõõdupuuks teiste hindamisel. Kui selleks osutub väga kõrge prioriteediga töö, siis võib tekkida olukord, et kõiki teisi hinnatakse sellest madalamaks enamike kriteeriumite puhul ning nad saavad samasugused koondhinded. Sellisel juhul tekib jälle probleem nende omavahelise järjestamisega. WSJF-i puhul on tegu etteantud kriteeriumitega tehnikaga, mis võimaldab tuvastada töid, mille teostamine toob suurima majandusliku kasu. Nelja erineva kriteeriumi ja Fibonacci numbrite kasutamine peaks tagama tööde koondhinde piisava erinevuse, et ei tekiks suuremaid sama hindega gruppe, mida on raske sisemiselt järjestada. Eriti kui tööde lõplik prioriteedi hinnang kujuneb mitme meeskonnaliikme WSJF väärtuse keskmisena. Seega otsustati loodava rakenduse jaoks kasutada WSJF tehnikat.

4 Tehnoloogiate valik

Käesolev peatükk annab ülevaate, millist serveri-poolset tehnoloogiat ja programmeerimiskeelt, andmebaasi ning kasutajaliidese keelt ja raamistikku loodava rakenduse jaoks kasutatakse ning analüüsitakse nende sobivust.

4.1 Serveri-poolse tehnoloogia ja keele sobivuse analüüs

Infotark AS-i sooviks on kasutada serveri poolel .NET platvormi tehnoloogiat ja keeleks C#. Lähtetingimuse põhjuseks on asutuse senine kallutatus Microsofti toodete kasutamise suunas, mida soovitakse jätkata. Lisaks on asutusesiseselt olemas kompetents ja soov nimetatud tehnoloogia ja keelega rakenduse edasi arendamiseks, mis näiteks Java raamistike puhul puudub. Kuna käesoleva töö raames luuakse rakenduse esmane versioon, mis vajab ilmselt edasi arendamist, siis on oluline nende soove arvesse võtta, kui ei ole põhjust miks seda mitte teha. Seetõttu veendutakse .NET tehnoloogiate sobivuses kavandatava rakenduse jaoks ning valitakse neist loodava rakenduse jaoks sobivaim.

Varasemalt on olnud .NET tehnoloogiate puhul võimalik veebirakenduste loomiseks valida .NET Frameworki ja .NET Core vahel. Lisaks on olemas ka Xamarin [28], aga see on mõeldud Androidi ja iOS-i rakenduste programmeerimiseks, seega seda käesoleval juhul ei kaaluta. .NET Frameworki saab kasutada nii töölaua rakenduste kui veebirakenduste loomiseks, ent see on mõeldud ainult Windowsi operatsioonisüsteemiga serverite jaoks [28]. .NET Frameworki soovitatakse kasutada juba selle tehnoloogia peale üles ehitatud rakenduste edasi arendamiseks või juhul, kui soovitakse kasutada tehnoloogiaid, mida .NET Core ei toeta [29]. Kõige uuem .NET Frameworki versioon on 4.8 [28], millele ka edaspidi tuge pakutakse.

Kuna ei ole olulist põhjust kasutada .NET Frameworki, siis on sobivam valik .NET 5, mis on 2020. aasta novembris välja tulnud järgmine versioon peale .NET Core 3.1 [30]. .NET 5 väljendab Microsofti soovi minna edasi ühe ühtse .NET tootega ning koondab enda alla nii .NET Frameworki, .NET Core kui Xamarini [30]. Sellega loodud rakendusi on

võimalik panna tööle nii Windowsi, Linuxi kui macOS-i peal ning enamik uutest .NET arendustest lisandub just sinna [28]. .NET 5 puhul ei ole tegu pikaajalist tuge omava versiooniga (LTS), milleks saab olema 2021. aasta novembris kättesaadavaks tehtav .NET 6.0 [30]. Siiski oleks mõistlik valida kõige uuem versioon, et oleks võimalik kasutada sellega lisandunud funktsionaalsust ning et hilisem versiooni uuendamine lihtsam oleks. Seega on uue rakenduse loomiseks parim valik just .NET 5 tehnoloogia.

.NET tehnoloogiad võimaldavad kasutada programmeerimiskeelena C#, F# ja VB.Net-i [28]. Kuna töö autor omab neist kogemust ainult C#-ga, siis ei ole mõistlik hakata käesoleva töö loomiseks ette nähtud piiratud ajalistes raamides hakata õppima uut keelt, mis välistab F# ja VB.Net-i. Seega on loodava rakenduse serveri poolse osa programmeerimiskeeleks C#, mis oli soovitud ka Infotark AS-i poolt.

.NET 5 kasutades on veebirakenduste loomiseks mõeldud raamistik ASP.NET Core, mille abil saab luua rakendusi kasutades rakenduse sisest mudel-vaade-kontroller (MVC – *Model-View-Controller*) mustrit, veebipõhist API-t (*Application Programming Interface* - rakendustarkvara liides) või mõlemaid paralleelselt [31]. ASP.NET Core võimaldab veebipõhise API dokumenteerimist Swaggeri abil kasutades selleks Swashbuckle või NSwag teeki [32]. Võimalik on ka API versioneerimine Microsoft.AspNetCore.Mvc.Versioning teegi abil [33], mis on lisaks dokumentatsioonile üheks oluliseks nõudeks. Ülejäänud nõuded tulenevad äri poole loogikast ning ei sea tehnoloogia valikule täiendavaid piiranguid. Seega, kuna kavandataval veebirakendusel ei ole nõudeid, mida ei oleks võimalik .NET 5 ja ASP.NET Core raamistikku kasutades täita, siis kasutatakse just neid rakenduse loomisel.

4.2 Andmebaasi valik

Infotark AS-i lähtetingimus loodava rakenduse jaoks on, et kasutatav andmebaas peab olema Microsoft SQL Server (MSSQL Server) või MySQL, eelistades esimest. See tuleneb asjaolust, et firmas kasutatakse juba neid andmebaase ning ilma tungiva põhjuseta ei olda huvitatud võtmast kasutusele uut andmebaasi. MSSQL Serveri valik läheks kokku asutuse sooviga kasutada suures osas Microsofti tooteid. Seega otsustati uue rakenduse jaoks vaadelda lähemalt MSSQL Serveri ja MySQLi omadusi leidmaks, kumb neist sobilikum oleks ning teised andmebaasid jäeti valikust kõrvale.

MSSQL Server ja MySQL on mõlemad relatsioonilised andmebaasid, mis põhinevad relatsioonilisel mudelil. See tähendab, et andmeid hoitakse tabelites, mille kirjed on välisvõtmete abil seotud teistes tabelites olevate andmetega. Relatsioonilisel mudelil põhinevad andmebaasid võeti kasutusele 1970ndatel ning on tänaseni väga laialdaselt kasutusel [34]. Nii MSSQL Server kui MySQL on populaarsed, suure jõudlusega, kiired ja skaleeruvad andmebaasid, mis sobivad nii väikeste kui suurte projektide jaoks [35].

MySQL on avatud lähtekoodiga andmebaas, mis loodi 1995. aastal ning mida on võimalik kasutada nii Windowsi, Linuxi kui ka macOS-i serveris [35]. Andmebaas kasutab SQL/PSM (*Structured Query Language / Persistent Stored Modules*) keele laiendit [36]. MySQL-i haldab ja kasutajatuge pakub Oracle [37]. Võimalik on kasutada tasuta versiooni, ent sellel on piiratud funktsionaalsus ja puudub kasutajatugi, ärilistel eesmärkidel kasutamiseks on mõeldud tasulised versioonid [38].

MSSQL loodi 1989. aastal Microsofti poolt ning kuigi see oli algselt mõeldud kasutamiseks ainult Windowsi serverites. siis tänaseks on tugi olemas ka Linuxi ja macOS-i jaoks [35]. Keele laiendina on kasutusel Transact-SQL [39]. MSSQL-i kasutamine nõuab litsentsi ning on kallim kui MySQL [35]. Samas on väiksemate rakenduste jaoks sobilik kuni 10 GB andmemahtu võimaldav MSSQL Express, mis on tasuta [40].

Kuna nii .NET kui MSSQL on Microsofti tooted, siis on nende koos kasutamine väga levinud ja integratsioon lihtne [35]. Vähemalt esimeses järgus ei ületa loodava rakenduse andmemaht 10 GB, seega valitakse rakenduse andmebaasiks MSSQL Express. Tulevikus on võimalik vajadusel andmebaas tasulise versiooni vastu välja vahetada.

MSSQL andmebaasi integreerimine rakendusse toimub peale koodibaasi üleandmist Infotark AS-i poolt, kuna töö autoril puudub ligipääs asutuse andmebaasidele ja serveritele. Andmebaasi vahetamine on .NET rakendustes võimalik muutes “Startup.cs” klassi konfiguratsiooni ning andmebaasi ühenduse parameetreid. Arenduse ajal kasutatakse lokaalset SQLite andmebaasi selle mugavuse tõttu. MSSQL lokaalset andmebaasi on võimalik kasutada vaid Windowsi masinate peal SQL Server Express Editioniga [41]. Siiski valideeriti rakenduse ühilduvus MSSQL baasiga migratsioonide (käskude jada andmebaasi ja tabelite loomiseks [42]) genereerimisega MSSQL konfiguratsiooniga ning korrigeeriti andmemudelit vastavalt kuvatud hoiatustele.

4.3 Kliendi-poolse tehnoloogia valik

Kliendi-poolse tehnoloogia valik hõlmab endast nii üldise disainimusti valikut loodava rakenduse jaoks kui ka konkreetsemat keele ja raamistiku valikut kasutajaliidese loomiseks. Järgnevates alapeatükkides on toodud välja kaalutud variandid ning põhjendatud langetatud otsuseid.

4.3.1 Disainimustri valik

Kavandatava rakenduse arhitektuur peab võimaldama tulevikus soovi korral erinevate osade hõlpsat muutmist või välja vahetamist. Selle tagab MVC disainimustri kasutamine. Mudeli osa koondab enda alla andmeobjektidega seotud loogika, vaade on nende kliendi poolne kuva ning kontroller määrab, kuidas rakendus reageerib kasutaja sisendi peale ehk on vahendaja mudeli ja kontrolleri vahel [43]. ASP.NET Core raamistik võimaldab MVC mustrit rakendada kahel viisil – kasutades ASP.NET Core MVC raamistiku kontrollereid ning C# ja Razor süntaksit kasutavaid vaateid või veebipõhiseid API kontrollereid ja eraldiseisvat klientrakendust.

Esimese variandi puhul on võimalik luua vaated kasutades Razor Pagesit [44], mille vaated saab genereerida rakenduse siseselt. Teine võimalus on luua rakendus RESTful teenuste loomiseks mõeldud API kontrolleritega, mis võimaldavad suhelda klientrakendustega, mis paiknevad mudelist ja kontrolleritest eraldi. 2000. aastal kasutusele võetud mõiste REST (*REpresentational State Transfer*) on tarkvara arhitektuuri stiil ning seda kasutavaid teenuseid nimetatakse RESTful teenusteks [45]. REST põhimõtete järgi käsitletakse vahendatavat informatsiooni ressurssidena ning kõik päringud on olekuta – iga päring peab sisaldama kogu informatsiooni, mida on vaja selle töötlemiseks ja vastuse tagastamiseks [45].

Kavandatava rakenduse loomiseks sobivad mõlemad eespool mainitud lahendused, ent veebipõhise API kasutamine võimaldab eraldada serveripoolse rakenduse klientrakendusest ning neid on võimalik üksteisest sõltumata edasi arenda või välja vahetada. Samuti annab see võimaluse panna serveripoolset rakendust kasutama mitut klienti paralleelselt. ASP.NET Core MVC-ga saab luua mitmelehelise rakenduse (MPA – *multiple-page application*), ent veebi API-de kasutamine annab võimaluse luua üheleheline rakendus (SPA – *single-page application*). Viimase suureks eeliseks on kiirus, kuna suur osa ressursse nagu HTML (*Hyper Text Markup Language*,

hüperteksti märgistuskeel), CSS (*Cascading Style Sheets,* kujunduskeel) ja skriptid loetakse vahemällu ning uuesti laetakse vaid päritud andmeid [46]. Kõike seda arvesse võttes otsustati loodava rakenduse jaoks kasutada ASP.NET Core veebipõhise API loomiseks mõeldud funktsionaalsust.

4.3.2 Klientrakenduse keele ja raamistiku valik

Klientrakenduse jaoks on valida PHP ja JavaScripti/TypeScripti vahel. PHP on üldotstarbeline keel, mis sobib nii ees- kui tagarakenduse loomiseks [47]. Sarnaselt on ka JavaScripti võimalik kasutada nii serveri- kui klientrakenduse poolel, ent väga levinud on selle kasutamine just veebirakenduste kasutajaliideste loomiseks [48]. TypeScript on loodud JavaScripti põhjal lisades sellele tüübikindluse [49]. Rakenduse kliendi poolse osa loomiseks on mõistlik kasutada raamistikke, millesse on suur hulk põhifunktsionaalsust juba sisse ehitatud. Tabelis 3 on toodud välja autori kogemus PHP ja JavaScripti keele ja raamistikega ning keelte õppimise keerukus. Valitud skaala õppimise keerukuse jaoks on “väike”, ”keskmine” ja ”suur”, kogemuse puhul lisandub valik “puudub”. Nii PHP kui JavaScripti õppimise keerukus on madal, ent kuna autori kogemus PHP-ga on väike ja raamistikega puudub, siis otsustati seda mitte kasutada. Tüübitud keele kasutamine muudab loodava rakenduse vähem veaaltiks ning hõlbustab edasist arendamist ja refaktoreerimist. Seega otsustati JavaScripti asemel kasutada TypeScripti.

Tabel 3. Klientrakenduse poolsete keelte võrdlus.

Keel	**Õppimise keerukus**	**Kogemus keelega**	**Kogemus raamistikega**
PHP	Väike [50]	Väike	Puudub
JavaScript / TypeScript	Väike [51]	Keskmine	Keskmine

Klientrakenduse raamistike osas kaasati valikusse viis 2020. aastal populaarseimaks valitud JavaScripti raamistikku [52] ning lisaks üks, millega autoril varasem kogemus olemas (Aurelia). Võrdlus nende õppimise keerukuse ja autori senise kogemuse aspektist on toodud tabelis 4.

Tabel 4. Klientrakenduse poolsete raamistike võrdlus.

Raamistik	**Õppimise keerukus**	**Kogemus raamistikuga**
React	Suur [52]	Väike
Angular	Suur [52]	Puudub
Ember	Suur [52]	Puudub
Vue	Väike [53] / Keskmine [52]	Keskmine
Backbone	Keskmine [52]	Puudub
Aurelia	Väike [54]	Keskmine

Nii Reacti, Angulari kui Emberi puhul on õppimise keerukus suur ning kuna varasem kokkupuude nendega puudub või on väike, siis ei ole otstarbekas neid kasutada. Suurim kogemus on autoril Vue ja Aureliaga, mille õppimise keerukus on ühtlasi võrreldes teiste raamistikega väiksem, seega tehakse lõplik valik nende kahe vahel. Vue on seni olnud populaarsem kui Aurelia ning sellest tulenevalt on ka Aurelia dokumentatsioon ja pluginate valik kehvem kui Vuel [55]. Seega kasutatakse loodava rakenduse kliendipoolse osa jaoks Vued.

4.4 Integreeritud arenduskeskkonna ja koodihalduskeskkonna valik

Sobiva arenduskeskkonna valik sõltub rakenduse loomiseks kasutatavast tehnoloogiast ja programmeerimiskeelest ning see võib olla erinev serveripoolse ja klientrakenduse jaoks. Koodi versioneerimise ja haldamise jaoks on alati mõistlik kasutada selleks mõeldud koodihalduskeskkondi. Järgnevalt on välja toodud põhjendused rakenduse loomiseks kasutatud integreeritud arenduskeskkonna ja koodihalduskeskkonna valikuks.

4.4.1 Integreeritud arenduskeskkonna valik

2021. aasta kolm populaarseimat integreeritud arenduskeskkonda C# kirjutamiseks paremusjärjestuses vastavalt kasutajate poolt saadud häältele on JetBrains Rider, Visual Studio ja Visual Studio Code (VS Code) [56]. JetBrains Rideri kasutamine on tasuline [56], ent tudengitel on võimalik saada tasuta litsentse [57]. Visual Studiol on olemas nii tasuline kui piiratud funktsionaalsusega tasuta versioon, VS Code kasutamine on tasuta [56]. JetBrains Rider ja VS Code on võimalik kasutada nii Windowsi, macOS kui Linuxi operatsioonisüsteemiga masinate peal, Visual Studiot Linuxi peal võimalik kasutada ei

ole [56]. Töö autori peamisel arvutil on macOS operatsioonisüsteem, seega sobivad kõik nimetatud arenduskeskkonnad. Kuna autor on seni C# rakenduste arendamiseks kasutanud JetBrains Riderit ning kasutab tööl igapäevasel Java rakenduste jaoks samuti JetBrainsi toodet IntelliJ, siis otsustati Riderit kasutada ka loodava rakenduse jaoks. JetBrains võimaldab erinevate toodete puhul kasutada samasid klahvikombinatsioone kiirkäskude jaoks ning samasugune struktuur võimaldab mugavalt kahe keskkonna vahel liikuda.

Klientrakenduse loomiseks valitud Vue raamistiku jaoks soovitatakse arenduskeskkonnana VS Code-i, JetBrainsi WebStrormi/PhpStrormi, CodeSandBoxi, Atomit või Sublime Texti. Nagu eelnevalt mainitud, on tudengitel võimalik Jetbrainsi tooteid tasuta kasutada, teised loetletud arenduskeskkonnad on tasuta. Autor omab varasemat kogemust VS Code ja JetBrainsi toodetega. Kuna Vue kogukond on esimese valikuna välja toonud VS Code kui töövahendi, mis on kiire ja mitmekülgne ning võimaldab kasutada suurt hulka pluginaid, siis langetati valik selle kasuks.

4.4.2 Koodihalduskeskkonna valik

Käesoleval aastal on viieks parimaks versioonihaldustarkvaraks nimetatud Git, CVS, Apache Subversion (SVN), Mercurial ja Monotone [58]. Esikohal välja toodud Git on ühtlasi ainus, millega autoril seni kogemus on ning selle kasutamine on tasuta. Seega valiti versioonihalduseks Git.

Arenduse ajal hoitakse rakenduse lähtekoodi töö autori personaalses hoidlas ning hiljem tõstetakse see ümber Infotark AS-i poolt valitud koodihalduskeskkonda. Arenduse ajal ei ole koodihalduse jaoks spetsiifilisi nõudeid, mis piiraks sobiva tarkvara valikut, seega valiti kolme keskkonna vahel, mida autor varasemalt juba kasutanud oli – GitLab, GitHub ja Bitbucket. GitHub ja GitLab on 2020. aastal valitud kaheks parimaks koodihalduskeskkonnaks [59] ning kuna töö autor on suuremat osa kooliga seotud projekte hallanud GitLabis, siis otsustati ka käesoleva töö raames loodava rakenduse jaoks kasutada GitLabi.

4.5 Tehnoloogiate valiku kokkuvõte

Infotark AS-i poolt esitatud eelistus oli serveri poolel kasutada .NET platvormi tehnoloogiat ja keelena C#. Analüüsi käigus leiti, et need on loodava rakenduse jaoks

sobilikud ning valituks osutus .NET 5 tehnoloogia. Andmebaasina eelistati asutuse Infotark AS poolt MSSQL Serverit või MySQLi ning kuna MSSQL Serveril on olemas ka piiratud mahuga tasuta versioon MSSQL Express, siis langetati valik selle kasuks. Arhitektuuriliselt kasutatakse MVC (*Model-View-Controller*) disainimustrit ning klientrakendusega käib suhtlus ASP.NET Core veebipõhiste API-de kaudu. Kliendipoolse tehnoloogia osas valiti keeleks TypeScript ning võrreldi erinevaid raamistikke, mis seda keelt toetavad ning neist sobivaimaks osutus Vue. Integreeritud arenduskeskkonnana kasutati serveripoolse rakenduse jaoks JetBrains Riderit, klientrakenduse jaoks VS Code-i. Koodihalduskeskkonnanaks valiti GitLab.

5 Lahenduse realiseerimine

Käesolev peatükk tutvustab samme, mis rakenduse struktuuri ja disaini väljatöötamiseks tehti ning annab ülevaate valminud rakendusest. Ühtlasi kajastatakse turvalisust, testimist ja dokumentatsiooni puudutavaid aspekte.

5.1 Rakenduse disain

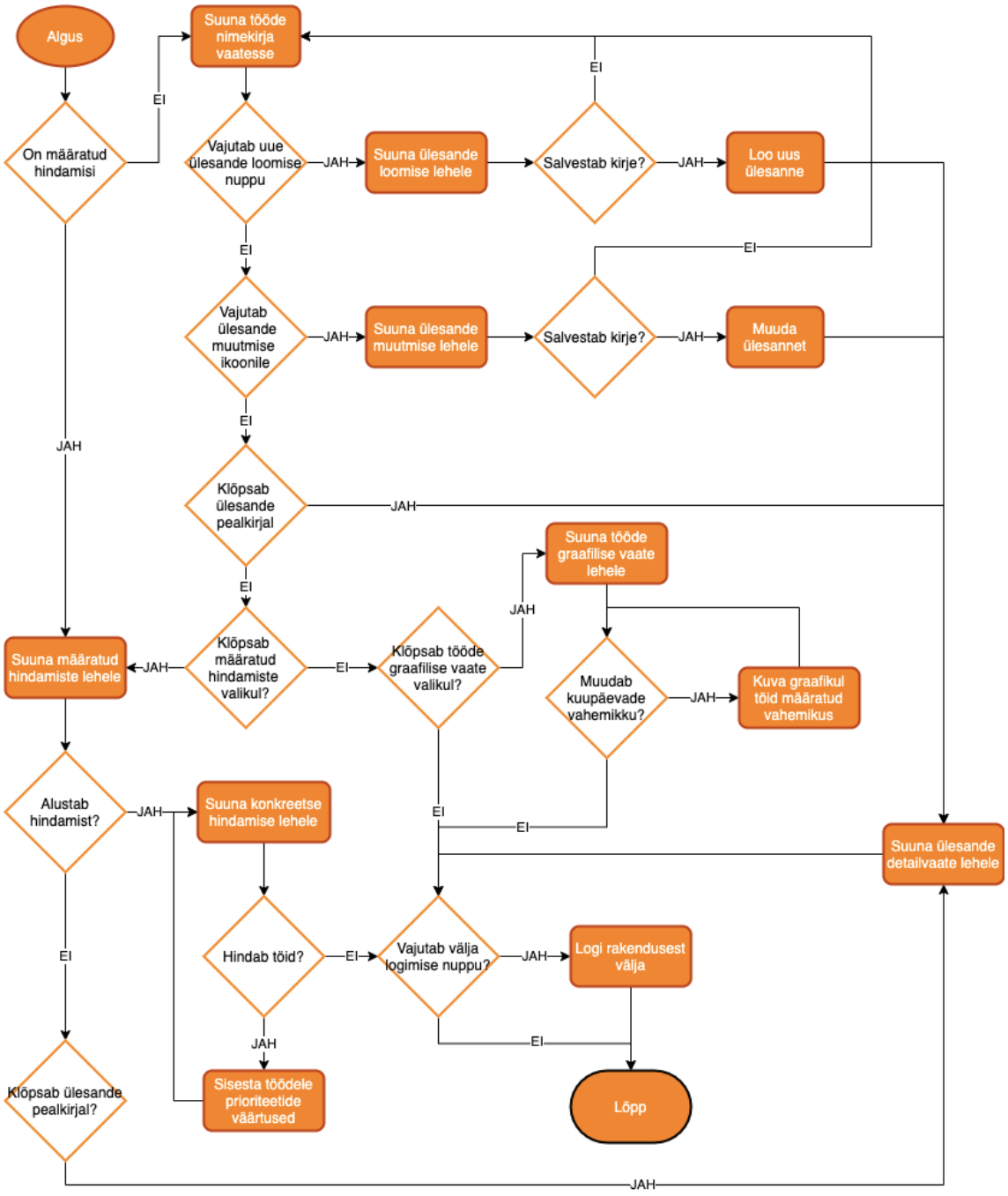
Rakenduse planeerimisel on oluline mõelda läbi funktsionaalsus, mida see võimaldama peab, millised saavad olema peamised kasutajaliidese vaated ning millise struktuuriga saab olema loodav rakendus arhitektuurilise poole pealt. Kõigi nende kujutamine graafiliselt annab hea pidepunkti, millised peaksid olema loodavad olemid ja nende vahelised suhted. Käesolevas alapeatükis ongi vastavad etapid läbi viidud.

5.1.1 Voodiagrammid

Üks võimalus loodava rakenduse funktsionaalsuse kaardistamiseks on protsesside modelleerimine graafilisel kujul. Protsesside mudel kirjeldab tegevuste järjestikust voogu ning annab ülevaate, millised sammud järgnevad kasutajate erinevate valikute korral. Mõned levinud tehnikatest protsesside modelleerimiseks on voodiagrammid (*flowcharts*), väärtusevoo kaardistamine (*value stream mapping*), andmevoo diagrammid (*data flow diagrams*), ühtne modelleerimiskeel (*unified modelling language* – UML) ning äriprotsesside mudel ja graafiline märgistik (*business process model and notation* – BPMN) [1].

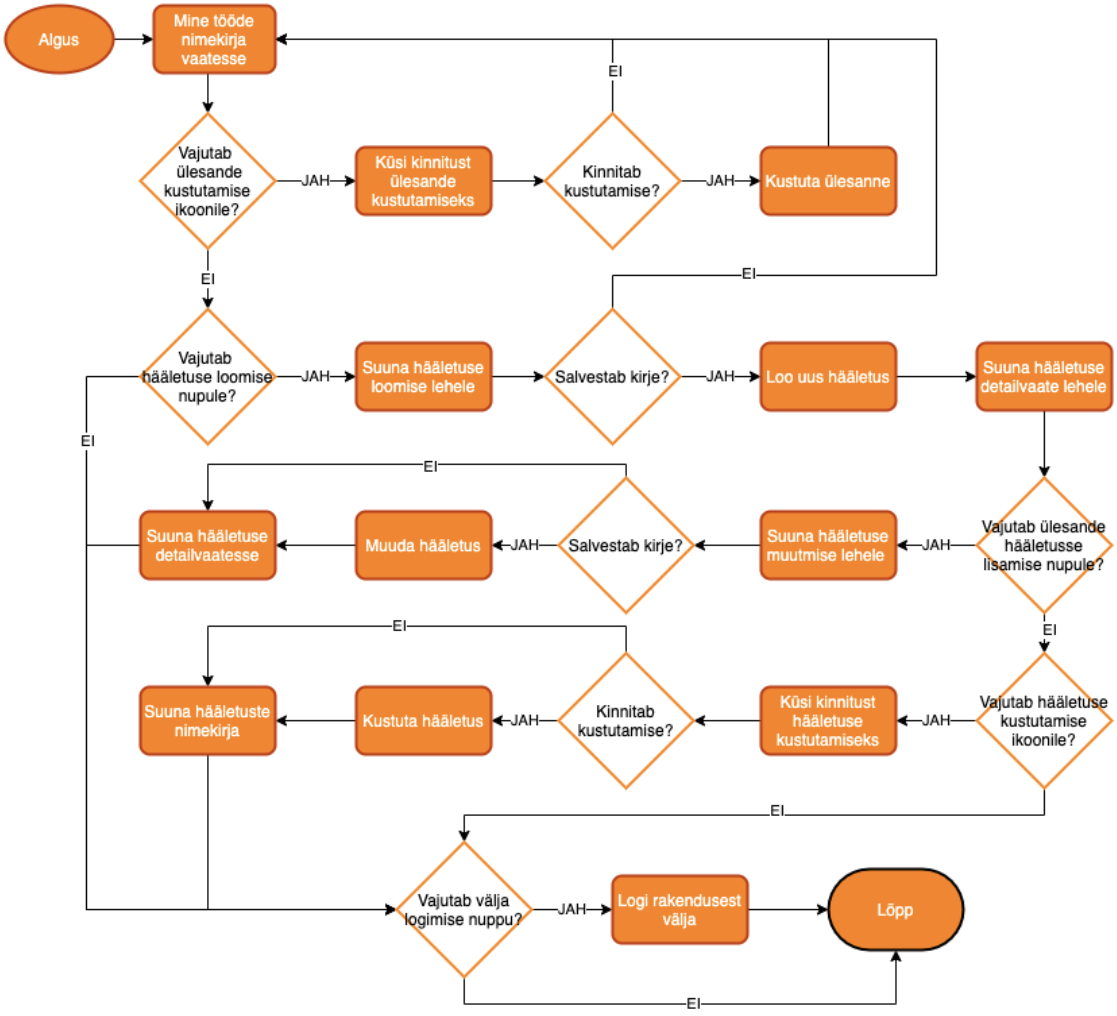
Käesolevas töös otsustati protsesside modelleerimiseks kasutada voodiagramme, kuna see ei nõua suurt detailsust ning annab ülevaate protsessidest tehnilistesse detailidesse laskumata, mida peeti antud rakenduse kavandamisel piisavaks. Diagrammide loomiseks kasutati rakendust Draw.io, kuna seda on mugav kasutada ning see on tasuta.

Suurem osa rakenduse funktsionaalsusest on ühine sõltumata kasutajal olevatest rollidest (milleks algselt on planeeritud tavakasutaja ning peakasutaja ehk administraator). Jagatud tegevuste voodiagramm on toodud välja joonisel 2.



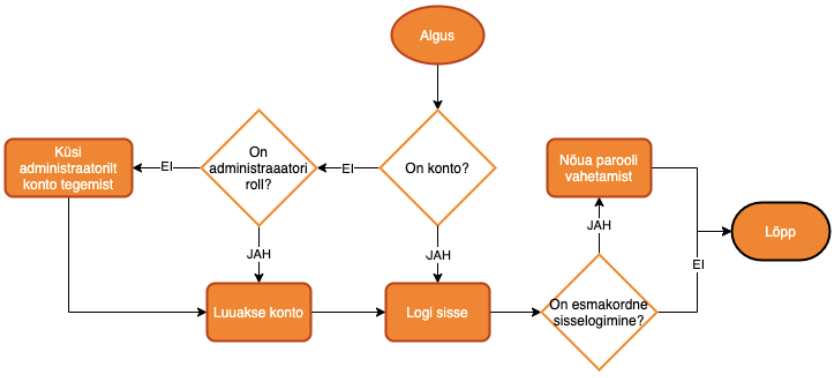
Joonis 2. Tavakasutaja õigustega tegevuste voodiagramm (allikas: autori koostatud).

Tavakasutajal ei ole lubatud uusi hääletusi luua ega kustutada, nad saavad vaid neile määratud hääletustel osaleda. Ühtlasi on tööde kustutamine võimalik ainult peakasutaja puhul. Voodiagrammi tegevustega, mida saab sooritada vaid peakasutaja, on kujutatud joonisel 3.



Joonis 3. Peakasutaja õigustega tegevuste voodiagramm (allikas: autori koostatud).

Joonisel 4 on kujutatud kasutaja loomise ja sisse logimise protsessi. Uue kasutaja loomise õigus on ainult peakasutajal (administraatori roll), kes määrab kasutajale ka algse parooli. Esmasel sisse logimisel nõuab rakendus parooli vahetamist uue vastu.



Joonis 4. Sisse logimise voodiagramm (allikas: autori koostatud).

Kahe erineva voodiagrammi loomise asemel erinevates õigustes kasutajate jaoks oleks olnud võimalik kasutada ka ujumisradade (*swimlane*) tehnikat. Ujumisrada on horisontaalne või vertikaalne ala voodiagrammil, mis eraldab neid tegevusi, mida on võimalik sooritada ainult konkreetse rolli poolt [18]. Kuna aga sel juhul oleks diagramm väga suureks läinud ning selle mahutamine ühele joonisele keeruline, otsustati kasutada eraldi diagramme.

5.1.2 Veebilehe disain

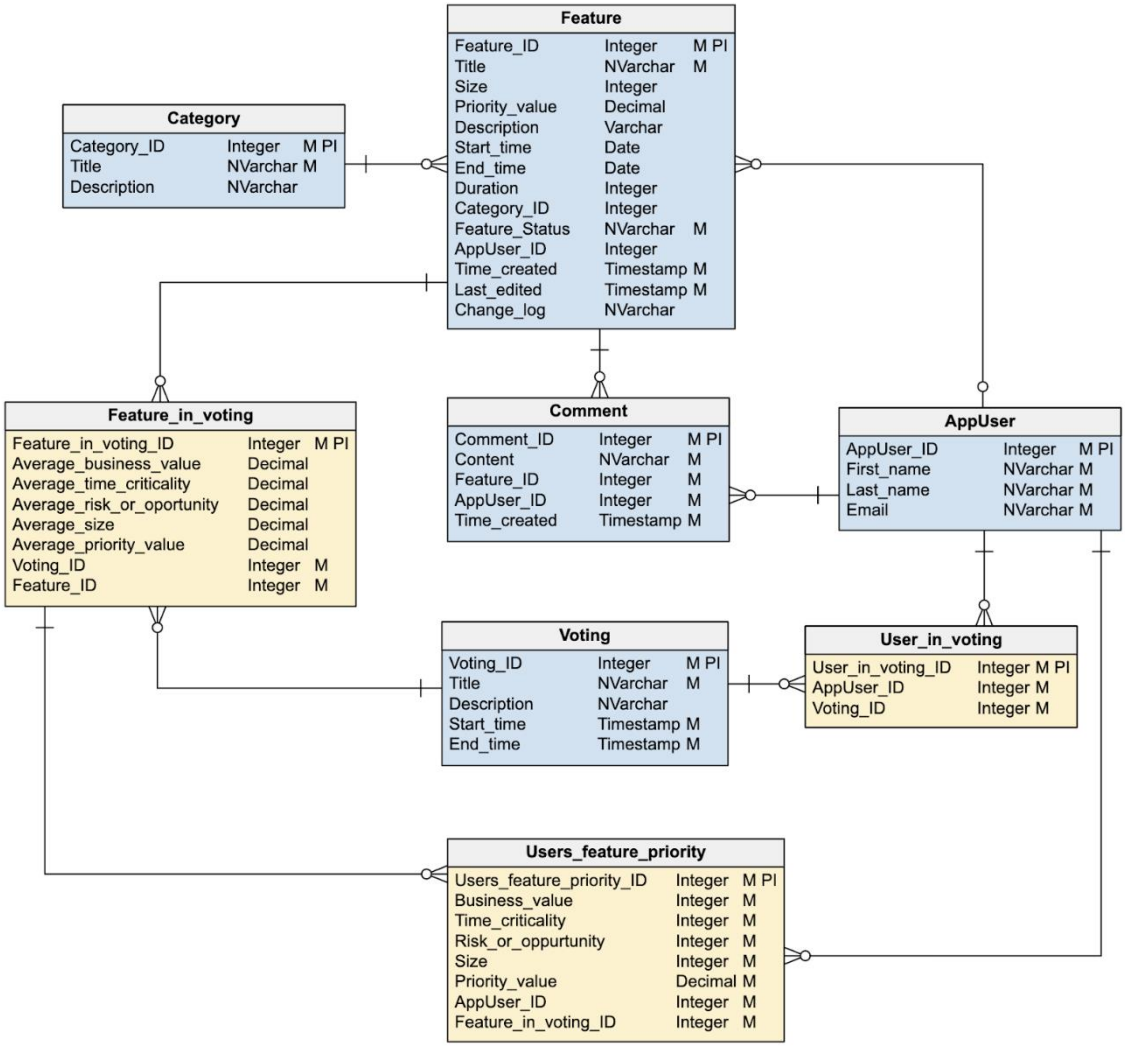
Üks viise loodava rakenduse kavandamiseks ja kliendi tagasiside saamiseks on luua (mittetöötav) prototüüp rakenduse peamiste vaadetega, mis peegeldavad olulisemat funktsionaalsust. Selleks kasutati programmi Figma, mis on kasutajakogemuse disainimiseks mõeldud töövahend. Disainitav kujundus ei vasta üheselt loodavale rakendusele, ent annab piisava ülevaate kavandatavast kasutajakogemusest. Prototüübi peamised vaated on toodud välja lisas 5.

Rakenduse vaadete kujundamine võimaldab loodav funktsionaalsus põhjalikult läbi mõelda ning annab sisendi olemi-suhte diagrammi loomiseks, mida on kirjeldatud järgmises alapeatükis.

5.1.3 Olemi-suhte diagramm

Andmebaasi struktuuri kujutamiseks loodi kavandatava rakenduse olemi-suhte diagramm, mida on kujutatud joonisel 5. Sinisega on toodud välja iseseisvalt eksisteerivad olemid, kollasega vahetabelid mitu-mitmele seose realiseerimiseks. Töö staatuse välja jaoks (*Feature_Status*) kasutatakse andmebaasis tekstilist formaati, ent programmikoodis määratakse selle väärtus klassifikaatorina (*enum – enumerated type* andmetüüp). Kaaluti ka eraldi teatmik-tüüpi tabeli loomist staatuse jaoks, ent kuna tegu on andmeväljaga, mis tuleks enne rakenduse kasutusele võtta ära määrata ning mis ei tohiks sageli muutuda, siis ei ole eraldi tabeli loomine mõistlik lahendus.

Diagrammi loomiseks valiti Vertabelo tarkvara, kuna see võimaldab lisada skeemile atribuute, esmast võtit ja kohustuslikkuse märget, seda on mugav kasutada ning see on tudengitele tasuta. Joonisel on märgitud esmaste võtmete tüübina *Integer* mitte *Guid (Globally Unique Identifier)*, mida tegelikult kasutatakse, kuna Vertabelo andmetüüpides ei ole määratud andmetüüpi *Guid*.



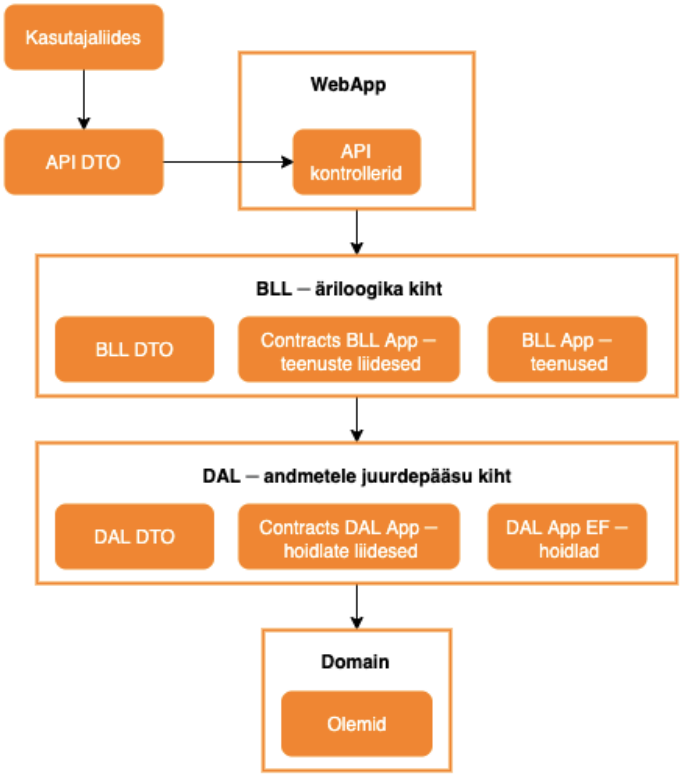
Joonis 5. Olemi-suhte diagramm (allikas: autori koostatud).

Kasutajate ja rollide haldamiseks kasutatakse ASP.NET Core individuaalse autentimise funktsionaalsusega loodud projektiga kaasa tulevat funktsionaalsust [60], laiendades seda enda poolt lisatud AppUser ja AppRole klassile. Joonisel 5 ei ole kujutatud tabeleid, mis andmebaasi kasutajate haldamiseks automaatselt luuakse, vaid ainult AppUser, kuna see on ainus, millele rakenduse spetsiifilisi välju lisatakse.

5.1.4 Serveri poolse rakenduse arhitektuur

Rakendus luuakse kihilise struktuuriga ja kasutades liideseid, mis võimaldab tulevikus teha muudatusi ühe taseme lõikes nii, et puuduks vajadus teisi ümber kirjutada. Ühtlasi hõlbustab see rakenduse eri osade sõltumatut testimist. Nendeks kihtideks on domeen,

andmetele juurdepääsu kiht, äriloogika kiht ja API kontrollerid. Rakenduse struktuuri graafiline kirjeldus on toodud välja joonisel 6.



Joonis 6. Rakenduse struktuur (allikas: autori koostatud).

- Domain – olemi-suhte diagrammis kirjeldatud olemitele vastavad objektid koos viidetega omavahelistele seostele. Entity FrameWork Core loob vastavate klasside põhjal andmebaasi struktuuri.
- Andmetele juurdepääsu kiht (*DAL – data access layer*):
    - Contracts.DAL.App – liidesed hoidlate (*repository*) jaoks, mille kaudu käib suhtlus andmebaasiga kasutades domeeni objekte.
    - DAL.App.EF – hoidlad ja muu realiseeritud loogika andmebaasi päringute tegemiseks Entity Frameworki funktsionaalsust kasutades.
    - DAL.App.DTO – andmeedastusobjektid (DTO – *data transfer object*) andmetele juurdepääsu kihi info edastamiseks.
- Äriloogika kiht (*BLL – business logic layer*):
    - Contracts.BLL.App – liidesed teenuste jaoks, mis vahendavaid päringuid kontrollerite ja hoidlate vahel.

    - BLL.App – liideste põhjal realiseeritud teenused, mis on mõeldud äriloogika rakendamiseks.
    - BLL.App.DTO – andmeedastusobjektid äriloogika kihi info edastamiseks.
- ApiControllers – veebipõhised API kontrollerid, mis vahendavad päringuid klientrakenduse ja äriloogika kihi vahel.
- API.DTO – andmeedastusobjektid, mida kasutatakse kontrollerites API päringute vahendamiseks.

Rakenduses kasutatakse autori poolt varasemalt loodud baasklasse, mis on avalikult kättesaadavaks tehtud kasutades NuGeti paketihaldurit. NuGet on Microsofti paketihaldussüsteem, mis sisaldab üle 100 000 paketi (kompileeritud kood) ning kuhu on kõigil võimalus oma pakette lisada [61]. Sellisel viisil olemasolevate baasklasside kasutamine lihtsustab rakenduse struktuuri ja vähendab koodi hulka, mida vaja hallata on. Paketid on loodud kasutades geneerikuid, seega toetavad need nii erinevaid esmase võtme tüüpe kui ka erinevat tüüpi objekte.

Äriloogika ja andmebaasi vaheliste muudatuste jälgimiseks ja rakendamiseks ühe päringu jooksul kasutatakse tööühiku mustrit (UoW – *Unit of Work*). UoW hoiab infot selle kohta, milliseid andmebaasi puudutavad muudatusi on päringu jooksul tehtud ning sooritab vajalikud tegevused andmete salvestamiseks pärast muudatuste tegemist [62].

5.2 Turvalisus

Ligipääs rakendusele võimaldatakse ainult sisse loginud kasutajatele. Vältimaks seda, et igal inimesel oleks võimalus end kasutajaks teha, on uute kasutajate loomine võimalik ainult administraatori rolliga kasutajate puhul. Autentimiseks ja autoriseerimiseks kasutatakse ASP.NET Core individuaalse autentimise valikuga loodava rakendusega kaasa tulevat funktsionaalsust. Kasutajaõiguseid hallatakse rollide baasil.

Autentimiseks kasutatakse JSON (*JavaScript Object Notation*) formaadis veebitõendit (*JSON Web Token* – JWT) HS512 algoritmiga. JWT on standard, mis määrab kuidas salavõtmega allkirjastatud JSON formaadis andmeid turvaliselt edastada [63]. HS512 algoritmi puhul on kasutusel HMAC (*hash-based message authentication code*) info tervikluse kontroll ja SHA-512 räsialgoritm salavõtme räsi arvutamiseks [64]. Sisse logimisel genereeritakse JWT, mis serveri poolelt kliendile vastusena tagasi saadetakse

ning kuni välja logimiseni pannakse kliendi poolsetele päringutele JWT kaasa. JWT sisaldab muuhulgas infot kasutaja rollide kohta, mille alusel rakenduse serveripoolses osas määratakse kasutaja ligipääs ressurssidele ja õigus sooritatavatele tegevustele.

ASP.NET Core 5.0 võimaldab lisaks autentimisele ja autoriseerimisele kasutada suurel hulgal funktsionaalsust rakenduse turvaliseks muutmiseks. Üks neist on saladuste (näiteks andmebaasi kasutajanime ja parooli) haldus. Kui rakenduse arenduse ajal kasutatakse lokaalset andmebaasi (mida käesolevas töös faasis ka tehti), siis võib andmebaasi külge ühendamise info tulla failist appsettings.json. Ent reaalse andmebaasi külge ühendades võimaldab näiteks tööriist ASP.NET Core Secret Manager tundliku info projekti lähtekoodist eraldada [65]. Produktsiooni keskkonnas peab saladusi alati hoidma koodist eraldi krüpteeritud kujul.

Uue rakenduse loomisel seadistatakse vaikimisi vahevara HTTP (*hypertext transfer protocol*) päringute ümbersuunamiseks HTTPS (*hypertext transfer protocol secure*) peale [66]. HTTPS on meetod andmete edastamiseks veebis krüpteeritud kujul [67], seega peaks turvaliseks andmeedastuseks alati eelistama seda HTTP kasutamisele.

ASP.NET Core võimaldab SQL (*Structured Query Language,* andmebaasi päringukeel) päringute tegemiseks kasutada Entity Framework Core LINQ (*Language Integrated Query*) päringukeelt, millesse on sisse ehitatud sisestatud parameetrite valideerimine SQL süstimise rünnakute vältimiseks [68].

Kuna loodava rakenduse puhul ei ole tegemist väga tundlikku infot või delikaatseid isikuandmeid käsitleva rakendusega, siis piisab ASP.NET Core sisseehitatud turvalisuse tehnoloogiatest ning lisameetmeid rakendada vaja ei ole.

5.3 Testimine

Peamine rakenduse testimine arenduse käigus toimub manuaalselt, kuna tegu on funktsionaalsuse poolest küllaltki väikese ja mitte väga keerulise äriloogikaga rakendusega. Iga lisandunud funktsionaalsusega kontrollitakse selle toimimist ning veendutakse varasemalt loodud üksuste korrektses toimimises koos lisatud mooduliga. Sellise mustri rakendamiseks arendatakse tagarakendust ja klientrakendust paralleelselt, testides jooksvalt mõlemal pool loodud funktsionaalsust kui nende vahelist integratsiooni.

Lisaks manuaalsele testimisele kasutatakse üksusteste veendumaks teenuste kihi loogika korrektses toimimises. .NET rakenduste testimiseks on Microsofti dokumentatsioonis toodud välja raamistikud xUnit, NUnit ja MSTest. Kuna autoril varasem kogemus .NET rakenduste testimiseks mõeldud raamistikega puudub ning seetõttu ka isiklik eelistus puudub, siis valiti testide kirjutamiseks xUnit kui neist kõige uuem tehnoloogia [69].

Üksustestide idee on kontrollida konkreetse mooduli toimimist sõltumata integratsioonidest teiste rakenduse osadega. Seetõttu kasutatakse testides teenustega integreeritud klasside (hoidla, tööühik) jaoks makette, mille vastused päringutele on ette antud. Makettide loomiseks kasutatakse raamistikku Moq, mis on valitud populaarseimaks makettide loomise vahendiks, mis NuGeti paketina saadaval on [70].

5.4 Dokumentatsioon ja verisoneerimine

Kasutajatele rakenduse põhifunktsionaalsuse kirjeldamiseks sisaldab rakendus lehekülge, kus on kirjeldatud kasutajagruppide põhiselt võimalikke tegevusi ja tööde prioriteetide arvutamise eripärasid. Rakendusliidese dokumenteerimiseks Swaggeri abil kasutati Swashbuckle teeki, mis võimaldab rakenduse koodi lisatud XML (*Extensible Markup Language,* laiendatav märgistuskeel) kommentaaride põhjal genereerida objektide ja API kontrollerite meetodite kirjeldusi [32]. Swaggeri kasutajaliidese abil saab kompaktse ülevaate rakenduse serveri poolse osa kasutatavusest ning võimaldab teha päringuid selle testimiseks ning ühtlasi reaalsete andmete pärimiseks.

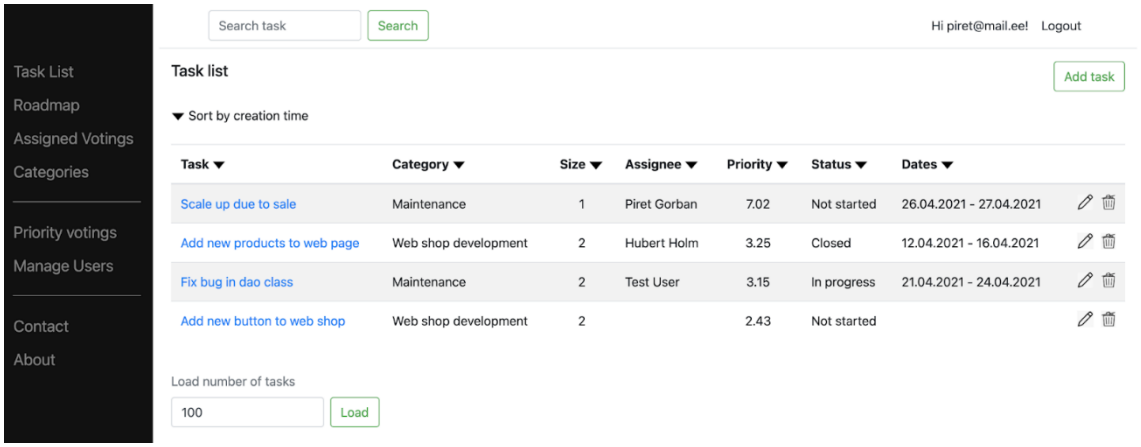
Rakendusliidese versioneerimiseks kasutati Microsoft.AspNetCore.Mvc.Versioning teeki. See võimaldab lisada uusi liideste versioone varasemaid integratsioone lõhkumata.

5.5 Loodud funktsionaalsus ja hinnang rakendusele

Peatükis 2.1.4 MoSCow metoodika alusel kui kindlasti nõutud funktsionaalsusest said rakendatud kõik. Erisus tekkis kasutajaõigustes – kuigi algselt leiti, et kasutaja saab töid sisestada, muuta ja kustutada, otsustati viimase õigus määrata vaid administraatorile. Kasutajale jääb võimalus ebavajalik töö sulgeda ning säilib ajalugu lisatud töödest.

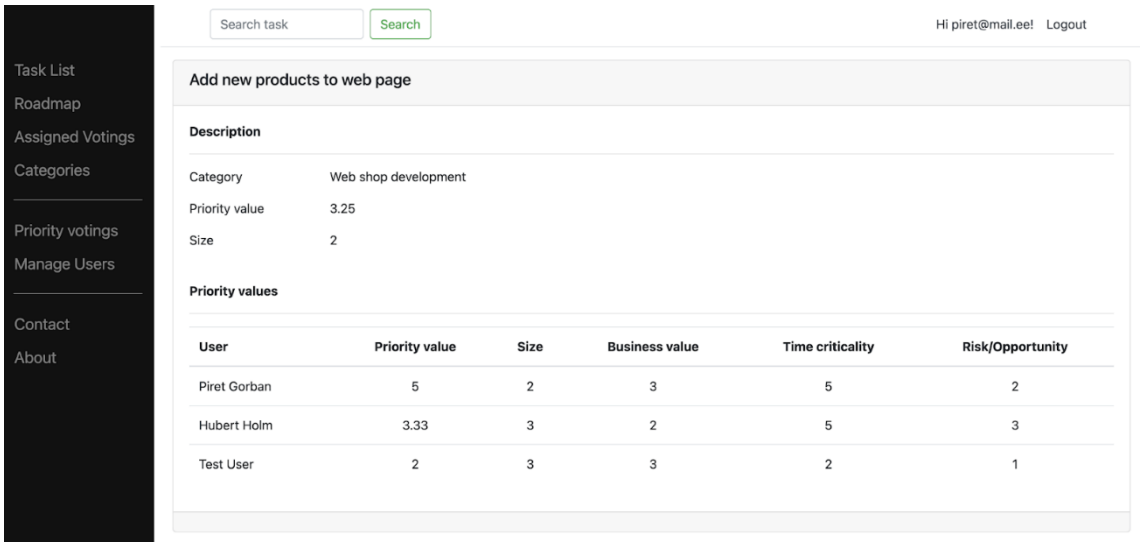
Rakendus võimaldab töid lisada, muuta, kustutada, pealkirja või kirjelduse põhjal otsida ja nimekirja vaates erinevate väljade põhjal sorteerida. Tööde nimekirja vaade on

kujutatud joonisel 7. Menüüs näha olev keskmine sektsioon hääletuste ja kasutajate haldamise lehtedega kuvatakse ainult administraatori õigustes kasutajale.



Joonis 7. Tööde vaade nimekirjana.

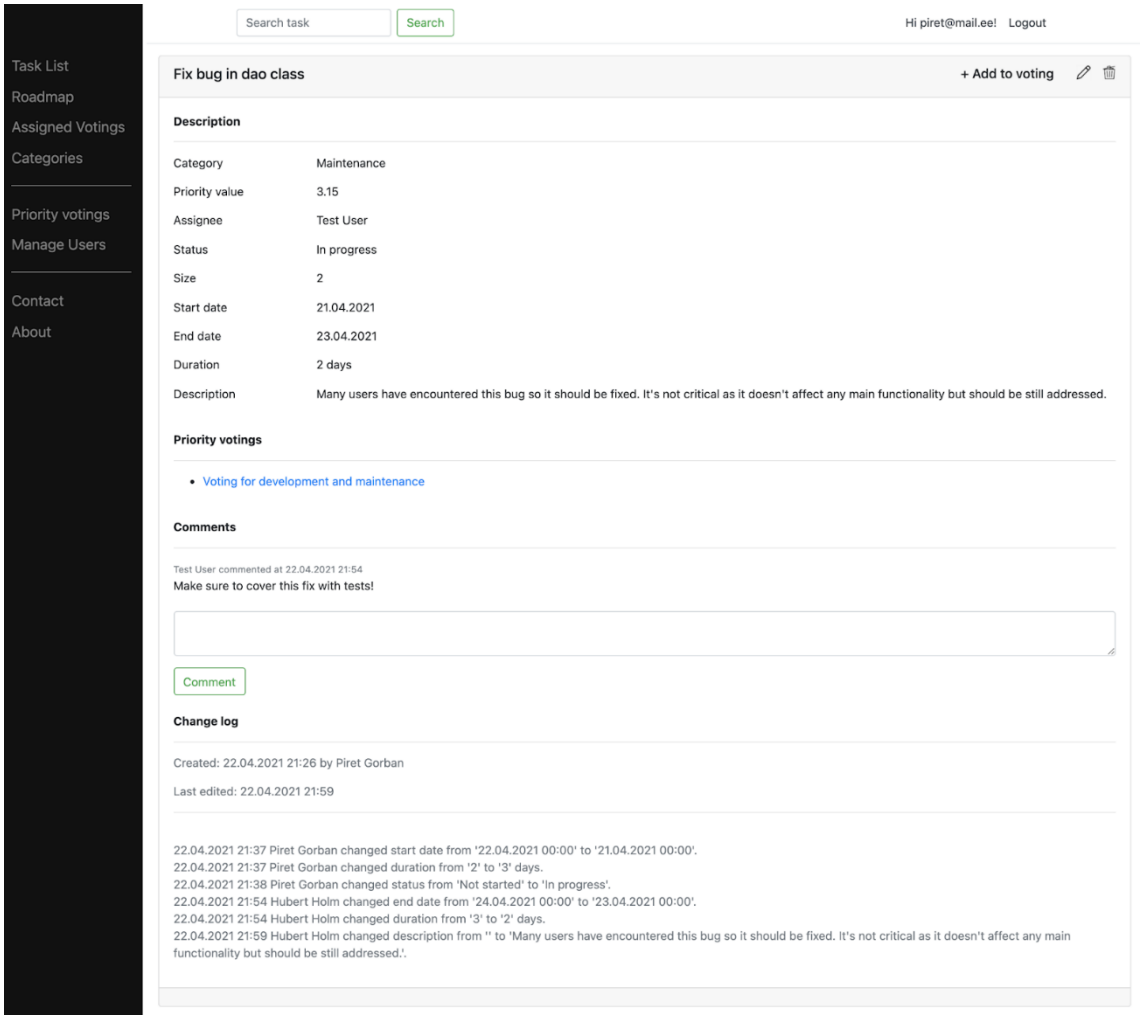
Töö suurus ja prioriteedi väärtus arvutatakse kasutades WSJF metoodikat kõigi kasutajate poolt määratud väärtuste keskmistena. Hääletamisel on soovitatav kasutada Fibonacci numbreid (vt peatükk 3.1.11). Prioriteedid kuvatakse tööde juurde alles siis, kui hääletus lõppenud on, et kasutajad üksteise hinnanguid ei mõjutaks. Kriteeriumite väärtuseid kasutajate põhiselt on samuti võimalik näha peale hääletuse lõppu. Nimetatud vaade on toodud välja joonisel 8. Kui kasutaja on oma hinnanguid muutnud, läheb arvesse viimane.



Joonis 8. Tööde prioriteetide hääletuse tulemused.

Kui kasutajale on määratud prioriteetide hääletusi ning mõni neist on avatud staatuses, siis suunatakse kasutaja peale sisse logimist talle määratud hääletuste lehele. See tagab, et kasutajal ei jääks märkamata, kui ta hääletusse lisatakse.

Peaks olema (*should have*) nõuetest said samuti kõik rakendatud. Töö detailvaates on võimalik näha, kes millal ja mida on muutnud ning lisada kommentaare. Ühtlasi kuvatakse millistesse prioriteetide hääletustesse töö on lisatud (prioriteete ühe töö jaoks on võimalik mitu korda arvutada). Kirjeldatud funktsionaalsus on näha joonisel 9.

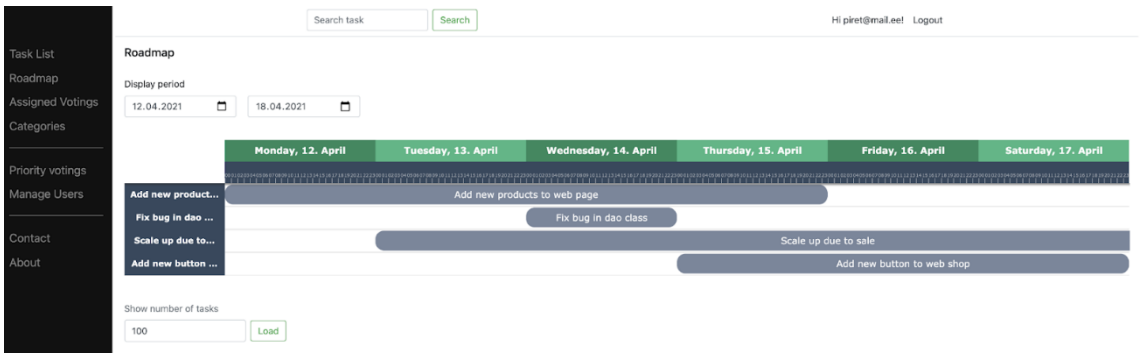


Joonis 9. Töö detailvaade koos muudatuste ajalooga.

MoSCoW võiks olla (*could have*) nõudena oli kirjas prioriteetide arvutamise ajaloo kuvamine kasutaja ja kriteeriumite kaupa ent rakenduse loomise käigus otsustati see funktsionaalsus kõrvale jätta. See oleks lisanud liigset keerukust ning sellest tulev kasu

rahuldab pigem uudishimu kui toodab reaalset väärtust. Ühtlasi tunnevad kasutajad end vabamalt kriteeriumitele väärtuseid andes kui teavad, et neid on võimalik ümber mõeldes muuta ilma, et sellest jälg maha jääks.

Peaks olema ja võiks olla osade vahel jagunenud tööde graafilisel ajaskaalal kuvamise nõuded said kõik ellu viidud. Kõik tööd, millel on algus- ja lõppkuupäev määratud, kuvatakse graafikul ning konkreetsel tööl klõpsates on võimalik liikuda selle detailvaatesse. Graafiku koostamiseks kasutati Vue jaoks mõeldud teeki vue-ganttastic [75], kuna seda on väga lihtne kasutada ning sisaldab kogu soovitud funktsionaalsust. Tööde graafiline vaade on toodud välja joonisel 10. Mitte teostatava (*won't have*) nõudena on MoSCoW tabelis kirjas tööde algus- lõppkuupäeva muutmine töid graafikul lohistada, ent vue-ganttastic võimaldab ka selle funktsiooni lisada, kui selleks edasiste arenduste käigus soov on.



Joonis 10. Tööde graafiline kuva ajaskaalal.

Kasutajate loomine ja õiguste lisamine toimub administraatori(te) poolt. Kasutaja loomisel määrab administraator parooli ning esmasel sisse logimisel nõutakse kasutajalt selle muutmist. Kui kasutaja parooli unustab, saab administraator talle uue parooli määrata ning järgmisel sisse logimisel nõutakse taaskord selle muutmist.

Veebiaadressid rakenduse koodile ligipääsemiseks GitLabi keskkonnas on toodud välja lisas 6. Peale rakenduse üleandmist Infotark AS-ile tõstetakse kood ümber nende poolt valitud versioonihalduskeskkonda.

Kokkuvõtlikult võib öelda, et püstitatud eesmärk sai saavutatud ning kasutatav rakendus loodud. Oluliseks peetud funktsionaalsusest ei jäänud midagi rakendusse lisamata.

Suurimaks väljakutseks oli prioriteetide arvutuse teostus, kuna hääletuses võib olla mitmeid ülesandeid, kasutajaid ja töid võib lisada mitmesse hääletusse, kasutajate hindeid peab olema hiljem võimalik eristada ning töö prioriteedi arvutamisel tuleb kasutada vaid viimase hääletuse tulemusi. See moodustas andmebaasi tabelite vahel sõltuvuste jadasid, mis muutis andmete pärimise kohati keeruliseks.

Uue kogemusena sai autor tutvuda .NET jaoks mõeldud testimise ja makettide kasutamise raamistikuga. Senine kogemus on piirdunud Java raamistikega. Ühtlasi puudus varasem kokkupuude klientrakenduse jaoks mõeldud graafikute koostamist võimaldavate teekidega ning oli meeldiv üllatus, et seda on võimalik üsna lihtsalt teostada.

Valminud rakendust tutvustati Infotark AS-i IT-juhile, kelle tagasiside oli positiivne. Leiti, et hästi on tabatud probleemi olemust ning see ära lahendatud. Heaks omaduseks peeti, et rakendus ei ole keeruline ning selle kasutamine on võimalik kiirelt kasutajatele selgeks teha. Plaan on rakendus firmas kasutusele võtta ning vastavalt jooksvalt tekkivatele vajadustele seda edasi arendada. Loodud rakendus annab võimaluse planeerida arendustöid ja neile prioriteete määrata ühes rakenduses ja asünkroonselt ning muudab tööprotsessi kiiremaks ja mugavamaks.

5.6 Edasiarendused

Rakenduse arenduse käigus sai Infotark AS-i IT-juhiga soovitud funktsionaalsuse osas konsulteeritud, ent lõplik rakenduse koodibaasi üleandmine on alles plaanis. Paigaldus asutuse serverisse ja reaalse andmebaasi ühendamine toimub peale rakenduse üleandmist.

Loodud dokumentatsioon peaks võimaldama rakenduse hõlpsat edasist haldust ja muudatusi. Tööde graafilist kuva saab täiendada võimalusega muuta tööde algus- ja lõppkuupäeva töid graafikul lohistades, mis käesoleva töö skoobist välja jäi. Soovi korral saab lisaks tööde vaatele nimekirjana ja graafilisel ajaskaalal luua võimaluse kuvada neid sprintidena (fikseeritud pikkusega üksus tööde planeerimiseks Scrumi metoodikas [71]) või Kanbani tahvlil (tööde visualiseerimise vahend Kanbani metoodikas [72]).

Esmase rakendusega tutvumise järel Infotark AS-i IT-juhi poolt leiti, et lisaks sorteerimise võimalusele võiks lisada ka erinevaid tööde filtreerimise võimalusi. Samuti oleks hea, et tööde staatuseid oleks võimalik hiljem lisada. Ülejäänud muudatused selguvad rakenduse testimise ja esmase kasutuse käigus reaalsete kasutajate poolt.

6 Kokkuvõte

Käesoleva bakalaureusetöö käigus teostati analüüs arendustööde planeerimise ja prioriteetide määramise rakenduse loomiseks ning valiti prioriteetide arvutamiseks sobivaim tehnika, milleks osutus WSJF (*Weighted Shortest Job First*). Ühtlasi valmis rakenduse esmane kasutatav versioon. Kavandamise faas hõlmas nõuete kogumist, kasutatavate tehnoloogiate analüüsi, rakenduse peamise funktsionaalsuse kaardistamist voodiagrammide abil ning prototüübi ja olemi-suhte diagrammi loomist.

Valminud rakendus võimaldab töid sisestada, muuta, kustutada ja kommenteerida ning algus- ja lõppkuupäeva põhjal neid graafilisel ajaskaalal kuvada. Kõik kasutajate poolt tehtud muudatused kuvatakse nimeliselt ja ajaliselt töö detailvaates muudatuste logina. Administraatori õigustega kasutaja saab lisada hääletusi töödele prioriteetide arvutamiseks ning määrata kasutajaid, kes hääletuses osaleda saavad. Tööde prioriteedid arvutatakse kõikide osalejate hinnangute keskmisena kasutades WSJF tehnikat. Loodud rakendust testiti nii manuaalselt kui ka üksusteste kasutades.

Bakalaureusetöö jaoks püstitatud eesmärk sai saavutatud. Infotark AS saab võtta kasutusele rakenduse arendustööde planeerimiseks ja prioriteetide arvutamiseks meeskonnana, mille puudumine oli selle töö ajendiks. Võimalus teha seda ühes rakenduses ja asünkroonselt muudab tööprotsessi kiiremaks ja mugavamaks. Rakendus antakse üle asutuse IT-juhile, kes vastutab selle edasise haldamise ja arendamise eest.

Kasutatud kirjandus

[1] International Institute of Business Analysis, „BABOK. A guide to the business analysis body of knowledge,“ 2018. [Võrgumaterjal]. Available: https://book.akij.net/eBooks/2018/September/5b8a80dd494ce/BABOK_Guide_v3_Member.pdf. [Kasutatud 30 01 2021].

[2] M. Rehkopf, „User Stories with Examples and Template,“ 2018. [Võrgumaterjal]. Available: https://www.atlassian.com/agile/project-management/user-stories. [Kasutatud 30 01 2021].

[3] T. Ottinger ja J. Langr, „FURPS+,“ 2009. [Võrgumaterjal]. Available: http://agileinaflash.blogspot.com/2009/04/furps.html. [Kasutatud 30 01 2021].

[4] Agile Business Consortium Limited, „Chapter 10: MoSCoW Prioritisation,“ 2021. [Võrgumaterjal]. Available: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation. [Kasutatud 30 01 2021].

[5] B. Aston, „10 Best Agile Tools For Managing Projects In 2021,“ 04 01 2021. [Võrgumaterjal]. Available: https://thedigitalprojectmanager.com/agile-tools/. [Kasutatud 28 01 2021].

[6] Airfocus, „Pricing,“ 2021. [Võrgumaterjal]. Available: https://airfocus.com/pricing/. [Kasutatud 28 01 2021].

[7] Hygger LLC, „Customer-Driven Software Development,“ 2021. [Võrgumaterjal]. Available: https://hygger.io/. [Kasutatud 01 02 2021].

[8] Monday.com, „Work OS: the visual platform that manages everything,“ 2021. [Võrgumaterjal]. Available: https://monday.com/product/. [Kasutatud 28 01 2021].

[9] Smartsheet inc, „Powerful agile project software,“ 2021. [Võrgumaterjal]. Available: https://www.smartsheet.com/s/agile-project-software. [Kasutatud 28 01 2021].

[10] Clarizen, „Product overview,“ 2020. [Võrgumaterjal]. Available: https://3ozxuk3equ8uk744atar6415-wpengine.netdna-ssl.com/wp-content/uploads/2020/05/clarizen-product-overview.pdf. [Kasutatud 28 01 2021].

[11] Taiga Agile, „Taiga project management tool,“ 2021. [Võrgumaterjal]. Available: https://www.taiga.io/. [Kasutatud 28 01 2021].

[12] Forecast, „Agile project management software,“ 2021. [Võrgumaterjal]. Available: https://experience.forecast.it/agile-project-management-software. [Kasutatud 28 01 2021].

[13] Wrike, Inc, „Wrike's work management platform,“ 2021. [Võrgumaterjal]. Available: https://www.wrike.com/. [Kasutatud 28 01 2021].

[14] Netsoft Holdings, LLC, „Hubstaff Tasks,“ 2021. [Võrgumaterjal]. Available: https://hubstaff.com/tasks. [Kasutatud 28 01 2021].

[15] ProjectManager.com, Inc, „Project Management Software for Professionals,“ 2021. [Võrgumaterjal]. Available: https://www.projectmanager.com/. [Kasutatud 28 01 2021].

[16] Zoho Corporation Pvt. Ltd, „Zoho Sprints,“ 2021. [Võrgumaterjal]. Available: https://www.zoho.com/sprints/. [Kasutatud 28 01 2021].

[17] Nutcache, „Nutcache project management software,“ 2021. [Võrgumaterjal]. Available: https://www.nutcache.com/. [Kasutatud 28 01 2021].

[18] ClickUp, „One app to replace them all,“ 2021. [Võrgumaterjal]. Available: https://clickup.com/. [Kasutatud 28 01 2021].

[19] Microsoft, „Microsoft Project,“ 2021. [Võrgumaterjal]. Available: https://www.microsoft.com/en-us/microsoft-365/project/project-management-software. [Kasutatud 28 01 2021].

[20] Atlassian, „Jira Software,“ 2021. [Võrgumaterjal]. Available: https://www.atlassian.com/software/jira. [Kasutatud 28 01 2021].

[21] J.-P. Lang, „Redmine,“ 2014. [Võrgumaterjal]. Available: https://www.redmine.org/projects/redmine/wiki. [Kasutatud 03 02 2021].

[22] Airfocus, „Build outstanding products with powerful prioritization and clear roadmaps,“ 2021. [Võrgumaterjal]. Available: https://airfocus.com/. [Kasutatud 28 01 2021].

[23] Sapience Consulting LLC, „Strategic Quadrant. An agile backlog prioritization tool,“ [Võrgumaterjal]. Available: https://www.strategicquadrant.com/agile-product-backlog-prioritization-software/. [Kasutatud 28 01 2021].

[24] D. Zacarias, „20 Product Prioritization Techniques: A Map and Guided Tour,“ 2016. [Võrgumaterjal]. Available: https://foldingburritos.com/product-prioritization-techniques/. [Kasutatud 31 01 2021].

[25] A. Nash, „Guide to Product Planning: Three Feature Buckets,“ 2009. [Võrgumaterjal]. Available: https://adamnash.blog/2009/07/22/guide-to-product-planning-three-feature-buckets/. [Kasutatud 01 02 2021].

[26] Scaled Agile, Inc, „Weighted Shortest Job First,“ 2021. [Võrgumaterjal]. Available: https://www.scaledagileframework.com/wsjf/. [Kasutatud 01 02 2021].

[27] B. Hartman, „An Introduction to Planning Poker,“ 2014. [Võrgumaterjal]. Available: http://athena.ecs.csus.edu/~buckley/CSc231_files/Introduction%20to%20Planning%20Poker.pdf. [Kasutatud 03 02 2021].

[28] Microsoft, „What is .NET Framework,“ 2021. [Võrgumaterjal]. Available: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework. [Kasutatud 20 02 2021].

[29] Microsoft, „.NET Core/5+ vs. .NET Framework for server apps,“ 2021. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server?toc=%2Faspnet%2Fcore%2Ftoc.json&bc=%2Faspnet%2Fcore%2Fbreadcrumb%2Ftoc.json&view=aspnetcore-5.0. [Kasutatud 20 02 2021].

[30] R. Lander, „Introducing .NET 5,“ 2019. [Võrgumaterjal]. Available: https://devblogs.microsoft.com/dotnet/introducing-net-5/. [Kasutatud 20 02 2021].

[31] D. Roth, R. Anderson ja S. Luttin, „Introduction to ASP.NET Core,“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0. [Kasutatud 20 02 2021].

[32] C. Nienaber ja R. Suter, „ASP.NET Core web API documentation with Swagger / OpenAPI,“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-5.0. [Kasutatud 20 02 2021].

[33] S. Kharche, „API Versioning in ASP.NET Core,“ 2020. [Võrgumaterjal]. Available: https://www.c-sharpcorner.com/article/api-versioning-in-asp-net-core-web-api/. [Kasutatud 20 02 2021].

[34] Oracle, „What Is a Relational Database,“ 2021. [Võrgumaterjal]. Available: https://www.oracle.com/database/what-is-a-relational-database/. [Kasutatud 21 02 2021].

[35] S. Contributor, „MySQL vs. MSSQL—Performance and Main Differences Between Database and Servers,“ 2020. [Võrgumaterjal]. Available: https://www.dnsstuff.com/mysql-vs-mssql-performance. [Kasutatud 21 02 2021].

[36] G. Harrison ja S. Feuerstein, „MySQL Stored Procedure Programming,“ 2006. [Võrgumaterjal]. Available: https://books.google.ee/books?id=YpeP0ok0cO4C&pg=PT75&redir_esc=y#v=onepage&q&f=false. [Kasutatud 21 02 2021].

[37] Oracle, „MySQL Database Service,“ 2021. [Võrgumaterjal]. Available: https://www.oracle.com/mysql/. [Kasutatud 21 02 2021].

[38] Oracle Corporation, „MySQL Products,“ 2021. [Võrgumaterjal]. Available: https://www.mysql.com/products/. [Kasutatud 21 02 2021].

[39] Microsoft, „Transact-SQL Reference (Database Engine),“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver15. [Kasutatud 21 02 2021].

[40] Microsoft, „Microsoft SQL Server 2019 Express,“ 2021. [Võrgumaterjal]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=101064. [Kasutatud 21 02 2021].

[41] ZZZ Projects & .NET Community, „LocalDB is not supported on this Platform,“ 2017. [Võrgumaterjal]. Available: https://entityframework.net/knowledge-base/45860851/localdb-is-not-supported-on-this-platform. [Kasutatud 21 04 2021].

[42] Microsoft, „Migrations Overview,“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli. [Kasutatud 21 04 2021].

[43] E. Gamma, R. Helm, R. Johnson ja J. Vlissides, „Design Patterns: Elements of Reusable Object-Oriented Software,“ 1997. [Võrgumaterjal]. Available: http://www.uml.org.cn/c++/pdf/designpatterns.pdf. [Kasutatud 21 02 2021].

[44] R. Anderson ja R. Nowak, „Introduction to Razor Pages in ASP.NET Core,“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-5.0&tabs=visual-studio. [Kasutatud 21 02 2021].

[45] Restfulapi.net, „What is REST,“ 2020. [Võrgumaterjal]. Available: https://restfulapi.net/. [Kasutatud 21 02 2021].

[46] Neoteric, „Single-page application vs. multiple-page application,“ 2016. [Võrgumaterjal]. Available: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. [Kasutatud 24 02 2021].

[47] The PHP Group, „Php,“ 2021. [Võrgumaterjal]. Available: https://www.php.net/. [Kasutatud 24 02 2021].

[48] I. Kantor, „An Introduction to JavaScript,“ 2021. [Võrgumaterjal]. Available: https://javascript.info/intro. [Kasutatud 24 02 2021].

[49] Microsoft, „What is TypeScript,“ 2021. [Võrgumaterjal]. Available: https://www.typescriptlang.org/. [Kasutatud 24 02 2021].

[50] E. Campos, „Is PHP Hard to Learn?,“ 2021. [Võrgumaterjal]. Available: https://study.com/academy/popular/is-php-hard-to-learn.html. [Kasutatud 24 02 2021].

[51] A. Meyster, „How long does it take to learn JavaScript,“ 2020. [Võrgumaterjal]. Available: https://jaxenter.com/learn-javascript-171411.html. [Kasutatud 24 02 2021].

[52] A. Sviatoslav, „The Best JS Frameworks for Front End,“ 2020. [Võrgumaterjal]. Available: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end. [Kasutatud 24 02 2021].

[53] M. Nowak, „Vue vs React in 2021: Which Framework to Choose and When,“ 2020. [Võrgumaterjal]. Available: https://www.monterail.com/blog/vue-vs-react-2021. [Kasutatud 24 02 2021].

[54] C. Dulanga, „Auralia JS vs. Angular,“ 2020. [Võrgumaterjal]. Available: https://betterprogramming.pub/auralia-js-vs-angular-ccaf61e4d27d. [Kasutatud 24 02 2021].

[55] Slant Team, „Aurelia vs Vue.js,“ 2021. [Võrgumaterjal]. Available: https://www.slant.co/versus/37/11378/~aurelia_vs_vue-js. [Kasutatud 24 02 2021].

[56] Slant Team, „What are the best C# IDEs,“ 2021. [Võrgumaterjal]. Available: https://www.slant.co/topics/4118/~c-ides. [Kasutatud 21 03 2021].

[57] JetBrains s.r.o., „Free License Programs,“ 2021. [Võrgumaterjal]. Available: https://www.jetbrains.com/community/education/#students. [Kasutatud 21 03 2021].

[58] Software Testing Help, „15 BEST Version Control Software (Source Code Management Tools),“ 2021. [Võrgumaterjal]. Available: https://www.softwaretestinghelp.com/version-control-software/. [Kasutatud 21 03 2021].

[59] Full Scale, „Top 10 version control systems,“ 2020. [Võrgumaterjal]. Available: https://fullscale.io/blog/top-10-version-control-systems/. [Kasutatud 21 03 2021].

[60] A. Vickers, „Identity model customization in ASP.NET Core,“ 2019. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-5.0. [Kasutatud 20 03 2021].

[61] Microsoft, „An introduction to NuGet,“ 2019. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/nuget/what-is-nuget. [Kasutatud 10 03 2021].

[62] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee ja R. Stafford, „Patterns of enterprise Application Architecture,“ 2003. [Võrgumaterjal]. Available: https://github.com/himanshugpt/ebooks-1/blob/master/Patterns%20of%20Enterprise%20Application%20Architecture%20-%20Martin%20Fowler.pdf. [Kasutatud 10 03 2021].

[63] Auth0, „Introduction to JSON Web Tokens,“ 2021. [Võrgumaterjal]. Available: https://jwt.io/introduction. [Kasutatud 21 03 2021].

[64] J. Padilla, „Digital Signature Algorithms,“ 2015. [Võrgumaterjal]. Available: https://pyjwt.readthedocs.io/en/latest/algorithms.html. [Kasutatud 21 03 2021].

[65] R. Anderson, K. Larkin, D. Roth ja S. Addie, „Safe storage of app secrets in development in ASP.NET Core,“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-5.0&tabs=windows. [Kasutatud 24 03 2021].

[66] R. Anderson, „Enforce HTTPS in ASP.NET Core,“ 2019. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/enforcing-ssl?view=aspnetcore-5.0&tabs=visual-studio. [Kasutatud 24 03 2021].

[67] A. Russell, „What is HTTPS,“ 2020. [Võrgumaterjal]. Available: https://www.ssl.com/faqs/what-is-https/. [Kasutatud 24 03 2021].

[68] ZZZ Projects & .NET Community, „LINQ Prevent SQL Injection,“ 2021. [Võrgumaterjal]. Available: https://entityframework.net/linq-prevent-sql-injection. [Kasutatud 24 03 2021].

[69] Microsoft, „Testing in .NET,“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/core/testing. [Kasutatud 21 04 2021].

[70] NuGet Must Haves, „Top 20 NuGet mocking Packages,“ 2021. [Võrgumaterjal]. Available: https://nugetmusthaves.com/Tag/mocking. [Kasutatud 21 04 2021].

[71] Scrum.org, „What is a Sprint in Scrum,“ 2021. [Võrgumaterjal]. Available: https://www.scrum.org/resources/what-is-a-sprint-in-scrum. [Kasutatud 22 04 2021].

[72] Digité, Inc, „What is a Kanban Board,“ 2021. [Võrgumaterjal]. Available: https://www.digite.com/kanban/kanban-board. [Kasutatud 22 04 2021].

[73] S. Addie ja T. Dykstra, „Create web APIs with ASP.NET Core,“ 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-5.0. [Kasutatud 21 02 2021].

[74] Vue Community, „Editors and Tools,“ 2021. [Võrgumaterjal]. Available: https://vue-community.org/guide/ecosystem/editors-and-tools.html#editors. [Kasutatud 21 03 2021].

[75] I. Marko Z, „Homepage of the vue-ganttastic project,“ 2020. [Võrgumaterjal]. Available: https://github.com/InfectoOne/vue-ganttastic-homepage. [Kasutatud 22 04 2021].

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

Mina, Piret Gorban

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Arendustööde planeerimise ja prioriteetide määramise rakenduse analüüs ja väljatöötamine", mille juhendaja on Nadežda Furs
   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.

16.05.2021

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. Intervjuu küsimused ja vastused

1. Millist tarkvara praegu tööde planeerimiseks ja projekti halduseks kasutatakse?

- *Milleks?* Kaardistamaks hetkeolukorda.

Excel – lihtsamate ülesannete kaardistamine.
Redmine – maja siseste ülesannete sisestamiseks. Kuna see on suunatud arendajale, siis on äri poole inimestel keeruline sellega töötada. Liiga palju funktsionaalsust.
Jira – e-poe tööde haldamiseks koos välise partneriga. Klient haldab oma ülesandeid seal. Ei ole primaarne töövahend.
MS Project – veidi kasutusel, asendamaks/täiendamaks Excelisse sisestatud ülesannete nimekirju. Suurim eelis on Gantti graafiku vaade, mis võimaldab ülesannete ajalist kuva.

2. Mis on probleemid olemasolevate lahenduste kasutamisel?

- *Milleks?* Milles seisneb probleem olemasolevate vahendite puhul.

Iga süsteem sisaldab omaette vajalikku funktsionaalsust, aga üheski neist ei ole võimalik kõiki soovitavaid asju ära teha. Seega pead pidevalt programmide vahel laveerima ning raske on saada terviklikku ülevaadet.

3. Millisest funktsionaalsusest puudu jääb?

- *Milleks?* Tuvastamaks konkreetsemaid kasutajate vajadusi.

Ennekõike tuntakse puudust võimalusest töödele prioriteete määrata/arvutada. Mitmed rakendused võimaldavad küll prioriteete määrata, kuid neid ei saa erinevate kriteeriumite põhjal arvutada. Meeskonnana prioriteetide määramiseks tuleb need koosolekutel suuliselt läbi arutada, kus kipub rohkem ühe-kahe inimese avamus peale jääma. Oleks hea, kui saaks prioriteete üksteise arvamustest sõltumatult arvutada. Samuti võiks olla näha kirjete muudatuste ajalugu. Redmine säilitab muudatuste ajalugu, ent teistel rakendustel selline funktsionaalsus puudub.

4. Kas ja kuidas olete proovinud seda lahendada?

- *Milleks?* Selgitamaks, kas on võimalik see olemasolevate vahenditega ära lahendada.

Iga uue vajaliku funktsionaalsuse jaoks on toodud sisse uus tarkvara, sellepärast neid nii palju paralleelselt ongi.

5. Kuidas te näeksite, et seda probleemi lahendada võiks?

- *Milleks?* Nägemaks nende visiooni lahenduse osas.

Tahaks lihtsat tarkvara, mis ei sisaldaks rohkelt üleliigset funktsionaalsust, vaid mida oleks lihtne kasutada ning teeks seda, mida vaja ja teeks seda hästi. Suur tarkvara vajab rohkem haldamist ja selle kasutamist on kasutajatele keerulisem selgeks õpetada.

6. Kes on kasutajad, keda see probleem puudutab? Palju neid on?

- *Milleks?* Kui suur oleks tulevaste kasutajate hulk, milline on nende taust ja oskused. Millisele kasutajate hulgale peab süsteem mõeldud olema.

Juhtkond (äri poole hinnangud), IT-osakond (tehnilisemad ideed ja teostus). Kokku oleks umbes 15 kasutajat.

7. Kui suureks võib kasutajate hulk tulevikus kasvada?

- *Milleks?* Arvestamaks loodava rakenduse mittefunktsionaalsete nõuete kirja panemisel tuleviku perspektiiviga.

Kasutajate hulk võib kasvad, aga mitte väga oluliselt. Suure varuga võttes võib tulevikus ette näha 50 kasutajat.

8. Kas ja kui palju ollakse valmis uue lahenduse eest maksma?

- *Milleks?* Selgitamaks, kas on võimalik kasutada tasulisi rakendusi ning mis oleks nende hinnapiir.

Oleks nõus kasutama tasulist vaid siis kui väga vaja. Kuna igapäevatööks kasutatakse ka mitmeid tasulisi tarkvarasid, millest ei soovita loobuda, siis ei pruugi juhtkonna silmis olla põhjendatud omakorda uute tasuliste tarkvarade sisse toomine. Igakuised kogukulud peaksid jääma alla 200 euro kuus. Eelistatud on tasuta lahendused.

9. Kuidas peaks välja nägema prioriteetide arvutamine meeskonnana

- *Milleks?* Sellest sõltub olemasolevate rakenduste analüüs ja vajadusel uue rakenduse planeerimine.

Kasutajad peaksid saama nimekirja töödest, mida on vaja hinnata ning saaksid kindlaks määratud perioodi jooksul saama määratud kriteeriumite põhjal töödele hindeid anda.

Hinnangut peaks saama selle perioodi jooksul muuta, muudatuste ajalugu peaks näha olema. Perioodi lõppedes arvutatakse üksikute hinnete põhjal igale tööle koondhinne.

10. Mis keeli rakendus toetama peaks?

- *Milleks?* Sellest sõltub milliste keelte tuge rakendusse lisada.

Rakendus võiks olla ingliskeelne. Tõlkimise võimalus võiks ju olla, aga see ei ole nii oluline. Kõik on harjunud kasutama inglise keelseid rakendusi.

11. Kas uue loodava rakenduse jaoks on teie poolt ka tehnilisi piiranguid või nõudeid?

- *Milleks?* Selgitamaks millised on tehnilised eeltingimused rakenduse loomisel, kui see peaks vajalikuks osutuma.

Rakendus peaks toetama versioneerimist, et peale esimest kasutuselevõttu oleks võimalik seda edasi arendada ning paralleelselt laiemas kasutuses oleva versiooniga oleks võimalik testida järgmist versiooni.

Serveri poole peal peaks kasutama .NET tehnoloogiat ja C#, kuna asutuses kasutatakse valdavalt Microsofti tehnoloogiaid ning soov on seda suunda jätkata. Ühtlasi on olemas kompetents nende põhjal loodud rakenduste edasi arendamiseks. Java rakendusi hetkel majas ei arendata ning puudub ka huvi seda tegema hakata.

Andmebaasi osas on eelistatuim variant Microsoft SQL Server, ent sobiks ka MySQL. Need on hetkel asutuses kasutavad andmebaasid ning ilma mõjuva põhjuseta uusi tehnoloogiad kasutusele võtta ei soovita.

12. Kas töökindluse osas on loodava rakenduse jaoks konkreetseid nõudeid?

- *Milleks? Selgitamaks mittefunktsionaalseid nõudeid.*

Kuna tegu ei ole ärikriitilise lahendusega, siis töökindluse osas konkreetsed nõuded puuduvad.

13. Kuidas peaks rakendusse sisse logida saama?

- *Milleks?* Sellest sõltub vajadus loodava rakenduse liidestamiseks teiste süsteemidega.

Esmases faasis võiks kasutajate sisse logimine toimuda kasutajanime ja parooliga. Administraatoril peaks olema õigus kasutajaid luua. Liidestus ID-kaardi, Mobiil-ID ja Smart-ID-ga oleks tore, aga see ei ole esmajärjekorras vajalik.

14. Kas erinevatel kasutajatel peaksid erinevate kasutajaõigustega rollid olema?

- *Milleks?* Rollipõhiste kasutajaõiguste kindlaks tegemises loodavas rakenduses.

Jah, kõigil kasutajatel peaks olema õigus ülesandeid luua, muuta ja kustutada, nende ajalugu näha ja neid graafikul kuvada. Ent ainult teatud kasutajatel (nii-öelda peakasutajal) peaks olema õigus luua gruppe kasutajatest ja töödest prioriteetide arvutamiseks, määrata hindamise perioodi algus- ja lõppkuupäeva ja seda ka vajadusel muuta.

Lisa 3. Nõudeid kirjeldavad kasutajalood

Epik 1: Kasutajana soovin sisestada, muuta ja kustutada ülesandeid.

<kellena>	**soovin**	**<teha midagi selleks, et>**	**<saavutada midagi>**
Kasutajana	soovin	sisestada uusi ülesandeid	et lisada planeeritavaid töid.
Kasutajana	soovin	muuta juba sisestatud ülesandeid	et parandada olemasolevat informatsiooni.
Kasutajana	soovin	kustutada sisestatud ülesandeid	et ei kuvataks mitte vajalikuks osutunud töid.
Kasutajana	soovin	määrata töödele algus- ja lõppkuupäeva	et oleks teada nende planeeritav teostamise aeg.
Kasutajana	soovin	määrata töödele teostaja	et oleks näha, kes sellega tegeleb.
Kasutajana	soovin	määrata tööle staatust	et oleks näha mis seisus see ülesanne on.

Epik 2: Kasutajana soovin näha sisestatud töid ajalisel skaalal Gannti graafiku sarnaselt.

<kellena>	**soovin**	**<teha midagi selleks, et>**	**<saavutada midagi>**
Kasutajana	soovin	valida perioodi, mille kohta mulle graafik kuvatakse	et näha erinevate perioodide kavandatud töid.
Kasutajana	soovin	et erineva pikkusega tööd oleks graafikul eristatavad	et oleks lihtne eristada pikemaid ja lühemaid töid.
Kasutajana	soovin	graafikul töö peal klõpsates näha selle detailvaadet	et ma saaksin kiirelt lisainfot lugeda.
Kasutajana	soovin	graafikul töid lohistada	et oleks mugav nende algus- ja lõppkuupäeva muuta.

Epik 3. Kasutajana soovin näha tööde prioriteete ja neid lisada.

<kellena>	**soovin**	**<teha midagi selleks, et>**	**<saavutada midagi>**
Peakasutajana	soovin	määrata töödele prioriteete	et oleks aru saada, milliseid töid on vaja varem teostada.
Kasutajana	soovin	arvutada tööde prioriteete erinevate kriteeriumite alusel	et oleks lihtsam lõplikku hinnangut kujundada.
Peakasutajana	soovin	määrata tööle prioriteeti mitme inimese hinnangu põhjal	et prioriteedid ei kajastaks vaid ühe inimese hinnangut.
Peakasutajana	soovin	valida kasutajaid, kes saavad töid hinnata	et hindaksid ainult asjasse puutuvad kasuajad.
Peakasutajana	soovin	valida tööde nimekirja, mida hindama hakatakse	et hetkel ebavajalikud tööd kõrvale jätta.
Peakasutajana	soovin	määrata perioodi, mil tööde hindamine võimalik on	et selle pikkust ise valida.
Peakasutajana	soovin	muuta perioodi, mil tööde hindamine võimalik on	et seda pikendada või varem lõpetada.
Kasutajana	soovin	näha teiste hindeid alles peale hindamisperioodi lõppu	et need ei mõjutaks minu ega teiste kasutaja hinnanguid.
Kasutajana	soovin	muuta oma hinnangut vastamise perioodi jooksul	et saaksin anda adekvaatsema hinde kui vahepeal oma arvamust muudan.

Epik 4: Kasutajana soovin rakendusse sisse logida.

<kellena>	**soovin**	**<teha midagi selleks, et>**	**<saavutada midagi>**
Peakasutajana	soovin	registreerida kasutajaid	et neil oleks võimalik rakendusse sisse logida.
Kasutajana	soovin	sisse logida	et näha mulle hindamiseks määratud ülesandeid ja et minu tehtud muudatusi logitaks minu nime all.
Kasutajana	soovin	muuta oma parooli	et saaksin parooli unustades rakendust edasi kasutada.

Epik 5: Kasutajana soovin näha kirjete muudatuste ajalugu.

<kellena>	**soovin**	**<teha midagi selleks, et>**	**<saavutada midagi>**
Kasutajana	soovin	näha tööde algus- ja lõppkuupäeva muudatusi	et ma teaksin, kui mingi töö algselt kavandatud aega on muudetud.
Kasutajana	soovin	näha millal töö staatust on muudetud	et hiljem oleks teada, mis staatuses ta millal olnud on.
Kasutajana	soovin	näha kui teostajat muudetakse	et teada, kui see on kellelegi teisele määratud.
Kasutajana	soovin	näha kui töö kirjeldust on muudetud	et olla teadlik sellest, kui midagi on hiljem lisatud.
Kasutajana	soovin	näha kui töö prioriteeti on muudetud	et planeerida töid vastavalt prioriteetidele ümber.
Kasutajana	soovin	näha prioriteetide arvutamise ajalugu kasutajate kaupa	et olla teadlik kasutajate hinnangutest ja nende muutmisest.

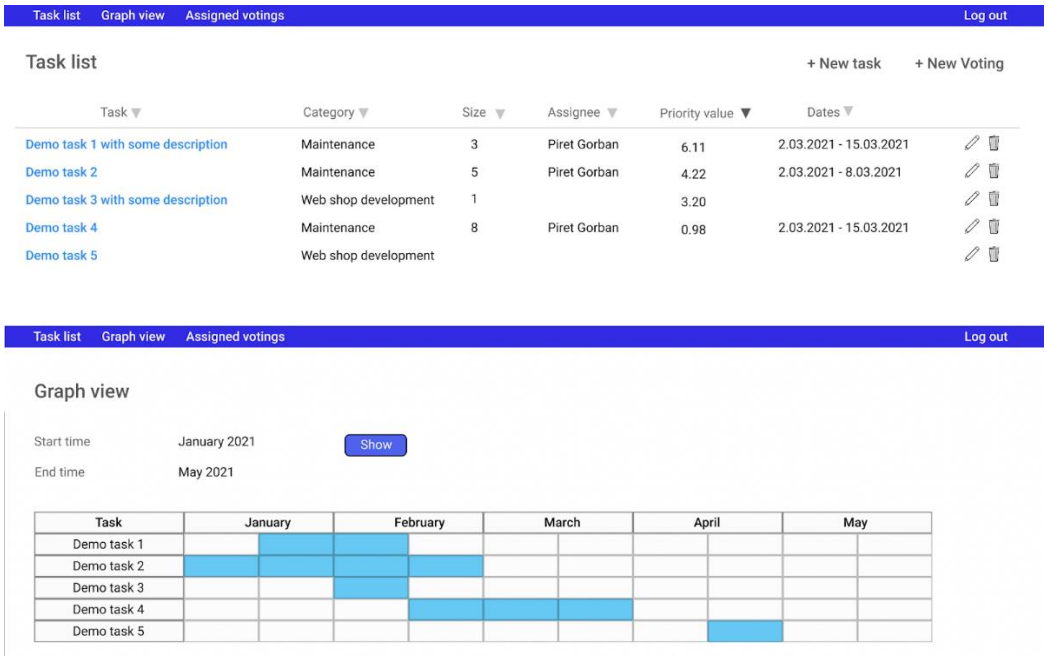
Lisa 4. Nõuete klassifitseerimine FURPS+ abil

Funktsionaalsus	Kasutaja saab sisestada, muuta ja kustutada ülesandeid. Kasutaja saab määrata töödele algus- ja lõppkuupäeva, teostajat ja staatust. Kasutaja saab arvutada tööde prioriteete erinevate kriteeriumite alusel. Töö prioriteet arvutatakse mitme inimese hinnangu põhjal. Peakasutaja saab valida kasutajaid, kes saavad töid hinnata. Peakasutaja saab valida tööde nimekirja, mida hindama hakatakse. Peakasutaja saab määrata ja muuta perioodi, mil tööde hindamine kasutajate jaoks võimalik on. Kasutaja saab näha teiste hindeid alles peale hindamisperioodi lõppu. Kasutaja saab muuta oma hinnangut vastamise perioodi jooksul. Kasutaja saab sisestatud töid näha graafilisel ajaskaalal. Kasutaja saab valida perioodi, mille kohta ajaline graafik kuvatakse. Erineva pikkusega tööd on ajalisel graafikul visuaalselt eristatavad. Ajalisel graafikul töö peal klõpsates on võimalik näha selle detailvaadet. Ajalisel graafikul saab töid lohistades muuta nende algus- ja lõppkuupäeva. Kasutaja saab näha kirjete muutmise ajalugu (algus- ja lõppkuupäeva, staatuse, teostaja, töö kirjelduse muudatusi). Kasutaja saab näha prioriteetide arvutamise ajalugu (hindepunktide muudatusi avatud hindamisperioodi jooksul) kasutajate kaupa. Peakasutaja saab registreerida uusi kasutajaid. Kasutaja saab rakenduse veebilehel sisse logida ja parooli muuta.

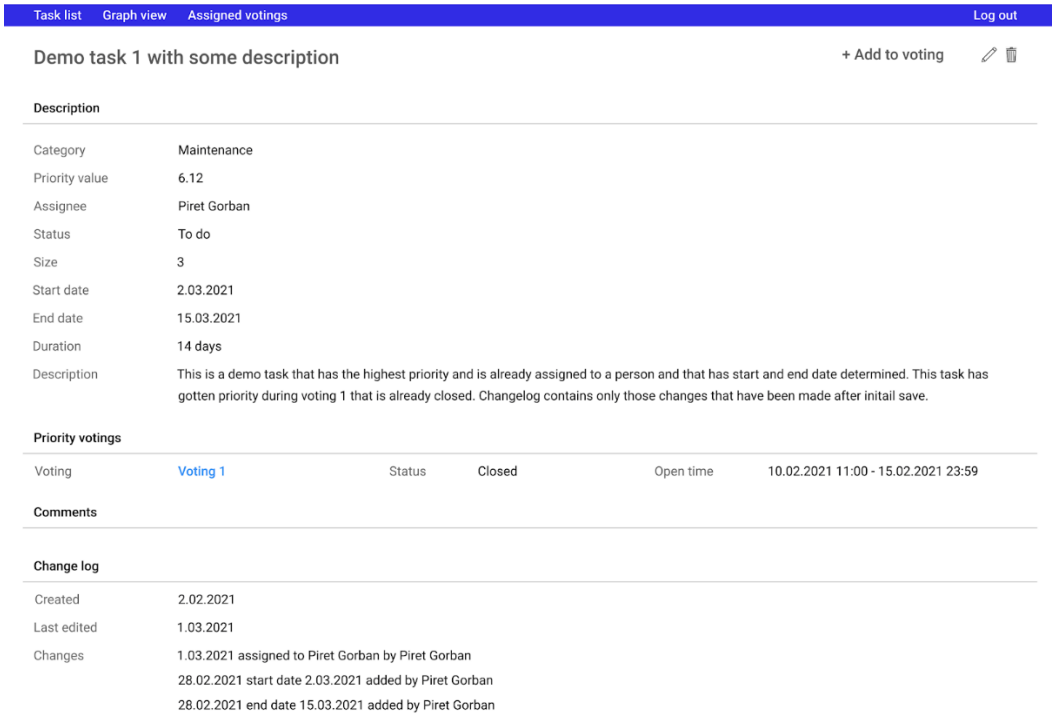
Kasutatavus	Rakenduse kasutusvõimalused peavad olema dokumenteeritud. Rakendus peab toetama versioneerimist. Kasutaja tuleb sisse logides suunata prioriteetide hindamise lehele kui on avatud hindamisi, mis on talle määratud.
Töökindlus	Kuna tegu ei ole ärikriitilise lahendusega, siis töökindluse osas konkreetsed nõuded puuduvad.
Jõudlus	Rakendus peab olema võimeline toetama 50 paralleelset sessiooni.
Toetatavus	Peab toetama erinevaid keeli (eesti keel, inglise keel, vene keel). Rakendus peab olema kasutatav enamlevinud kaasaegsetes veebilehitsejates.

+ Peab saama sisse logida ID-kaardiga, Mobiil-IDga ja Smart-IDga.

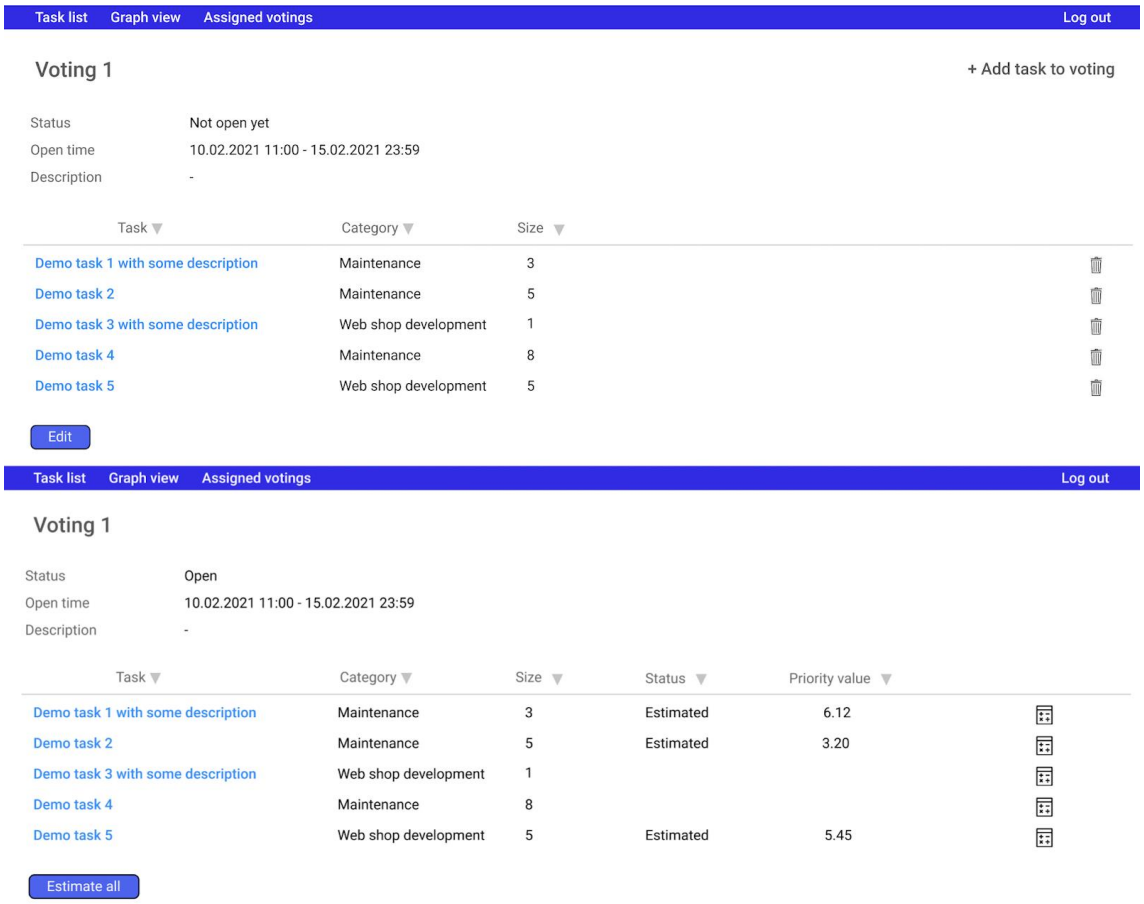
Lisa 5. Rakenduse prototüübi peamised vaated



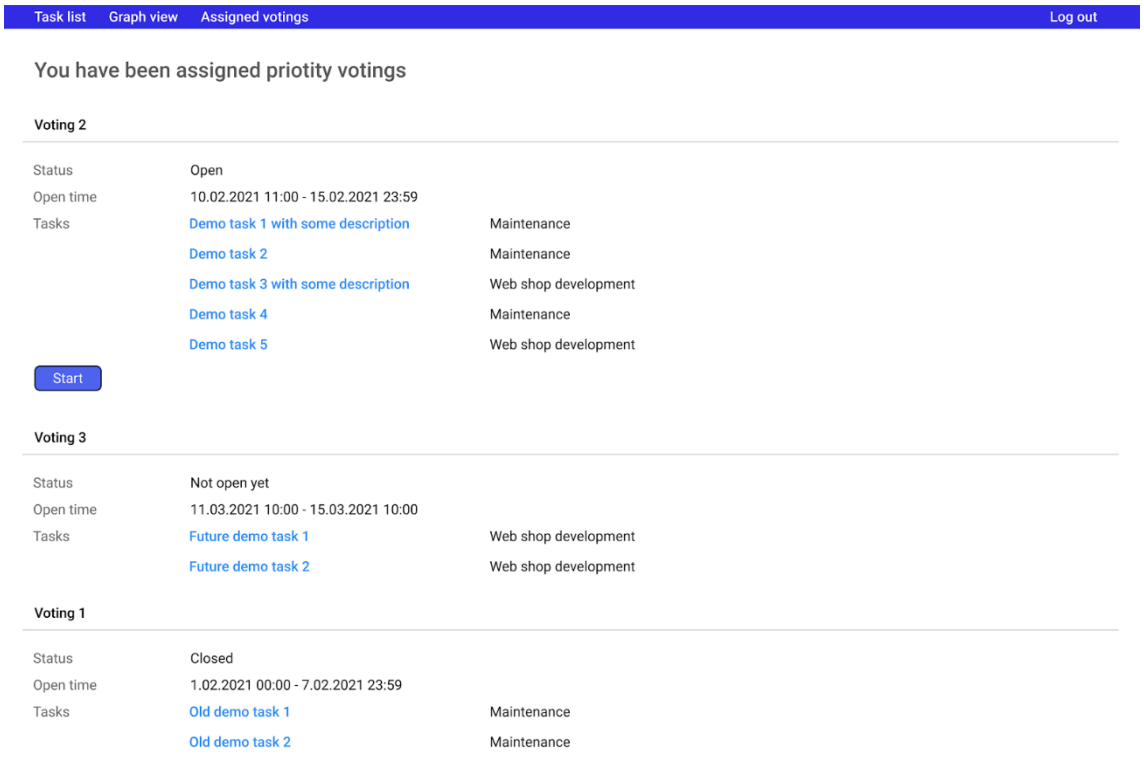
Joonis 3-1. Tööde vaade nimekirjana (üleval) ja graafilisel ajaskaalal.



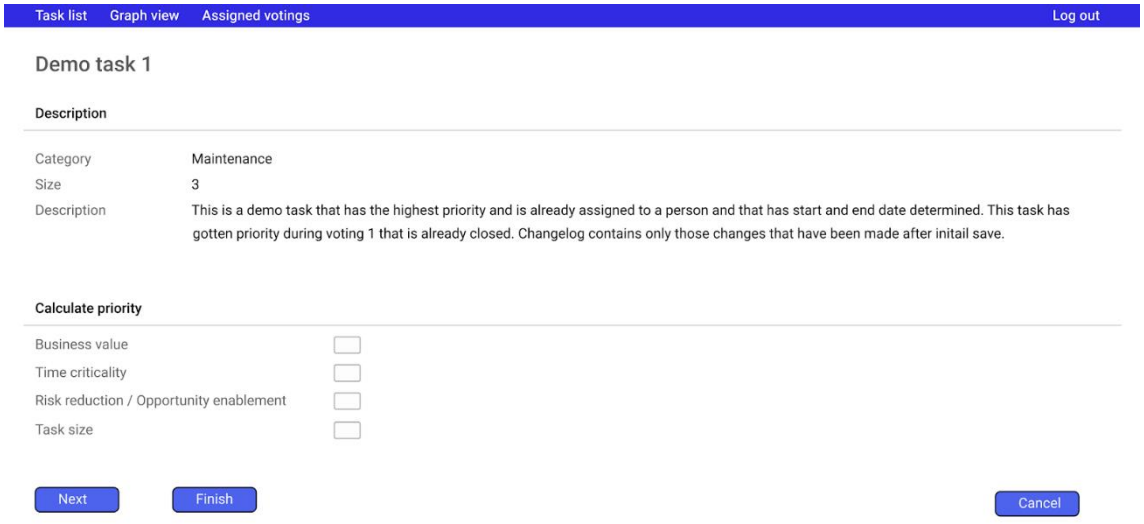
Joonis 3-2. Töö detailvaade.



Joonis 3-3. Prioriteetide hääletamise vaade administraatori (ülemine) ja tavakasutaja jaoks.



Joonis 3-4. Kasutajale määratud prioriteetide hääletused.



Joonis 3-5. Arendustöö prioriteedi arvutamise vaade.

Lisa 6. Viide rakenduse koodile

Tagarakenduse kood: https://gitlab.cs.ttu.ee/pigorb/PlanAndPrioritize
Klientrakenduse kood: https://gitlab.cs.ttu.ee/pigorb/PlanAndPrioritizeFront

Ligipääs on kõigil GitLabi kasutajakontot omavatel isikutel.

Projekti DAL.App.EF kausas Helpers on DataInitializers klass, kuhu on testimise jaoks sisse kirjutatud esmase administraatori õigustega kasutaja andmed, millega rakendusse sisse saab logida. Hiljem eemaldatakse see projekti koodist.