TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aneta Claudia Marttila 179459IADB

TalTech üliõpilasesinduse rahastusplatvormile ruumide haldamise ja pikaajalise kasutusõiguse jagamise funktsionaalsuse arendus

Bakalaureusetöö

Juhendaja: Brita Moorus
B.Sc.
Kaasjuhendaja: Krõõt Grete Mänd
B.Sc.

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Aneta Claudia Marttila

03.01.2023

Annotatsioon

Lõputöö kirjutamise ajahetkel on tudengiorganisatsioonil võimalik läbi rahastusplatvormi üliõpilasesindusele (edaspidi ÜE) teada anda, et nad soovivad taotleda ÜE-lt organisatsiooni tegevuseks ruumi. Lisaks ei ole võimalik tudengiorganisatsioonil läbi rahastusplatvormi valida, millistele ruumidele nad soovivad pikaajalist kasutusõigust. Samuti puudub avalik nimekiri ruumidest, mis on ÜE poolt hallatavad.

Käesoleva lõputöö raames võtab autor üle varasemate lõputööde raames poolt loodud koodibaasi ning hakkab tegema lisaarendusi. Lisaarenduste eesmärgiks on luua rahastusplatvormile funktsionaalsused, mis aitavad lihtsustada Tallinna Tehnikaülikooli üliõpilasesindusel ruumide haldamist ja ruumide pikemaajaliste kasutusõiguste väljajagamist.

Funktsionaalsuste realiseerimiseks kogub autor erinevaid tehnikaid kasutades vajalikud nõuded ning nende alusel viib muudatused olemasolevasse koodibaasi. Nõuete kogumiseks suhtleb autor tellijaga ning küsib temalt jooksvalt täiendavat tagasisidet.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 4 peatükki, 14 joonist, 8 tabelit.

Abstract

Development of Room Management and Use-Rights Distribution Functionality for TalTech Student Union's Funding Platform

It is not always possible to predict the future during the analysis phase of a software development project. Functionalities that were not considered during the analysis may come to light only after the application has been in use for some time. Such is the case with the funding platform created for the student council of Tallinn University of Technology.

At the time of writing this thesis, the student organization can inform the student council through the funding platform that they are requesting long-term use rights for student council's rooms for the organization's activities from the student council. Although, it is not possible for the student organization to choose for which rooms they need long-term right of use through the funding platform. There is also no public list of premises managed by the student council.

To solve the problem for this thesis, the author takes over the code base, which was created during previous theses and starts making additional developments. The purpose of these additional developments is to create functionalities for the financing platform that would help to simplify the management of the rooms, which are managed by the Tallinn University of Technology student council, and the distribution of long-term rights to use the premises. To realize these functionalities, the author collects the necessary requirements using various techniques and does necessary changes in the existing code base.

The thesis is in Estonian and contains 30 pages of text, 4 chapters, 14 figures, 8 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
BRD	*Business Requirements Document*, ärinõuete dokument
Controller	Kolmekihilises API arhitektuuris klass, mis kirjeldab kasutaja poolt teostatavaid toiminguid
HTTP	*Hypertext Transfer Protocol,* hüperteksti edastusprotokoll, ettekirjutus, kuidas edastada teavet arvutivõrkudes
JAD/RAD	*Joint Application Development/Rapid Application Development,* ühine tarkvara arendus/kiire tarkvara arendus
JSON	*JavaScript Object Notation*, andmete vorming, mida kasutab enamik veebiliideseid
Modaalaken	Ing. k. *modal window*, kasutajaliidese põhiaknas avanev sekundaaraken
REST	*Representational State Transfer*, tarkvara arhitektuuri stiil, mis kirjeldab kuidas luua veebirakendusi
Service	Kolmekihilises API arhitektuuris klass, mis hoiustab endas äriloogikat
SPA	*Single Page Application,* veebirakenduse liik, mis laeb üles ühe veebidokumendi ja muudatuste korral värskendab seejärel selle dokumendi põhisisu
SRS	*Software Requirements Specifiation*, tarkvara nõuete spetsifikatsioon
Tegevuste skeem	Ing. k. *storyboard*, meetod, mille abil on võimalik kirjeldada visuaalselt eesmärgi saavutamiseks vajalikke järjestatud tegevusi
Tehase muster	Ing. k. *factory pattern*, tarkvaraarenduses kasutatav disainimuster
ÜE	Tallinna Tehnikaülikooli üliõpilasesindus
URI	*Unified Resource Identifier,* adresseerimise tehnoloogia, mida kasutatakse internetis ressursside tuvastamiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tarkvaraarenduse projekti analüüsi etapis ei ole alati võimalik ennustada tulevikku. Funktsionaalsused, millele ei mõeldud analüüsi käigus, võivad tulla välja alles siis, kui rakendus on juba mõnda aega kasutuses olnud. Sama lugu on ka Tallinna Tehnikaülikooli üliõpilasesindusele (edaspidi ÜE) loodud rahastusplatvormiga. Rahastusplatvorm on tööriist, mille abil ÜE saab otsustada, milliseid organisatsioone ja eraisikuid kavatsetakse toetada korraldavate konkursside raames nii rahaliselt kui ka materiaalselt.

Rahastusplatvorm sai loodud Krõõt Grete Mändi [1] ja Raimond Lume [2] poolt bakalaureusetööde raames. Rahastusplatvorm on ÜE poolt kasutuses olnud alates 2020. aasta sügisest. Käesoleva lõputöö raames võtab autor üle varasemate autorite poolt loodud koodibaasi ning hakkab tegema lisaarendusi.

Autor kavatseb töö analüüsi osas uurida erinevaid nõuete kogumise tehnikaid ja selgitab välja enda ja projekti jaoks sobivaimad variandid. Valitud meetodite alusel asub ta kaardistama rahastusplatvormile arendatavate funktsionaalsuste nõudeid.

Töö praktilises osas asub autor teostatud analüüsi põhjal realiseerima rahastusplatvormile lisafunktsionaalsusi. Selle jaoks on tal vaja teha täiendusi andmebaasi, rakendus- ja kasutajaliidesele kasutades varasemalt paika pandud tehnoloogiaid ning arhitektuuri mustreid. Lõpuks kirjeldab autor praktilises osas valminud tööd, arutleb, mida saab paremini teha ja toob välja rahastusplatvormi edasi arendamise võimalusi.

1.1 Projekti taust

Tallinna Tehnikaülikooli üliõpilasesindus toetab tudengiorganisatsioonide tegevusi ning tudengitele suunatud sündmuste korraldamist. Tudengiorganisatsioonide tegevust toetatakse näiteks ruumide kasutusõiguse andmise või rahalise toetuse näol. Selleks, et organisatsioonid saaksid rahastust või ruume taotleda oma tegevuse jaoks, kuulutab üliõpilasesindus välja konkursi.

Üliõpilasesindusel on eraldi rahastuskomisjon, kes vaatab konkursitele laekunud avaldused üle, annab oma hinnangud ja langetab otsuse, kes toetust saavad. Konkursse on lõputöö kirjutamise ajahetkel kahte tüüpi – aastatoetused ja projektid.

Projektide konkursi eesmärgiks on rahastada üritusi, mis on suunatud üliõpilastele. Sellel saavad osaleda nii tudengiorganisatsioonid kui ka eraisikud. Aastatoetused on eelkõige mõeldud tudengiorganisatsioonidele ning konkursi eesmärgiks on anda organisatsioonile üheks kalendriaastaks vajalikud rahalised vahendid.

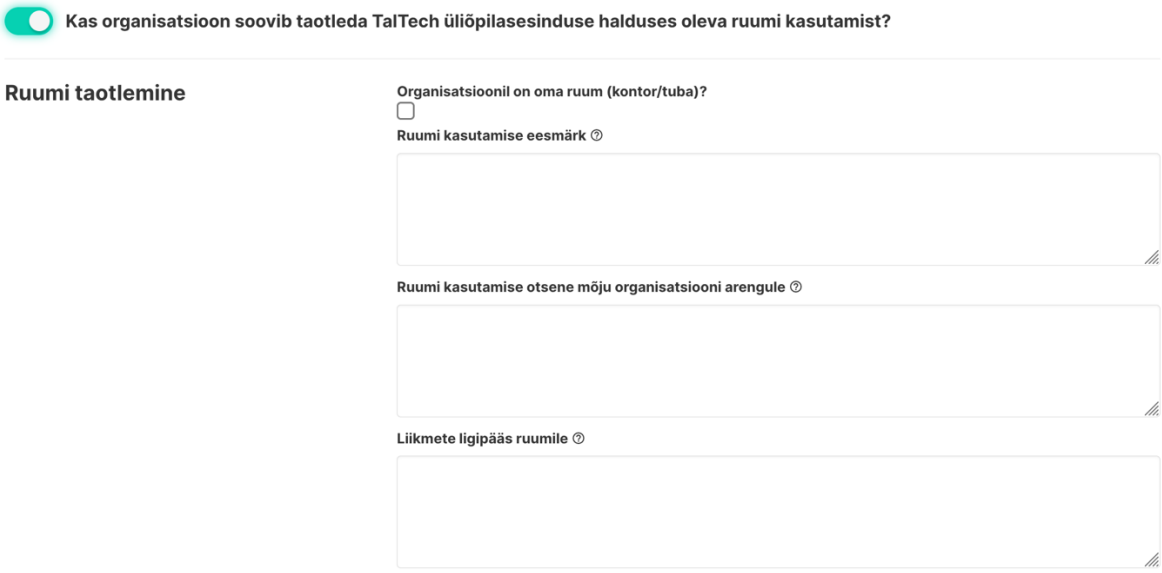
Peale rahalise toetuse, on võimalik vajaduse korral tudengiorganisatsioonidel aastatoetuste konkursi kaudu avaldada soovi ÜE poolt hallatavate ruumide igapäevaseks kasutamiseks.

Enne rahastusplatvormi oli vaja toetuse taotlejatel täita keerukaid ja mahukaid Exceli tabeleid ning rahastuskomisjon andis taotlustele tagasisidet e-posti teel. Nüüd on tabelite täitmine asendunud mugava kasutajaliidesega ning rahastuskomisjoni liikmetel on võimalik anda loodud platvormi kaudu tagasisidet konkursitele laekunud taotlustele [1, 2].

1.2 Probleemi püstitus

Üliõpilasesinduse rahastuskomisjoni liikmed on aastatoetuse konkursile laekunud taotlusi läbi töötades täheldanud seda, et kõik tudengiorganisatsioonid ei vaja alati aastaseks tegevuseks rahalisi vahendeid. Lõputöö kirjutamise ajahetkel on organisatsioonil võimalik aastatoetuse taotlust esitades märkida, et neil on vaja ainult ruume, kust saaks oma tööd edasi teha.

Lisaks ei ole võimalik aastatoetuse taotlust täites tudengiorganisatsioonil valida, millist ruumi või milliseid ruume soovitakse saada enda kasutusse. Joonisel 1 on kuvatõmmis rahastusplatvormi kasutajaliidesest, kus on näha milline näeb välja ruumi taotlusega seotud avaldus.



Joonis 1. Kuvatõmmis rahastusplatvormi ruumitaotlusest.[1]

Aastatoetuse taotluses oleva vormi täitmine tähendab, et üliõpilasesindus saab teada, et on organisatsioon, kes vajab enda tegevuseks ruumi. Järgmise sammuna selgitab ÜE omalt poolt välja, milliseid ruume on võimalik organisatsioonile pakkuda ja asub nendega kirjavahetusse. Hiljem märgib üliõpilasesindus endale üles, kus ruumis organisatsioon tegutseb ja mis ajal.

Sellest tulenevalt soovib ÜE, et rahastusplatvormi kaudu oleks võimalik organisatsioonidel avalduse täitmise ajahetkel valida endale sobivaimad ruumid. Selleks, et organisatsioon teaks, milliseid ruume on võimalik taotleda, on vaja seda infot ka rahastusplatvormil välja kuvada. Lõputöö kirjutamise ajahetkel ei ole üliõpilasesinduse poolt hallatavate ruumide nimekiri avalik ÜE välistele inimestele.

Üliõpilasesindus on avaldanud intervjuude käigus kaks peamist soovi – võimaldada organisatsioonidel rahastusplatvormi kaudu valida konkreetseid ruume üheks kalendriaastaks ning rahastusplatvormil välja näidata ÜE enda poolt hallatavaid ruume.

[1] https://rahastus.tipikas.ee/

1.3 Eesmärk

Käesoleva lõputöö eesmärgiks on luua olemasolevale rahastusplatvormi kasutajaliidesele mugavad ja kasutajasõbralikud vaated, mille abil on võimalik:

- luua eraldi konkurss, mille kaudu saavad tudengiorganisatsioonid endale taotleda ruumide kasutusõigust üheks kalendriaastaks;
- üliõpilasesinduse rahastuskomisjonide liikmetel eristada ruumidega seonduvaid taotlusi ja anda nendele tagasisidet;
- näha üliõpilasesinduse poolt hallatavaid ruume ja muuta nendega seonduvat infot;
- näha ja muuta ruumide kasutamise ajalugu, näiteks muuta ruumi kasutamise lõppkuupäeva.

Selleks, et oleks võimalik eelnevalt mainitud toiminguid teha läbi kasutajaliidese, tulevad ka täiendused rahastusplatvormi rakendusliidesele (ing. k. *Application Programming Interface*, lühendatult API) ja andmebaasile.

1.3.1 Lähtetingimused

Rahastusplatvormile loodavad täiendused peavad kasutama eelnevate arenduste ja lõputööde raames [1, 2] paika pandud tehnoloogiaid:

- rakendusliides on Java 11-l põhinev rakendus ning see kasutab *Spring Boot* 2.2 raamistikku,
- andmebaasi tehnoloogiaks on *PostgreSQL* 11,
- kasutajaliides on *Vue.js* 2 raamistikuga ehitatud *Single Page Application* (edaspidi SPA).

Rakendus- ja kasutajaliides kasutavad pikaajalise toega versioone ning neid ei ole uuendatud, sest see hõlmaks väliste teekide ja nende versioonide suuremamahulist uuendamist. Rakendusliides on ehitatud üles järgides kasutuslookeskse arhitektuuri põhimõtteid ning autor on otsustanud ühtse koodistiili säilitamise mõttes järgida seda mustrit edasi [1].

2 Metoodika

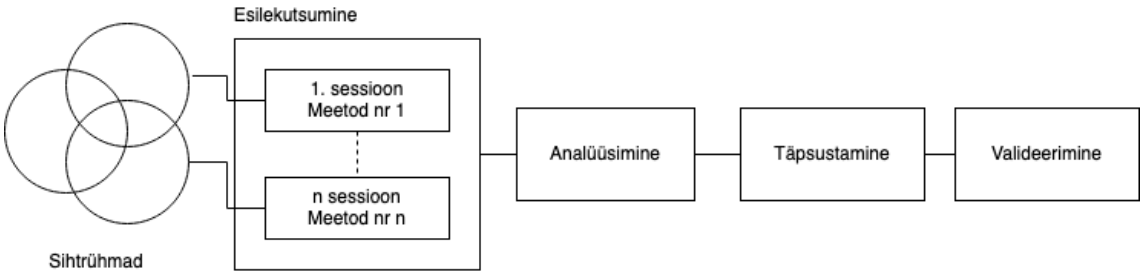
Käesolevas peatükis kirjeldab autor erinevaid meetodeid, kuidas on võimalik koguda nõudeid, valib sobivaima nõuete kogumise tehnika ja sõnastab need loodavatele funktsionaalsustele.

2.1 Nõuded rahastusplatvormile

Selleks, et saavutada lõputöö raames seatud eesmärgid, on vaja teostada täiendav analüüs seatud probleemile. Analüüs algab nõuete väljaselgitamisega. Käesolevas alampeatükis tutvustab autor erinevaid viise, kuidas on võimalik nõudeid koguda ning milliste meetodite kasuks ta otsustab.

2.1.1 Nõuete kogumise protsess

Joonisel 2 on kirjeldatud, millised on peamised tegevused nõuete väljaselgitamise protsessis – esilekutsumine (ing. k. *elicitation*), analüüsimine, täpsustamine ja valideerimine. Joonis 2 on koostatud Zheying Zhang artikli „Effective Requirements Development – A Comparison of Requirements Elicitation techniques“ [3] alusel.



Joonis 2. Nõuete kogumise protsess.

Nõuded kutsutakse esile konsulteerides sihtrühmadega (ing. k. *stakeholder*). Sihtrühmad ei ole ainult inimesed, nagu näiteks kliendid, lõppkasutajad või tarkvaraarendajad. Nad võivad viidata ka füüsilisele, organisatsioonilisele, kultuurilisele või seadusandlikule keskkonnale, kus soovitakse süsteemi kasutada [3].

Nõuete esile kutsumine võib toimuda mitme sessiooni käigus ja erinevaid tehnikaid kasutades. Peale sessiooni toimumist analüüsitakse saadud teadmisi, vajadusel korratakse protsessi uuesti, kuni on olemas selge arusaam probleemi valdkonnast ning ootustest süsteemile.

Leitud nõuete alusel koostatakse vastav dokumentatsioon. Nõuete dokumentatsioone on võimalik kirja panna:

- BRD-na ehk ärinõuete dokumendina (ing. k. *Business Requirements Document*).
- SRS-na ehk tarkvara nõuete spetsifikatsioonina (ing. k. *Software Requirements Specification*.
- Kasutusjuhtudena (ing. k. *use case*).
- Agiilses ehk kiiresti muutuvas kontekstis kasutajalugudena (ing. k. *user stories*) [4].

2.1.2 Nõuete kogumise tehnikad

Nõuete kogumise tehnikaid on võimalik liigitada nelja erinevasse kategooriasse vastavalt suhtlusvahenditele – vestlemismeetod (ing. k. *conversational method*), vaatlusmeetod (ing. k. *observational method*), analüüsimismeetod (ing. k. *analytic method*) ja sünteetiline ehk koostööl põhinev meetod (ing. k. *synthetic method*) [3].

Üks enim kasutatavaid meetodeid, mis kuulub vestlemise kategooriasse, on sihtrühmade intervjueerimine. Intervjueerija on tavaliselt inimene, kellel on üldised teadmised rakenduse teemavaldkonna (ing. k. *domain*) kohta. Intervjuu tulemusena on võimalik välja selgitada rakenduse jaoks vajalikud funktsionaalsused. Vestlemise kategooriasse kuuluvad ka näiteks fookusgruppidele suunatud töötoad kui ka ajurünnakud [3].

Vaatlemise kategooriasse kuuluvad meetodid annavad põhjaliku ülevaate rakenduse teemavaldkonnast. Vaatlemine tuleb kasuks siis kui on protsesse, mida on sihtrühmadel raske vestluse käigus kirjeldada. Selliseid nõudeid nimetatakse vaikivateks nõueteks (ing. k. *tacit requirement*). Vaatleja läheb teatud perioodiks jälgima sihtrühma töötegemist. Lisaks ta peab olema süsteemiga niivõrd palju kursis, et ta ei segaks sihtrühma igapäevast tööd [3].

Analüüsimise kategooriasse kuuluvaid meetodeid on vaja kasutada kui on vaja näiteks täiendavat sisendit vanade süsteemide ehk varasemalt kasutuses olnud lahenduste või sihtrühma õigusruumi kohta. Meetoditeks on näiteks varasemalt kogutud nõuete taaskasutamine, dokumentatsiooniga tutvumine ja sisuanalüüs (ing. k. *content analysis*) [3].

Sünteetilised meetodid ehk koostööl põhinevad meetodid on süstemaatiliselt kokku pandud erinevatest kategooriatest, mis moodustavad ühtse terviku. Sünteetiliste meetodite korral analüütikud ja sihtrühmade esindajad suhtlevad ning koordineerivad omavahel selleks, et jõuda ühise arusaamani nõuete osas. Sünteetilised meetodid on näiteks stsenaariumite esitlemine, kontekstipõhine uurimine, JAD/RAD sessioonid (ühine rakenduse arendus/kiire rakenduse arendus, ing. k. *Joint Application Development/Rapid Application Development*), prototüüpimine ja passiivsed tegevuste skeemid (ing. k. *storyboard*) [3]. Tegevuste skeem on meetod, mis kirjeldab visuaalselt, nagu näiteks illustratsioonide abil, järjestatud tegevusi, kuidas kasutaja suhtleb arvutiga [5].

2.1.3 Nõuete kogumise tehnika valimine

Sobivaima nõuete kogumise tehnika leidmiseks on vaja arvesse võtta järgnevat:

- Kas tegu on uue tootega või on see juba mõnda aega kasutuses olnud?
- Kes on toote sihtrühm(ad)?
- Kui hästi on organisatsioon kursis toote teemavaldkonnaga [3]?

Kui on tegu tootega, millel on selgelt defineeritud visioon, skoop ja eesmärk, on võimalik nõudeid välja selgitada peatükis 2.1.1. viidatud vestlemise meetodiga, näiteks sihtrühmade intervjueerimine, või sünteetilisele meetodiga, näiteks prototüüpimine [3].

Vestlemise kategooriasse kuuluvaid meetodeid on võimalik kasutada igal ajahetkel, kui on vaja nõudeid paika panna ja sihtrühmaks on inimesed, mitte keskkond [3].

Vaatlusel põhinevad meetodid on tõhusad, kui süsteemi arendamine on algusjärgus ning on vaja saada parem arusaam füüsilisest, organisatsioonilisest, seadusandlikust ja kultuurilisest keskkonnast, kus valmiv toode hakkab kasutust leidma [3].

Analüüsimisele tuginevad meetodid koguvad vajalikke teadmisi tuginedes ekspertidele ning eksisteerivatele dokumentidele. Sellised meetodid on mõistlik võtta kasutusele siis, kui on olemas dokumentatsioon ja eksperdid, kel on olemas teadmised toote teemavaldkonna kohta. Lisaks analüüsimisele tuginevate meetodite alusel on võimalik taaskasutada senini omandatud teavet [3].

Käesoleva arendusprojekti raames on peamiseks sihtrühmaks Tallinna Tehnikaülikooli üliõpilasesindus, kellel on olemas selge ülevaade enda organisatsioonis toimuvatest protsessidest. Lisaks eksisteerivad varasemalt koostatud bakalaureusetööd, kus on selgitatud rahastusplatvormi andmebaasi struktuuri ja rakendus- ja kasutajaliidese ülesehitust. Lisaks on käesoleva bakalaureusetöö üks juhendajatest varasemalt koostatud tööde autor. Sellest tulenevalt on otsustanud autor kombineerida igast kategooriast järgnevad meetodid:

- sihtrühma intervjueerimine;
- dokumentatsiooni analüüs;
- sisuanalüüs ehk varasema koodibaasiga tutvumine;
- prototüüpimine.

2.1.4 Nõuete sõnastamine

Peatükis 2.1.1 tõi autor välja, et nõudeid on võimalik kirja panna ärinõuete dokumendina, tarkvaranõuete spetsifikatsioonina, kasutusjuhtudena või kasutajalugudena.

Ärinõuete dokument on koostatud eelkõige ärilisest perspektiivist, kuna see sisaldab projekti ärilahenduse üksikasju [6]. Ärinõuete dokumendiga võrreldes on tarkvaranõuete spetsifikatsiooni koostamise ajahetkel mõeldud eelkõige tarkvarale endale. Tarkvaranõuete spetsifikatsioonis tuuakse välja, mida peab tarkvara tegema selleks, et ärivajadusi rahuldada [7].

Kasutusjuhtude sõnastamisel tuuakse välja rakenduses eesmärgi saavutamiseks vajalik tegevuste jada [5]. Kasutajaloo formaadi puhul tuuakse välja üks konkreetne funktsioon, mida teatud kasutajatüüp soovib teha [5].

Autor viis läbi nõuete väljaselgitamiseks intervjuu üliõpilasesinduse esindajaga. Selleks, et nõudeid valideerida, sõnastatakse need kasutajalugude stiilis, andmaks üliõpilasesindusele paremini mõista, mis toiminguid kasutaja saab rakenduses teha. Intervjuu tulemusena sõnastab autor nõuded järgnevalt:

1. Üliõpilasesinduse esindajana tahan luua ruumide taotlemise konkursi selleks, et saaksin rahalisest toetuse jagamisest eraldiseisvalt otsustada, millised organisatsioonid saaksid meie poolt hallatavate ruumide kasutusõigusi.

2. Üliõpilasesinduse esindajana tahan ülevaadet ruumide kasutamise ajaloo kohta selleks, kui kunagi peaks tulema ruumide kasutamise kohta audit.

3. Tudengiorganisatsiooni esindajana tahan saada ülevaadet, millised ruumid on üliõpilasesinduse poolt hallatavad selleks, et leiaksin minu organisatsiooni vajadustele sobivaimad ruumid.

4. Tudengiorganisatsiooni esindajana tahan anda üliõpilasesindusele teada, milliseid ruume soovin oma organisatsioonile, et meie töö saaks jätkata.

2.1.5 Nõuete valideerimine

Selleks, et kindel olla, kas autor lahendab õiget probleemi, on ta otsustanud läheneda visuaalselt ning kasutada selleks prototüüpimist. Prototüüpimise tehnika abil on võimalik esile kutsuda täiendavaid nõudeid rakendusele või olemasolevaid nõudeid valideerida. Lisaks annab see sihtrühmale hea ülevaate, kuidas rakendus hakkab välja nägema [3]. Prototüüpimisel on kolm võimalikku tehnikat – sõrestikmudel (ing. k *wireframe*), makett (ing. k. *mockup*) ja prototüüp [8].

Sõrestikmudelid näitavad, kuidas hakkab rakenduses välja nägema kasutajaliideses kasutatavate elementide asetus ning milliseid andmeid kasutajale kuvatakse. Sõrestikmudeli puhul ei ole vaja rõhku panna kasutajaliidese välimusele, nagu näiteks värvidele ja tüpograafiale. Nende abil on võimalik välja selgitada, mis on toote skoop ja vajadusel täpsustada ärinõudeid. Lisaks nende loomine nõuab vähe aega ja ressurssi [8].

Makett annab ülevaate selle kohta, milline on toote lõplik välimus. Nende loomiseks kasutatakse digitaalse disaini jaoks vajalikke tööriistu, näiteks Adobe XD[1]. Makette kasutatakse, kui toote arendus on disainimise etapis. Võrreldes sõrestikmudelitega nõuavad nad rohkem ajalist ressurssi [8].

Prototüübi eesmärgiks on demonstreerida toote lõplikku välimust, testida funktsionaalsust ning valideerida ideed ennast. Prototüüpe kasutatakse toote testimise faasis ning testimisse on kaasatud ka lõppkasutajad. Nende loomiseks kasutatakse samuti digitaalseid tööriistu, nagu näiteks Adobe XD, ning kõigist kolmest tehnikast nõuab prototüübi loomine kõige rohkem aega ja ressurssi [8].

Autor on otsustanud sõnastatud nõuete valideerimiseks luua sõrestikmudelid eelkõige seetõttu, et olla kindel selles, kuidas kasutajaliidese rakendus hakkab andmeid välja kuvama ja milliseid välju peab lõppkasutaja täitma. Lisaks saab autor sõrestikmudelit kasutades välja tuua, kuidas kasutaja saab mingisuguseid valikuid teha rakenduses ja milliste elementide abil kommunikeeritakse võimalikke tegevusi edasi.

Sõrestikmudeleid on võimalik joonistada kasutades paberit ja pliiatsit [8]. Autor otsustas kasutada rakendust, mis võimaldab virtuaalselt jagada tehtud tööd ja teostavad muudatused on koheselt näha. Kaks kõige populaarsemat prototüüpimise tööriista on Adobe XD ning Figma[2] [9]. Autor otsustas sõrestikmudelite loomiseks kasutada Figmat, kuna tal on sellega varasem tööalane kogemus, see on tasuta kasutamiseks ja visuaalide nägemiseks ei ole vaja eelnevalt kasutajaks registreerida.

2.1.6 Sõrestikmudelid

Käesolevas alampeatükis tuuakse välja sõrestikmudelid, mis toetavad peatükis 2.1.4 sõnastatud nõudeid. Autor näitas üliõpilasesindusele jooniste abil, kuidas võiks rahastusplatvormi tehtavad uuendused välja näha ning kohtumise käigus viidi sõrestikmudelitele täiendavad muudatused sisse. Antud töös on välja toodud lõplikud mudelid.

[1] https://www.adobe.com/products/xd.html

[2] https://www.figma.com/

Peatükis 1.2 välja toodud joonisel 1 on näha, milline näeb välja ruumi taotluse vaade lõputöö kirjutamise ajahetkel. Joonisel 3 on näha loodava ruumi taotluse vaate sõrestikmudel, mis rahuldab peatükis 2.1.4 sõnastatud 3. nõuet.

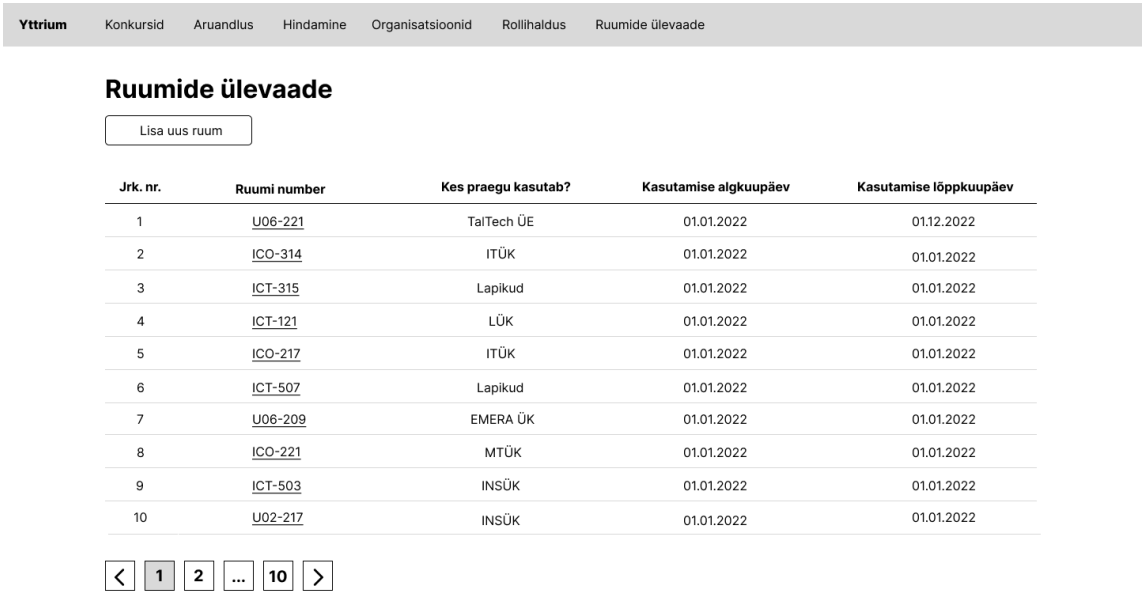
Joonis 3. Sõrestikmudel ruumi taotluse vaatele.

Ruumi taotluse vaates on otsustanud autor kasutada märkeruute (ing. k. *checkbox*) seetõttu, et tegu on vormi väljaga, kus süsteem soovib saada vastuseks jah või ei. Märkeruudu kasutamine annab lõppkasutajale teada, et tegu on elemendiga, mida saab nii-öelda sisse ja välja lülitada [10]. Vastavalt sellele, kas väärtus on olemas, kuvatakse kuvatakse järgnevas plokis välja rippmenüüd.

Ruumitaotluse vaates enamike väljade kirjelduse kõrval on väike ikoon, kus on näha ringiga ümbritsetud küsimärki, mis tähistab vihjemulli (ing. k *tooltip*). Vihjemullid on

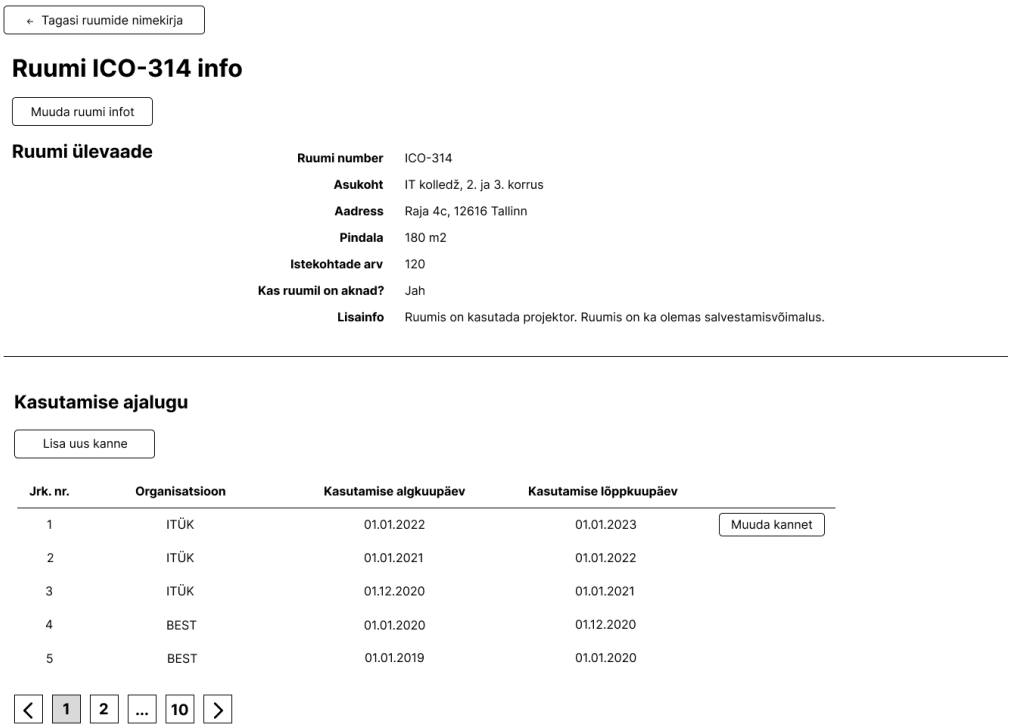
vajalikud, et anda lõppkasutajale paremini edasi seda, millist väärtust oodatakse täidetavasse välja. Osad väljad on täidetud näidisväärtusega või juhisega, mida on vaja teha. Näiteks osade väljade puhul on tegu rippmenüüdega – peale selle, et on väljale lisatud nooleke, on sellel ka täiendav juhend. Kui tegu on ebaintuitiivse kasutajaliidesega, on vajalik lõppkasutajale välja kommunikeerida see, mida temast oodatakse [10].

Peatükis 2.1.4 välja toodud 2. nõudest tulenevalt sai loodud sõrestikmudel, kus on välja toodud, kuidas saab olema rahastusplatvormis välja kuvatud ÜE poolt hallatavate ruumide nimekiri. Mudel on välja toodud joonisel 4.



Joonis 4. Sõrestikmudel ruumide nimekirja vaatele.

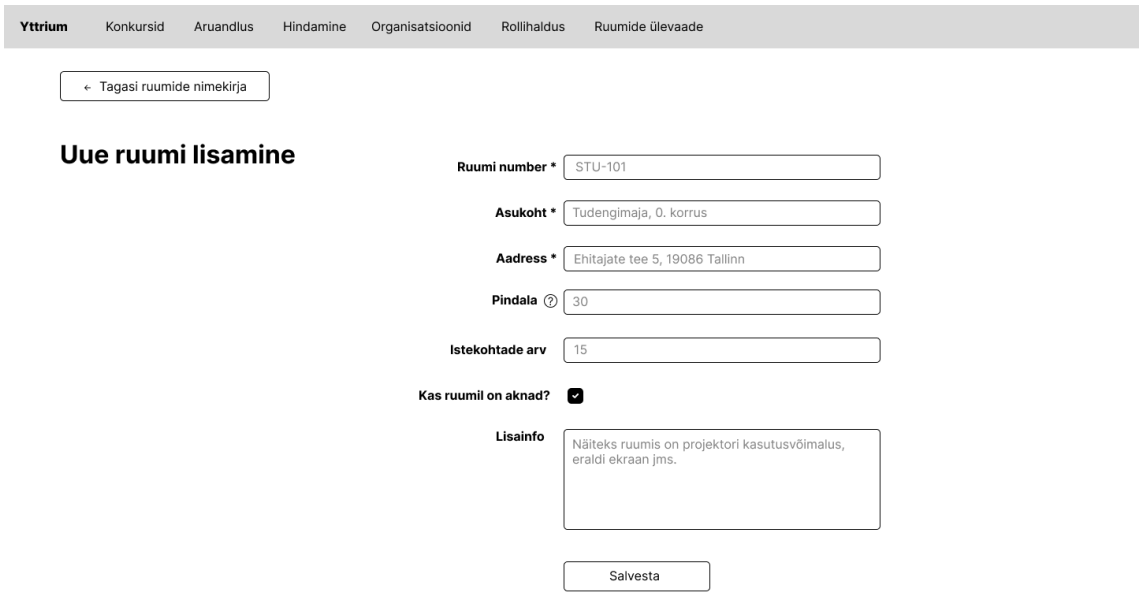
Joonisel 4 oleval sõrestikmudelil on näha, et ruumi numbrid on alla joonitud. Sedasi saab lõppkasutaja aru, et tegu on lingiga, millele saab peale vajutada. Selleks, et kasutaja ei jääks kasutajaliideses vaadet lõpmatult alla kerima, kuvatakse korraga välja kuni 10 ruumi. Peale vajutades suunatakse lõppkasutaja ruumi detailvaatesse. Nuppu „Lisa uus ruum“ kuvatakse välja vaid rahastusplatvormi sisse logitud üliõpilasesinduse liikmele. Ruumi detailvaade on välja toodud joonisel 5. Ruumi detailvaadet on võimalik näha sisse logimata kasutajal.



Joonis 5. Sõrestikmudel valitud ruumi detailvaatele.

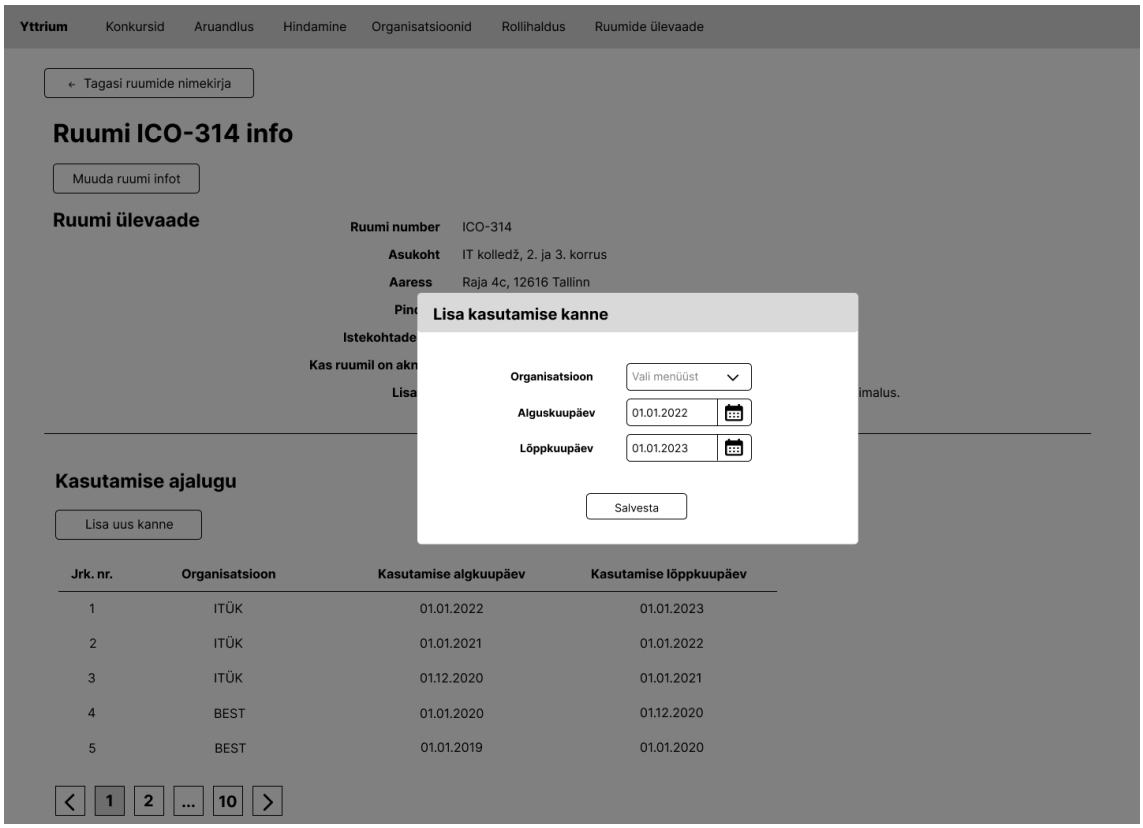
Ruumi detailvaate sõrestikmudelil on kaks plokki – ruumi ülevaade ning ruumi kasutamise ajalugu. Ruumi ülevaate plokis kuvatakse välja detailsem info ning ruumi kasutamise ajaloo juures on ruumide kasutamise ajalugu koos organisatsiooni infoga. Sisse logitud üliõpilasesinduse liikmetele kuvatakse välja „Muuda ruumi infot“, „Lisa uus kanne“ ja “Muuda kirjet” nuppe. „Muuda kirjet“ nuppu kuvatakse välja ainult siis, kui ruumi kasutamise lõppkuupäev ei ole minevikus.

„Muuda ruumi infot“ ja „Lisa uus ruum“ nupud suunavad sisse logitud üliõpilasesinduse liikme vaatesse, kus on vaja täita ankeet (joonis 6). Kui on tegu ruumi info muutmisega, on ankeedis väljad eelnevalt täidetud ning pealkirjas kuvatakse ruumi number välja. Muul juhul jäävad väljad tühjaks. Kohustuslike väljade nimetuste juures on tärn. „Pindala“ välja juures on vihjemull selleks, mis selgitab ära, et oodatav numbriline väärtus on ruutmeetrites.

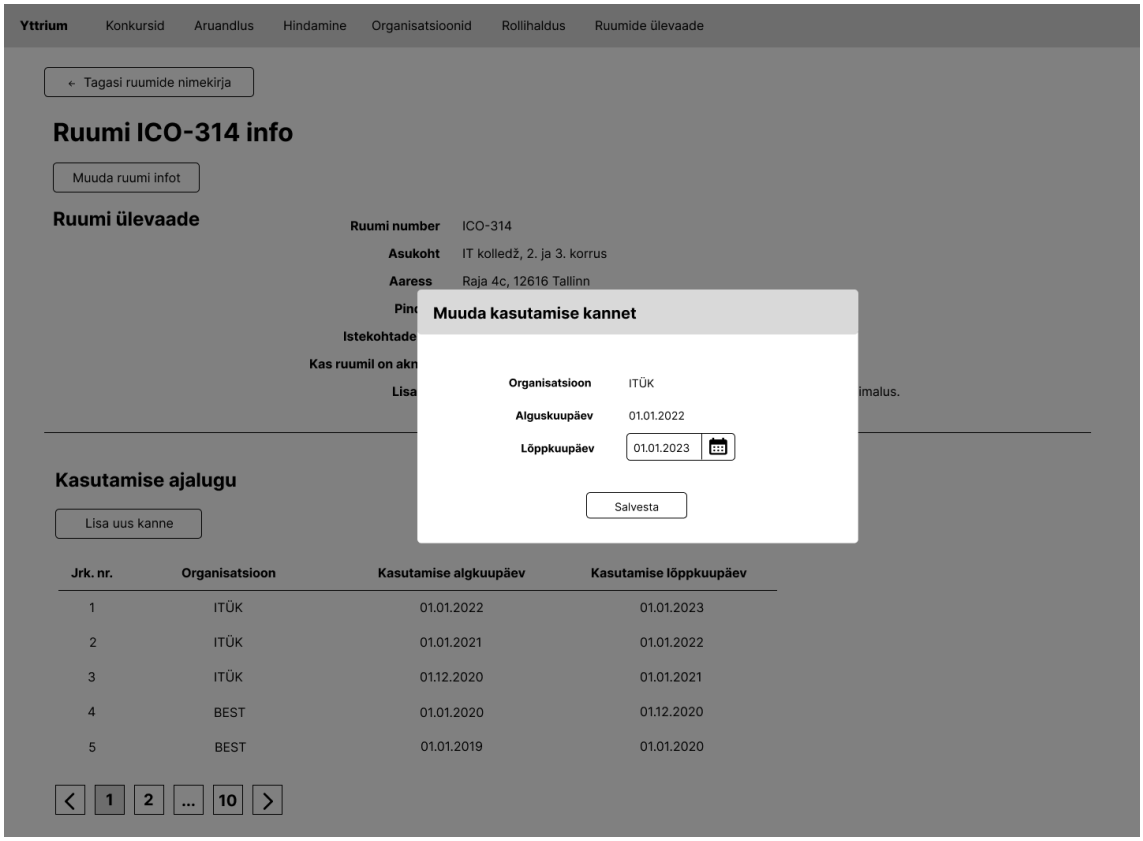


Joonis 6. Sõrestikmudel ruumi lisamise ja muutmise ankeedile.

„Lisa uus kanne“ ja „Muuda kirjet“ nupud avavad modaalaknad (ing. k. *modal window*, joonis 7 ja 8), kus on võimalik üliõpilasesinduse töötajal lisada või muuta kirjet ruumi kasutamise kohta. Modaalakent tasub kasutada siis, kui ei ole palju välju, mida lõppkasutaja peab täitma [10]. Ruumi kasutusajaloo muutmise peamine eesmärk võimaldada üliõpilasesinduse liikmel muuta lõppkuupäeva, juhul kui organisatsioon lõpetab ruumis tegutsemise ruumi kasutusõiguse lõppkuupäevast varem.



Joonis 7. Ruumi kasutusajaloo lisamine.



Joonis 8. Ruumi kasutusajaloo muutmine.

3 Praktiline osa

Käesolevas peatükis kirjeldab autor, millised muudatused viib ta sisse olemasolevasse koodibaasi, võttes aluseks peatükis 2.1.4 sõnastatud nõuded. Muudatusi sisse viies on vaja arvestada peatükis 1.3.1 seatud lähtetingimusi.

3.1 Muudatused andmebaasis

Käesolevas alampeatükis kirjeldab autor, milliseid muudatusi ta viib sisse andmebaasi tasandil.

3.1.1 Muudatused seoses avaldustega

Lõputöö kirjutamise ajahetkel kõikide konkursitüüpidega seotud avalduste info salvestatakse andmebaasis „application“ tabelisse. Sellest tulenevalt on andmebaasis avaldustega seotud tabeli veergude arv võrdlemisi suureks kasvanud – „application“ tabelis on kokku 33 veergu. Mainitud tabeli algne struktuur on leitav „Lisa 2“ alt. Selleks, et oleks võimalik näha ka ruumitaotlustega seonduvat infot, oleks vaja lisada antud tabelisse veerge juurde.

Avaldustega seotud tabelis ei ole kõik väljad seotud kõikidele konkursitele laekuvate avaldustega. Sellest tulenevalt võib tekkida mälu raiskamise anomaalia. Mälu raiskamise anomaalia puhul jääb tabelis grupp väljasid tühjaks ning reserveeritakse ilma asjata ilma asjata tühja mäluruumi. Anomaaliast vabanemiseks on vaja normaliseerida andmemudelit [11]. Käesoleva lõputöös on vaja normaliseerida avaldustega seotud andmebaasi tabelit.

Andmemudel on normaliseerimata, kui temas leidub tabeleid, milles on korduv-gruppe [11]. „application” tabelis saab eristada andmete gruppe rahastusplatvormi konkursitüüpide alusel. Sellest tulenevalt on otsustanud autor “application” tabelist luua kolm täiendavat tabelit, mis hakkavad esindama konkursitüübist olenevalt avalduse sisu.

Loodavate tabelite semantika on välja toodud tabelis 1. Avaldustega seotud andmebaasi tabeli uus struktuur on toodud välja tabelis 2. Tabelites 3-5 kirjeldatakse lähemalt, mida loodavate andmebaasi tabelite veergude väärtused kirjeldavad.

Tabel 1. Loodavate ja kohandatavate tabelite semantika.

Tabeli nimi	**Semantika**
application	Tabelis hoiustatakse konkursitele laekuvate avalduste ühisosa.
room_application_content	Tabelis hoiustatakse ruumi kasutusõiguste taotlustega seotud teavet.
project_application_content	Tabelis hoiustatakse projektitaotlustega seotud teavet.
annual_application_content	Tabelis hoiustatakse aastatoetuse taotlustega seotud teavet.

Tabel 2. Tabeli "application" struktuur.

Veeru nimi	**Andmetüüp**	**Selgitus**
id	UUID	Primaarvõti.
contest_id	UUID	Välisvõti. Näitab, millise konkursiga on avaldus seotud.
status	VARCHAR(50)	Laekunud avalduse staatus. Avaldus võib olla näiteks arhiveeritud, positiivse või negatiivse otsuse saanud.
type	VARCHAR(50)	Konkursi tüüp, millele avaldus laekus.
applicant_type	VARCHAR(50)	Avalduse esitaja tüüp. Selleks võib olla kas tudengiorganisatsioon või eraisik.
applicant_id	INTEGER	Avalduse esitaja ID. Selle väärtus tuleb Auth0-st[1]. Väli on tehnilise vajadusega.
applicant_name	VARCHAR(255)	Avalduse esitaja nimi.
organization_id	BIGINT	Välisvõti. Avalduse esitaja organisatsioon.
applicant_desc	TEXT	Avalduse esitaja kirjeldus. Kasutajaliideses vaba tekstiväli.
account_owner_name	VARCHAR(255)	Pangakonto omanik.
account_number	VARCHAR(50)	Pangakonto number.
address	TEXT	Avalduse esitaja aadress – võib olla organisatsiooni aadress või eraisiku enda oma.

[1] https://auth0.com/

phone	VARCHAR(20)	Avalduse esitaja või organisatsiooni telefoninumber.
email	VARCHAR(255)	Avalduse esitaja e-posti aadress, mille kaudu on võimalik temaga ühendust võtta.
home_page	VARCHAR(255)	Projekti või organisatsiooni koduleht.
personal_code	VARCHAR(20)	Avalduse esitaja isikukood. Kui esitajaks on organisatsioon, siis organisatsiooni esindaja enda isikukood.
files	JSONB	Failide info JSON formaadis, mille abil on võimalik hiljem küsida Dropbox'i[1] API kaudu õiged failid.

Tabel 3. Tabeli "room_application_content" struktuur.

Veeru nimi	**Andmetüüp**	**Selgitus**
application_id	UUID	Primaarvõti ja välisvõti. Viitab „application“ tabelile, kus on taotlusega seotud lisainfo.
has_own_room	BOOLEAN	Väärtus näitab, kas tudengiorganisatsioonil on oma ruum olemas.
own_room_location	VARCHAR(255)	Tudengiorganisatsiooni ruumi asukoht. Väärtus võib olla näiteks "ICO-314, IT kolledži 2. ja 3. korrus“.
current_rooms	JSONB	Kui tudengiorganisatsioonil on avalduse esitamise ajahetkel kasutuses ruumid, mis on üliõpilasesinduse poolt välja antud, hoiustatakse seda infot siin veerus.
main_room	JSONB	Avalduses valitud põhiline ruum, mida soovitakse üliõpilasesinduselt taotleda.
additional_rooms	JSONB	Avalduses valitud lisaruumid, mida soovitakse üliõpilasesinduselt taotleda.
purpose_desc	TEXT	Ruumide kasutamise eesmärk. Vabateksti väli.
impact_desc	TEXT	Organisatsioon kirjeldab, millist mõju avaldab organisatsiooni tegevusele ruumi kasutusõiguse saamine. Vabateksti väli.
access_desc	TEXT	Organisatsioon kirjeldab, millised liikmed nende koosseisust saavad ligipääsu ruumidele, mis kellaaegadel jms. Vaba tekstiväli.

[1] https://www.dropbox.com/

member_list_file	JSONB	Üleslaetava faili info. Faili sisuks on organisatsiooni liikmete koosseis.

Tabel 4. Tabeli "project_application_content" struktuur.

Veeru nimi	**Andmetüüp**	**Selgitus**
application_id	UUID	Primaarvõti ja välisvõti. Viitab „application“ tabelile, kus on taotlusega seotud lisainfo.
project_team	JSONB	Projektiga seotud meeskonna info. Tuuakse välja, millises rollis projekti meeskonnaliige on ja mis taustaga ta on.
project_name	VARCHAR(255)	Projekti nimi.
start_date	DATE	Projekti alguskuupäev.
end_date	DATE	Projekti lõppkuupäev.
project_desc	TEXT	Projekti tutvustus.
target_audience_desc	TEXT	Projekti sihtgrupi kirjeldus ehk keda oodatakse projektile osalema.
goals_desc	TEXT	Projekti raames saavutatavate eesmärkide kirjeldus.
sponsors	JSONB	Projekti toetajate nimekiri. Tuuakse välja ka see, millisel moel aitab toetaja projekti toimumisse kaasa.
tasks	JSONB	Projekti korraldusega seotud tööde nimekiri.
additional_value_desc	TEXT	Kirjeldatakse, millist lisaväärtust planeeritav projekt loob tudengkonnale.
budget_justification	TEXT	Kirjeldatakse ära, kuidas planeeritakse taotletavat rahalist toetust ära kasutada projekti korraldusel. Vabateksti väli.

Tabel 5. Tabeli "annual_application_content" struktuur.

Veeru nimi	**Andmetüüp**	**Selgitus**
application_id	UUID	Primaarvõti ja välisvõti. Viitab „application“ tabelile, kus on taotlusega seotud lisainfo.
action_plan_file	JSONB	Üleslaetava faili info. Faili sisuks on organisatsiooni aastane tegevuskava.
member_list_file	JSONB	Üleslaetava faili info. Faili sisuks on organisatsiooni liikmete nimekiri.

projects	JSONB	Üleslaetava faili info. Kirjeldab ära, milliseid projekte organisatsioon korraldab ühe kalendriaasta jooksul.
room_application	JSONB	Organisatsiooni sooviavaldus ruumi jaoks.
additional_value_desc	JSONB	Kirjeldatakse ära, kuidas ja millist lisaväärtust organisatsiooni tegevuskava loob tudengkonnale.

3.1.2 Muudatused seoses ruumihaldusega

Lõputöö kirjutamise ajahetkel puudub võimalus salvestada andmebaasi ÜE poolt hallatavate ruumide ja nende kasutamisega seotud infot. Seetõttu loob autor juurde kaks tabelit – „room“ ja „organization_room“.

Tabelis „room“ hakatakse hoiustama ÜE poolt hallatavate ruumide üldinfot. Tabeli struktuur on lahti kirjeldatud tabelis 6. Tabelisse „organization_room“ salvestatakse maha teave, milline ruum on organisatsiooni poolt mingil ajahetkel kasutuses olnud. Tabelite struktuurid on kirjeldatud lahti tabelites 6-7.

Tabel 6. Tabeli "room" struktuur.

Veeru nimi	**Andmetüüp**	**Selgitus**
id	BIGSERIAL	Primaarvõti.
room_number	VARCHAR(50)	Ruumi number. Ruumi number võib olla näiteks „STU-305“ kui ka „Akadeemia tee 5-008a ja 5-008b“.
location	VARCHAR(255)	Täpsem asukoht, kus ruum hoones paikneb. Väärtus võib olla näiteks „Tudengimaja, 3. korrus“.
address	VARCHAR(255)	Hoone aadress, kus ruum paikneb.
area	NUMERIC	Ruumi pindala numbriline väärtus, ilma mõõtühikuta. Väärtus võib olla näiteks 50,1.
capacity	INTEGER	Ruumi istekohtade arv või mitu inimest see keskmiselt mahutab.
has_windows	BOOLEAN	Näitab ära, kas ruumis on aknad.
additional_information	TEXT	Täiendavad märkmed ruumi enda kohta. Vabateksti väli.

Tabel 7. Tabeli "organization_room" struktuur.

Veeru nimi	Andmetüüp	Selgitus
id	BIGSERIAL	Primaarvõti.
organization_id	BIGSERIAL	Välisvõti. Viitab „organization“ tabelile.
room_id	BIGSERIAL	Välisvõti. Viitab „room“ tabelile.
active_from	DATE	Kuupäev, mis ajahetkest alatest on organisatsioon ruumis tegutsemas.
active_to	DATE	Kuupäev, millal organisatsioon lõpetab ruumis tegutsemise.

3.2 Muudatused kasutajaliideses

Kasutajaliidesesse sisse viidavate muudatuste aluseks on peatükis 2.1.6 loodud sõrestikmudelid. Kasutajaliideses oli vaja ruumide konkurssidega seoses olemasolevat ruumitaotluse vaadet täiendada, lisades taotlusele juurde vajalikke välju ja näidata välja ainult vajalikke samme.

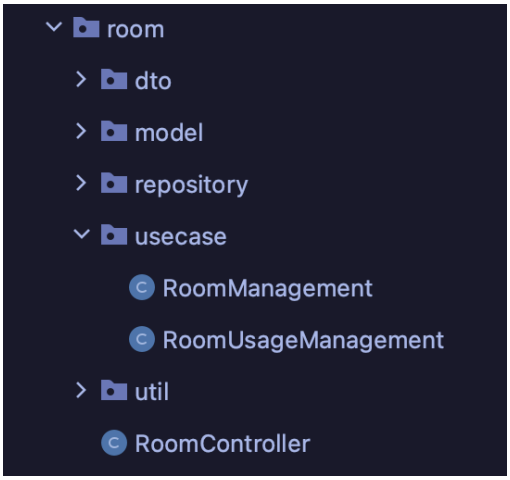
Ruumide haldusega seoses tegi autor sõrestikmudeleid järgides uued vaated, mille kaudu on sisse logimata kasutajal võimalik näha kõiki üliõpilasesinduse halduses olevaid ruume ning iga ruumi kohta detailsemat infot, sealhulgas ka kasutamise ajalugu. Sisse logitud üliõpilasesinduse töötajal on võimalik süsteemi lisada ja muuta olemasolevaid ruume. Samuti on tal ka võimalik lisada ning muuta ruumi kasutusajalooga seotud kandeid. Kasutajaliidesesse lisati juurde ka funktsioonid, mille abil on võimalik teha peatükis 3.3 loodud API teekonna pihta päringuid.

3.3 Muudatused rakendusliideses

Käesolevas alampeatükis kirjeldab autor, kuidas rakendusliidest on senini arendatud ja milliseid muudatusi tal oli vaja teostada selleks, et üliõpilasesindusel oleks võimalik ruume hallata ja luua konkursse ruumide kasutusõiguste taotlemiseks.

Krõõt Grete Mänd tegi oma lõputöö [1] raames rakendusliidese, järgides kasutuslookeskse ja REST API arhitektuuri põhimõtteid ning kolmekihilise API arhitektuurimustreid. Raamatus „Clean architecture“ on välja toodud, et kasutuslookeskse arhitektuuri peamiseks ideeks on esile tuua rakenduse funktsionaalsused ja nõutud

käitumismustrid. Rakenduse funktsionaalsus on struktuurile peale vaadates selge ja arusaadav. Arhitektuurist kajastub, mis toiminguid on võimalik läbi rakenduse teha. See tähendab, et erinevad kasutuslood on realiseeritud klassidena, mis on kohe nähtavad pakettide struktuuris ning klassi nimi väljendab selgelt, mis on klassi eesmärk nagu näha joonisel 9 [12].



Joonis 9. Näide kasutajalookesksest struktuurist

Rahastusplatvormi rakendusliideses on kolm kihti: esituskiht, äriloogika kiht ja andmepääsu kiht. Kõik kasutusjuhtumid on kokku kogutud äriloogika kihti. Äriloogika kihis analüüsitakse ja tehakse otsuseid ning see on esituskihi ja andmepääsu kihi vahel. Esituskihis on klassid, kuhu on kokku kogutud meetodid, mis on seotud kasutajate toimingutega. Andmepääsu kiht tegeleb andmebaasi või failisüsteemi suhtlusega [1].

Rahastusplatvormi rakendusliideses on järgnevad kasutajarollid [1]:

1. ADMIN – Rakenduse administraator, kellel on ligipääs kõigele. Tal on õigus luua uusi kasutajaid ning anda kasutajatele täiendavatele õigusi.
2. MANAGER – Kasutajad, kes on eelkõige ÜE töötajad.
3. Külaline - Sisse logimata kasutajad.

Autor kirjeldab järgnevates alampeatükkides, milliseid muudatusi on tal vaja kihtides teostada.

3.3.1 Muudatused seoses ruumide kasutusõiguse taotlemisega

Varasemalt sai välja toodud, et rakendusliides sai üles ehitatud järgides kasutuslookeskse arhitektuuri põhimõtet ja kolmekihilise API arhitektuuri mustrit. Rakenduse ärikihis on mitu Java klassi, mis kirjeldavad rahastusplatvormi avaldustega seotud kasutusjuhte, nagu näiteks avalduste pärimine ja nende muutmine.

Äriloogika kiht pöördub andmepääsu kihi poole, küsides avaldustega seonduvat infot. Enne küsiti kogu info välja ühest tabelist, milleks oli „application“. Sõltuvalt sellest, mis on konkursi tüüp, on vaja teave edaspidi välja küsida vastavalt „room_application_content“, „project_application_content“ või „annual_application_content“ tabelitest.

Raamatus “Clean Architecture” viidatakse sellisele printsiibile nagu “The Dependency Inversion Principle” – sõltuvuse inversiooni printsiip. Printsiip ütleb, et paindlikud süsteemid on need, mille lähtekoodi sõltuvused viitavad vaid abstraktsioonidele, mitte millelegi konkreetsele. Java koodi mõistes tähendab see seda, et viidatakse abstraktsetele klassidele või liidestele (ing. k. *interface*). Selleks, et seda printsiipi realiseerida, ei tohi viidata muutuvatele, konkreetsetele klassidele ja nendest klasse tuletada. Lisaks ei tohi konkreetseid funktsioone üle kirjutada ja mainida midagi muutuvat ning konkreetset [12].

Selle probleemi käsitlemiseks on abiks tehase meetodite muster (ing. k. *factory pattern*). Kui objektide loomisel kasutatakse konkreetseid klasse, siis tehase mustri meetodite tagastustüüp on tavaliselt deklareeritud kas abstraktse klassi või liidese kujul [12, 13]. Rahastusplatvormis kasutusjuhtumi klass pöördub tehase meetodi poole, mis tagastab konkursitüübile vastavalt äriloogikat peegeldava klassi (ing. k. *service class*). Äriloogikat peegeldavate klasside kiht ehk *service* kiht juhib transaktsioone [14].

Tehase meetodi programmikoodi näidis on välja toodud joonisel 10.

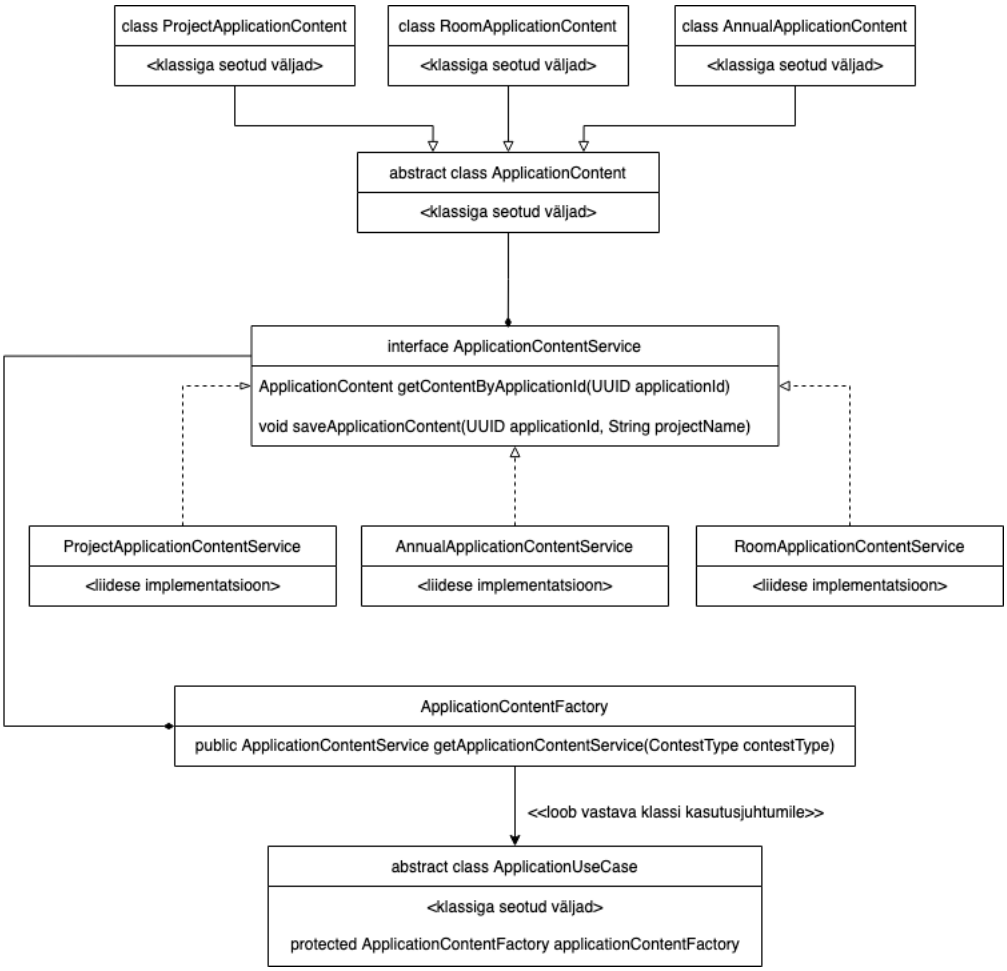
```
@Component
@AllArgsConstructor
@NoArgsConstructor
public class ApplicationContentFactory {
    @Resource
    private ProjectApplicationContentService
projectApplicationContentService;
    @Resource
    private AnnualApplicationContentService
annualApplicationContentService;
    @Resource
    private RoomApplicationContentService
roomApplicationContentService;

    public ApplicationContentService
getApplicationContentService(ContestType contestType) {
        switch (contestType) {
            case SMALL:
            case BIG:
                return projectApplicationContentService;
            case ANNUAL:
                return annualApplicationContentService;
            case ROOM:
                return roomApplicationContentService;
            default:
                throw new IllegalArgumentException("Contest type does
not exist: " + contestType.name());
        }
    }
}
```

Joonis 10. Service klassi pärimise factory programmikoodi näide.

Avalduse sisuga tegeleva *service* klassi pärimise puhul on vaja teada, millise konkursitüübiga on tegu. Konkursitüüp antakse tehase meetodisse kaasa ja vastavalt sellele ka tagastatakse soovitud *service*. Autor on loonud liidese (ing. k. *interface*) ApplicationContentService, kus on kõrgel tasemel defineeritud, milliseid toiminguid saab avalduse sisuga teha, nagu näiteks avalduse sisu pärimine või avalduse sisu kirje lisamine andmebaasi. Vastavalt igale konkursitüübile saab olema kolm *service* klassi, mis hakkavad implementeerima ApplicationContentService liidest – ProjectApplicationContentService, AnnualApplicationContentService ja RoomApplicationContentService. Juhul, kui on tegu tundmatu konkursitüübiga, tagastatakse vastava sõnumiga erind.

Rahastusplatvormi ärikihis olevad avaldustega seotud kasutusjuhud on seotud abstraktse klassiga ApplicationUseCase. Selleks, et kõik avaldustega seotud kasutusjuhtude klassid

saaksid ligi tehase meetodile, lisatakse juurde vastav väli ApplicationUseCase klassile. Joonisel 8 tuuakse esile, kuidas on omavahel seotud avalduste sisu esindavad klassid, kasutusjuhu klass, tehase meetod ja *service* klassid.



Joonis 11. Avalduse sisu, kasutusjuhu, *service* klasside ja tehase meetodi seotus.

Konkursitele laekunud avaldusi töötab läbi rahastuskomisjon. Kui rahastuskomisjon on ruumi kasutusõiguse taotlemise konkursile laekunud avalduse heaks kiitnud, salvestatakse peatükis 3.3.2 välja toodud RoomUsageManagement kasutusjuhu klassis implementeeritud meetodi abil „organization_room“ tabelisse vastav kirje.

3.3.2 Muudatused seoses ruumide haldusega

Selleks, et üliõpilasesindusel oleks võimalik ruume ja nende kasutamist hallata läbi kasutajaliidese, oli vaja rakendusliidesesse üles seada täiendavad URI-d. Selleks, et pärida ressursse, järgib REST API teekondade nimetamisel URI mustrit (ing. k. *unified resource identifiers*).

Iga infokild, millele saab nime anda, on REST API mõistes ressurss. Ressurss võib olla üksik objekt või kollektsioon objektidest [15]. Iga URI peab peegeldama seda, millise ressursiga täpsemalt tegemist on, näiteks kas tegemist on ühe objektiga või mitmega [15]. Rahastusplatvormi rakendusliidese kontekstis on ressurssideks ruum ja ruumi kasutus. Tabelis 8 on välja toodud rakendusliidesesse lisatavad teekonnad, kasutatav HTTP meetod ja lühikirjeldus tegevusest.

Tabel 8. Ruumi haldusega seotud API teekonnad.

URI	**Meetod**	**Kirjeldus**
/rooms	GET	Tagastab kõik üliõpilasesinduse poolt hallatavad ruumid.
/rooms/{roomId}	GET	Tagastab üliõpilasesinduse poolt hallatava ruumi detailse info ruumi primaarvõtme alusel.
/rooms	POST	Võimaldab lisada üliõpilasesindusel süsteemi uue ruumi.
/rooms/{roomId}	PUT	Võimaldab üliõpilasesindusel muuta nende olemasoleva ruumi infot.
/rooms/{roomId}/usages	GET	Tagastab ühe üliõpilasesinduse poolt hallatava ruumiga seonduva kasutusajaloo.
/rooms/{roomId}/usages	POST	Võimaldab lisada üliõpilasesindusel süsteemi ruumi kasutamise kohta uue kirje.
/rooms/{roomId}/usages/{usageId}	PUT	Võimaldab üliõpilasesindusel muuta ruumi kasutusajaloo kirjet.

Kõik toimingud, mis hõlmavad ruumiga seotud info muutmist või lisamist, eeldavad, et seda teeb sisse logitud üliõpilasesinduse kasutaja. Need tegevused, mis eeldavad autoriseeritud üliõpilasesinduse kasutajat, on koodi tasemel märgitud @Allowed annotatsiooniga ning parameetrina on kaasa antud MANAGER. Kõik ruumi haldusega seotud päringud lähevad läbi esituskihis oleva *controller* klassi – RoomController (vt joonis 12). *Controller* klassi eesmärgiks on tegeleda sisse tulevate päringutega ning saata neid edasi äriloogika kihti [16].

```
@RestController
@RequestMapping("/rooms")
public class RoomController {

    @Resource
    private RoomManagement roomManagement;
    @Resource
    private RoomUsageManagement roomUsageManagement;

    @GetMapping()
    public ResponseEntity<?> getRooms() {...}

    @GetMapping("/{roomId}")
    public ResponseEntity<?> getRoomById(@PathVariable Long roomId)
{...}

    @Allowed(role = MANAGER)
    @PostMapping()
    public ResponseEntity<?> createRoom(@RequestBody RoomDetails
roomDetails) {...}

    @Allowed(role = MANAGER)
    @PutMapping("/{roomId}")
    public ResponseEntity<?> updateRoom(@PathVariable Long roomId,
@RequestBody RoomDetails roomDetails) {...}

    @GetMapping("/{roomId}/usages")
    public ResponseEntity<?> getRoomUsageHistory(@PathVariable(name =
"roomId") Long roomId) {...}

    @Allowed(role = MANAGER)
    @PostMapping("/{roomId}/usages")
    public ResponseEntity<?> createRoomUsageRecord(@PathVariable Long
roomId, @RequestBody RoomUsageRequestBody roomUsageHistory) {...}

    @Allowed(role = MANAGER)
    @PutMapping("/{roomId}/usages/{usageId}")
    public void updateRoomUsageRecord(@PathVariable Long roomId,
@PathVariable Long usageId, @RequestBody RoomUsageRequestBody
roomUsageHistory) {...}
}
```

Joonis 12. RoomController klassi programmikoodi näide.

RoomController klassist suunatakse päring edasi äriloogika kihti. Äriloogika kihti on lisatud ruumi haldusega ja ruumi kasutusajalooga tegelevad kasutusjuhtumeid peegeldavad klassid, mis on nimetatud vastavalt RoomManagement ja

RoomUsageManagement. RoomManagement ja RoomUsageManagement klassides on kirjeldatud ühe ja mitme kirje pärimine andmepääsukihist, kirje loomine ja muutmine.

4 Tulemused ja analüüs

Käesoleva lõputöö eesmärgid said sõnastatud peatükis 1.3. Töö peamiseks eesmärgiks oli teha vajalikud täiendused üliõpilasesinduse rahastusplatvormi kasutaja- ja rakendusliidesesse ning andmebaasi, et ÜEl oleks võimalik läbi süsteemi:

- luua eraldi konkurss, mille kaudu saavad tudengiorganisatsioonid endale taotleda ruumide kasutusõigust üheks kalendriaastaks;
- eristada ruumidega seonduvaid taotlusi ja anda nendele tagasiside;
- näha hallatavaid ruume ja muuta ruumidega seonduvat infot;
- näha ja muuta ruumide kasutamise ajalugu.

Lõputöö raames sõnastatud eesmärkides sai täidetud järgnev – ruumi taotlemise konkursiga seotud toiminguid on võimalik läbi rahastusplatvormi kasutajaliidese teha. Samuti on võimalik näha, millised ruumid on üliõpilasesinduse poolt hallatavad ja nende kasutamise ajalugu. Vajadusel saab ruumidega seonduvat infot ka muuta. Kahjuks pole veel võimalik rahastusplatvormi kasutaja- ja rakendusliidese kaudu kustutada ruume ning nende kasutamisega seotud infot.

Rakendusliidesele ruumi konkursi loomise funktsionaalsuse lisamine kujunes oodatust keerulisemaks. Seda seetõttu, et autor ei olnud varasemalt kursis, mis sammud on seotud projektide konkursiga ja millised sammud aastatoetuste konkursiga. Kasutajaliidesele muudatuste sisse viimine oli väljakutseks oli samuti seepärast, et autor ei olnud kursis, mis on mingisuguse tegevuse eesmärgi saavutamiseks vajalikud sammud. Nagu näiteks mis on võimalikud erijuhud, kus on samuti vaja muudatusi koodibaasi sisse viia. Küll aga mõlemas rakenduses oli võimalik taaskasutada olemasolevaid komponente ja loogikat.

Lõputöö raames arendatud funktsionaalsused nõuavad kindlasti täiendavat testimist enne kui lahendus jõuab lõppkasutajateni. Autor jõudis lõputöö kirjutamise ajahetkel demonstreerida rahastusplatvormi kasutajale loodud täiendusi ning selle käigus tulid kasutajaliideses esimesed kitsaskohad välja. Nagu näiteks tekkisid küsimused, mida kuvatakse lõpp-kasutajale siis, kui ei ole andmeid välja näidata ja kuidas lõpp-kasutaja teab, et ta tegevus on edukalt lõpetatud.

Peale täiendava testimise ning kitsaskohtade parendamise, on vaja autoril ka teha andmete migratsioon. Lõputöö raames tehti muudatused andmebaasi struktuurile ja selle tulemusena on vaja eksisteerivad andmed ümber paigutada loodud tabelitesse. Rahastustplatvormi uuendatud olemi-suhte diagramm on leitav Lisa 3 alt.

Autoril on kavas rakendusliidest rohkem katta ühiktestidega, et edasistel arendajatel oleks selgem pilt ees, kuidas nende muudatused mõjutavad rakenduse edasiarendust. Samuti loodab ta lisada rakendusliidesele integratsioonitestid ning kasutajaliidesele endale ühiktestid. Lisaks on vaja rakendustes kasutatavad tehnoloogiad kaasajastada enne, kui nendele versioonidele kaob tugi.

Lõputöö kirjutamise ajahetkel autor täheldas, et kasutajaliidese rakenduses on kasutusel ainult eesti keel. Tallinna Tehnikaülikooli linnakus on tegutsemas välistudengitele pühendatud organisatsioonid, nagu näiteks AIESEC ja Erasmus, kes on ka taotlenud varasemalt ÜE-lt enda tegevuse jaoks toetust. Sellest tulenevalt usub autor, et rahastusplatvormil võiks olla rakenduses ka ingliskeelsed tõlked.

Kokkuvõte

Käesoleva lõputöö raames võttis autor üle Krõõt Grete Mändi ja Raimond Lume lõputööde raames loodud Tallinna Tehnikaülikooli üliõpilasesindusele loodud rahastusplatvormi koodibaasi, et teha sellele lisaarendusi. Lisaarenduste eesmärgiks oli luua süsteemi funktsionaalsused, mille abil saab ÜE luua ruumide kasutusõiguste taotlemiseks eraldi konkursi ning hallata olemasolevaid ruume ja nende kasutamisega seotud infot.

Autor uuris töö analüüsi osas erinevaid nõuete kogumise tehnikaid. Lõputöö eesmärgist tulenevalt leidis autor, et nõuete väljaselgitamiseks sobivaimad tehnikad on üliõpilasesinduse intervjueerimine, rahastusplatvormi dokumentatsiooni analüüs, varasema koodibaasiga tutvumine ning prototüüpimine. Intervjuu ja dokumentatsiooni tulemusena sõnastas autor esmased nõuded. Nõuete valideerimiseks kasutas autor prototüüpimist. Selleks tegi autor sõrestikmudelid, kus oli välja toodud, milliseid andmeid kasutajatelt oodatakse kasutajaliidese vaadetes.

Töö praktilises osas autor realiseeris teostatud analüüsi põhjal rahastusplatvormile lisafunktsionaalsused. Autor pidi andmebaasile looma tabeleid juurde, kuhu saaks salvestada konkursitüübist tulenevalt avalduste sisu ja ruumide haldusega seonduvat infot. Lisaks pidi autor viima sisse muudatused rahastusplatvormi rakendusliideses, et oleks võimalik luua ruumide taotlemiseks konkurss ja tegeleda ruumide haldusega. Selleks, et ÜE ja tudengiorganisatsioonid saaksid mugavalt rahastusplatvormis toiminguid teha, tegi autor sõrestikmudelite alusel uued vaated kasutajaliidesesse.

Töö koostamisel sai autor teadmisi selle kohta, kuidas nõudeid koguda ja kuidas on neid võimalik valideerida. Samuti õppis autor selle kohta, milliseid elemente tasub kasutajaliideses kasutada teatud toimingute jaoks. Lisaks sai autor uusi teadmisi puhta arhitektuuri põhimõtete ja URI nimetamise hea tava kohta.

Töö tulemusena valmisid täiendused rahastusplatvormile, mille abil on võimalik tudengiorganisatsioonidel taotleda oma tegevuse jaoks ÜE-lt kasutusõigust ja saavad näha ÜE poolt hallataid ruume. Selleks, et lisaarendused jõuaksid lõppkasutajateni, on vaja teha andmete migratsioon ning võimalike kitsaskohtade väljaselgitamiseks täiendav testimine.

Kasutatud kirjandus

[1] K. G. Mänd, *TalTech üliõpilasesinduse veebipõhise rahastusplatvormi rakendusliidese arendus järgides kasutuslookeskse arhitektuuri põhimõtteid,* Tallinn, 2019.

[2] R. Lume, *TalTech üliõpilasesinduse rahastusplatvormi arendus jätkusuutliku tarkvaraarenduse põhimõtteid järgides,* Tallinn, 2020.

[3] Z. Zhang, „Effective Requirements Development - A Comparison of Requirements Elicitation techniques," %1 *Software Quality Management Conference (SQM 2007)*, Tampere, 2007.

[4] S. Mahnot, „The Community Blog for Business Analysts," 16 March 2017. [Võrgumaterjal]. Available: https://www.modernanalyst.com/Community/CommunityBlog/tabid/182/ID/3741/Defining-Requirements.aspx. [Kasutatud 22.10.2022].

[5] G. Koelsch, Requirements Writing for System Engineering, Herndon: Apress, 2016.

[6] ReQtest, „Business Requirements Document (BRD) - Understanding the basics," 5 September 2018. [Võrgumaterjal]. Available: https://reqtest.com/requirements-blog/business-requirements-document-brd/. [Kasutatud 23.10.2022].

[7] IEEE, 830-1998 - IEEE Recommended Practice for Software Requirements Specifications, IEEE, 1998.

[8] Sketch B.V., „Wireframe vs mockup vs prototype: What's the difference?," 8 April 2022. [Võrgumaterjal]. Available: https://www.sketch.com/blog/2022/04/08/wireframe-vs-mockup-vs-prototype/. [Kasutatud 23.10.2022].

[9] M. Myre, „Figma vs Adobe XD: An In-Depth Comparison for UX/UI Designers," Designlab, [Võrgumaterjal]. Available: https://designlab.com/blog/figma-vs-adobe-xd/. [Kasutatud 27.10.2022].

[10] E. N. McKay, UI is communication: How to design intuitive, user-centered interfaces by focusing on effective communication, Newnes, 2013.

[11] P. Raspel, „Andmemudeli normaliseerimata kuju," [Võrgumaterjal]. Available: https://enos.itcollege.ee/~priit/1.%20Andmebaasid/1.%20Loengumaterjalid/07/7-1.htm. [Kasutatud 27.11.2022].

[12] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Pearson, 2017.

[13] Refactoring.Guru., „Factory Method in Java," [Võrgumaterjal]. Available: https://refactoring.guru/design-patterns/factory-method/java/example. [Kasutatud 16.11.2022].

[14] R. Stafford, „Service Layer," /thoughtworks, [Võrgumaterjal]. Available: https://martinfowler.com/eaaCatalog/serviceLayer.html. [Kasutatud 27.11.2022].

[15] L. Gupta, „REST Resource Naming Guide,“ [Võrgumaterjal]. Available: https://restfulapi.net/resource-naming/. [Kasutatud 13.11.2022].

[16] P. Dutta, „Quick Guide to Spring Controllers,“ Baeldung, [Võrgumaterjal]. Available: https://www.baeldung.com/spring-controllers. [Kasutatud 28.11.2022].

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

Mina, Aneta Claudia Marttila

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „TalTech üliõpilasesinduse rahastusplatvormile ruumide haldamise ja pikaajalise kasutusõiguse jagamise funktsionaalsuse arendus“, mille juhendaja on Brita Moorus
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

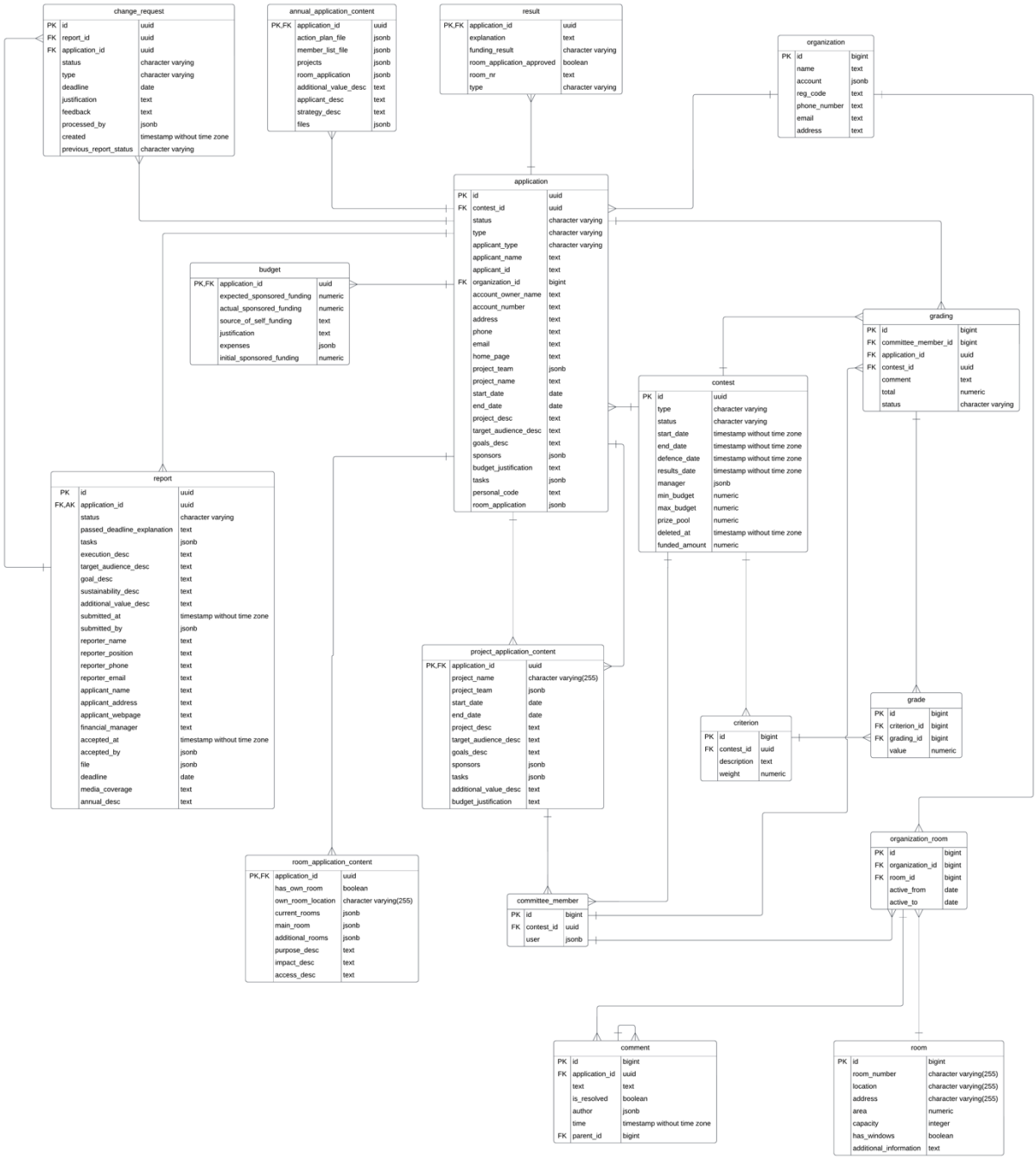
03.01.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – avaldustega seotud tabeli algne struktuur

Joonis 13. Avaldustega seotud tabeli algne struktuur.

Lisa 3 – Rahastusplatvormi uuendatud olemi-suhte diagramm



Joonis 14. Rahastusplatvormi uuendatud olemi-suhte diagramm.