TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Krõõt Grete Mänd 164024IAPB

TALTECH ÜLIÕPILASESINDUSE VEEBIPÕHISE RAHASTUSPLATVORMI RAKENDUSLIIDESE ARENDUS JÄRGIDES KASUTUSLOOKESKSE ARHITEKTUURI PÕHIMÕTTEID

bakalaureusetöö

Juhendaja: Gert Kanter

MSc

Tallinn 2019

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: Krõõt Grete Mänd

20.05.2019

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua veebirakenduse rakendusliides Tallinna Tehnikaülikooli Üliõpilasesinduse rahastusplatvormi jaoks. Loodud rakenduse eesmärgiks on hallata tudengielu rahastamise protsessi ning lihtsustada selle toimumist üliõpilasesinduse, tudengiorganisatsioonide ja tudengite jaoks. Valminud rakendus peab vastama puhta ja kasutuslookeskse arhitektuuri põhimõtetele.

Töö esimeses osas antakse ülevaade hea rakendusliidese arendamise ning puhta ja kasutuslookeskse arhitektuuri põhimõtetest. Samuti antakse ülevaade tudengielu rahastamise korrast Tallinna Tehnikaülikoolis.

Töö teises osas antakse esmalt ülevaade kasutatud tehnoloogiatest. Seejärel kirjeldatakse valminud rakenduse arhitektuuri ning selgitatakse, miks selline arhitektuur loodi. Lõpuks antakse ülevaade valminud rakenduse funktsionaalsusest.

Töö tulemuseks on reaalselt töötav rakendusliides, mis täidab tudengielu rahastamise protsessi jaoks vajaliku põhifunktsionaalsust ning, mis vastab kasutuslookeskse puhta arhitektuuri põhimõtetele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 8 peatükki ja 13 joonist.

Abstract

Development of TalTech Student Union's Web Based Finance Platform API Following the Principals of Use Case Oriented Architecture

The purpose of this thesis is to create an API for the web based finance platform of the Student Union of Tallinn University of Technology. The application is supposed to support the process of funding studentlife in TalTech and make it easier for the Student Union, student organisations and the students themselves. The architecture of the developed application must abide by the principles of use case oriented clean API design.

The first part of this thesis gives an overview of clean API development and the principles of use case oriented design as well as the process of funding studentlife in Tallinn University of Technology.

The second part gives an overview of technologies used in the development of the API. It then goes on to explain the architecture and functionality of the application.

The result of this thesis is a working API that fulfils the main functionality needed in the process of funding studentlife and has a clean use case oriented structure.

The thesis is in Estonian and contains 35 pages of text, 8 chapters and 13 figures.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* Rakendusliides
ÜE	Tallinna Tehnikaülikooli Üliõpilasesindus
DAO	*Data Access Object* Andmepääsu objekt
MVC	*Model-View-Controller* Mudel-Vaade-Kontroller
JDBC	*Java Database Connectivity*
SOAP	*Simple Object Access Protocol*
JSON	*JavaScript Object Notation*
REST	*Representational State Transfer*
CRUD	*Create Read Update Delete* Loo Loe Uuenda Kustuta

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tallinna Tehnikaülikoolis käib tudengielu rahastamine läbi üliõpilasesinduse. Organisatsioonidel ja tudengitel on võimalik erinevate konkursside raames taotleda raha oma tegevuste ja projektide läbi viimiseks. Siiani on avalduste ja hiljem nende aruannete esitamine olnud tülikas protsess nii tudenngite kui üliõpilasesinduse jaoks. Täita tuleb suuri ja keerukaid exceli tabeleid, mida on raske kontrollida ja üle vaadata ning kogu suhtlemine toimub e-kirjade teel. Ka varasemalt on alustatud arendusprotsessi kogu süsteemi veebiplatvormile viimiseks, kuid antud projekt on siiani poolik.

2018. aasta sügise seisuga oli arendatud osaliselt valmis rakendus, mida oli vaja veel täiendada ja edasi arendada. Kuid ehitatud süsteem oli raskesti mõistetav, kood mitteloetav ja selle edasiarendamine selleks, et veebiplatvorm reaalselt valmis saaks ja kasutatav oleks, oli raske. Käesoleva töö eesmärgiks sai olukorra parandamine ning puhtalt lehelt alustamine.

Käesoleva bakalaureuse töö eesmärgiks on luua üliõpilasesinduse veebipõhise rahastusplatvormi jaoks uus rakendusliides, mis täidaks nõutud funktsionaalsusi ning oleks tänu heale arhitektuurile hästi loetav ja arusaadav.

Eesmärgini jõudmiseks luuakse esmalt rakendusele arhitektuur, mis järgib puhta ja kasutuslookeskse arhitektuuri põhimõtteid ning seejärel realiseeritakse rakenduse põhifunktsionaalsus. Rakenduse realiseerimiseks kasutatakse erinevaid tehnoloogiaid, mille hulka kuuluvad Gradle, Java, Spring, PostgreSQL.

Töö esimene osa annab ülevaate API arenduse ja arhitektuuri põhimõtetest, millele rakenduse arhitektuuri välja töötades tugineti. Samuti antakse ülevaade tudengielu rahastamise süsteemist Tallinna Tehnikaülikoolis, mis on rakenduse funktsionaalsuse sisuks.

Teises osas kirjeldatakse valminud rakenduse arhitektuuri ja funktsionaalsust ning tuuakse välja selle edasi arendamse võimalused.

2 Hea rakendusliidese arendamine

2.1 Mis on rakendusliides?

Rakendusliides ehk API (Application Programming Interface) on suhtluslüli kahe tarkvara vahel – see võimaldab kahel osapoolel omavahel andmeid ja käske vahetada. Nendeks osapoolteks võivad olla kaks masinat, arvutit, rakendust või ka nagu käesolevas töös kliendipoolne kasutajaliides ja andmebaas [1].

Enamik inimesi kasutab päeva jooksul mitmeid API-sid. Iga kord, kui me vaatame telefonist oma e-kirju, kontrollime ilmateadet või saadame Facebookis sõnumeid, kasutame me mõnda API-t. Nende rakenduste kasutamisel on rakendus meie telefonis ühendatud internetiga ning saadab andmeid serverisse. Seal võetakse info vastu, tõlgendatakse ja töödeldakse seda ning saadetakse uued andmed tagasi meie telefoni, kus neid andmeid kuvatakse läbi kasutajaliidese meile arusaadaval kujul [1].

API peamiseks eesmärgiks siin on turvalisuse tagamine. Tänu API kasutamisele ei ole meie telefonis leiduv informatsioon kunagi täielikult serverile nähtav ja vastupidi. Selle asemel suheldakse väikeste informatsiooni hulkadega ning edastatakse alati ainult vajalikud andmed [1].

Suurem osa rakendusliideseid, millega me kokku puutume on veebipõhised API-d, tuntud ka kui veebiteenused, mis ühendavad omavahel osapooli, kes suhtlevad üle interneti. Alates 2005. aastast on veebipõhiste API-de kasutamine märkimisväärselt kasvanud ning välja on arenenud mitmeid erinevaid protokolle ja standardeid, millele need ülesehitatud on. Algselt oli üks populaarsemaid standardeid SOAP, mis kasutab andmete edastamiseks XML-vormingut, kuid peale JSON-i esiletõusmist toetutakse aina enam HTTP protokollile. Praeguseks hetkeks ongi kõige populaarsemaks tarkvara arhitektuuri mustriks veebirakenduste arendamisel saanud REST [1].

2.2 REST printsiibid

REST on tarkvaraarhitektuuri muster, mis seab API-i loomisele kindlad raamid ja piirangud. See võib toetuda erinevatele protokollidel, kuid tüüpiliselt seostatakse seda just HTTP-ga. Esmakordselt kirjeldas REST põhimõtteid Dr. Roy Fielding 2000. aastal oma doktoritöös, kus ta tõi välja 5 peamist printsiipi, mida nö RESTful veebirakendused järgima peaks [2].

2.2.1 Klient-Server

Kliendi ja serveri printsiip väidab, et klient ja server (näiteks kasutajaliides ja andmebaas vastavalt) peaksid olema üksteisest eraldatud ning võimelised eraldiseisvalt arenema. Ehk siis näiteks mobiilirakenduse puhul, peame me saame teha muudatusi rakenduse kasutajaliideses ilma, et need mõjutaksid kuidagi andmebaasi objekte või disaini serveri tasemel ja vastupidi [1, 2].

2.2.2 Olekuta

REST API-il ei ole olekut. See tähendab, et kõik päringud on eraldiseisvad ning meie API ei talleta informatsiooni selle kohta, kes ja milliseid päringuid on varasemalt tehtud. Kõik päringud peavad sisaldama piisavalt informatsiooni, et vajalikud tegevused edukalt sooritada [1, 2].

2.2.3 Vahemälu

Olekuta API võib oluliselt suurendada päringute hulka, mis sisaldavad suurt hulka sissetulevaid ja väljaminevaid andmeid. Selleks, et vältida liigset koormust peaks REST API olema üles ehitatud nii, et see soodustaks informatsiooni talletamist vahemälus. Ehk saadetav vastus võiks sisaldada informatsiooni selle kohta, kui kaua tagastatav informatsioon on kehtiv või, kui tegu on andmetega, mida ei peaks vahemällu salvestama, siis hoopis seda informatsiooni. Selle printsiibi järgimine vähendab oluliselt päringute arvu ja seeläbi koormust serverile ning kindlustab ühtlasi kiire ja efektiivse rakenduse kasutamise [1, 2].

Kuigi vahemälu kasutamine on oluline RESTful veebirakenduse puhul, toimub see siiski kliendi poolel. Ehk siis selle eest vastutab, meie rakendusliidest kasutav rakendus või kasutajaliides, mitte API ise. API ülesanne on anda juhiseid selles osas, kuidas

informatsiooni talletama peab – antud informatsiooni saab näiteks edastada HTTP vastuse päises [1, 2].

2.2.4 Universaalne liides

Eelnevalt seletatud klient-server põhimõtte kohaselt peavad klient ja server olema üksteisest selgelt eraldatud nii, et näiteks kasutajaliidese loogika ei sõltu liialt API-ist ega andmebaasist. Universaalne liides viitab sellel, et klient ja server saavad omavahel suhelda mõlema jaoks nö arusaadavas keeles, sõltumata üksteise arhitektuurist või näiteks sellest, mis programmeerimiskeelt on kummagi realiseerimiseks kasutatud. Seega peab RESTful API võimaldama suhtlust standardiseeritud ja muutumatul kujul toetudes näiteks HTTP protokollile ja edastades andmeid JSONi kujul või andmebaasiga suheldes toetudes CRUD (Create, Read, Update, Delete) operatsioonidele [1, 2].

2.2.5 Kihtide süsteem

Viimane viiest printsiibist näeb ette, et API peaks koosnema erinevatest kihtides, millel on igaühel oma kindel funktsionaalsus ja ülesanne. Näitena saab vaadata MVC mustrit, kus on need erinevad ülesanded väga selgelt näha:

Mudeli kiht vastutab andmete, loogika ja rakenduse reeglite eest ning töötleb andmeid mis saadakse kontrolleri käsklustest ning kuvatakse vaate abil. Kontroller tegeleb sissetulevate päringutega ning muudab need käsklusteks, mis edastada mudelile või vaatele. Vaate kiht tegeleb välja saadetavate vastustega ehk see on kokkuvõttes vastutav info esitluse eest. Kõik kihid on eraldiseisvad, kuid suhtlevad üksteisega. RESTi puhul kehtib sama idee. Erinevad arhitektuuri kihid loovad koostöös hierarhia, mis aitab ehitada rakenduse, millel on mitmeid eeliseid [1, 2].

Esiteks lihtsustub vanade rakenduste edasi arendamine. Uued ja vanad osad saab tänu kihtidele üksteisest eraldada ning me ei ole uue funktsionaalsuse arendamisel sunnitud jääma vanema ja aegunud tarkvara raamidesse vaid saame kasutada uuemaid võimalusi. Samuti on lihtsam erinevaid rakenduse osasid tehnoloogia edasiarenemisel uuendada. Tänu sellele pikendame oluliselt rakenduse eluiga ja vähendame pärandkoodi teket. See kõik aga nõuab, et erinevad kihid oleksid üksteisest nii sõltumatud kui vähegi võimalik [1, 2].

Teiseks parandab selline struktuur rakenduse turvalisust. Rünnakuid on võimalik peatada kõigis kihtides ning kokkuvõttes on tõenäosus, et need jõuavad serverini, tunduvalt väiksem. Kuna erinevate turvakontrollide puhul on alati võimalus, et neist pääsetakse mööda, siis erinevad kihid annavad meile võimaluse implementeerida rohkem eraldiseisvaid kontrolle igas kihis. Kasutades kihilist arhitektuuri saame peita serveri olulised aspektid nii, et kliendi pool ei pääse neile kunagi otse ligi [1, 2].

2.3 Puhas arhitektuur

2.3.1 Puhas kood

Raamatus "Clean Archtecture" tuuakse autori poolt välja analoogia, kus tarkvara rakendust võrreldakse ehitise ja telliskividega. Ühest küljest, kui tellised ei ole kvaliteetsed, siis neist on raske ehitada korraliku hoonet, teiselt poolt aga võib meil olla väga hea alusmaterjal, kuid seda valesti kasutades saame me ilusa ehitise asemel siiski lihtsalt kivihunniku. Mõte seisneb selles, et puhas arhitektuur algab eelkõige puhtast koodist ning rakendab puhta koodi põhimõtteid ka laiemas mastaabis mitte ainult mõne klassi või meetodi raames [3].

Puhtast koodist rääkides tulevadki mängu S.O.L.I.D printsiibid. Need on põhimõtted, mis ütlevad meile, kuidas jagada oma meetodid ja andmestruktuurid klassideks ning kuidas neid klasse omavahel siduda. S.O.L.I.D printsiipide eesmärk on luua tarkvara komponendid, mida saab väikese vaevaga muuta ja edasi arendada ning mis on loetavad ja kergesti arusaadavad [3].

Robert C. Martin kirjeldab S.O.L.I.D põhimõtteid järgmiselt [3]:

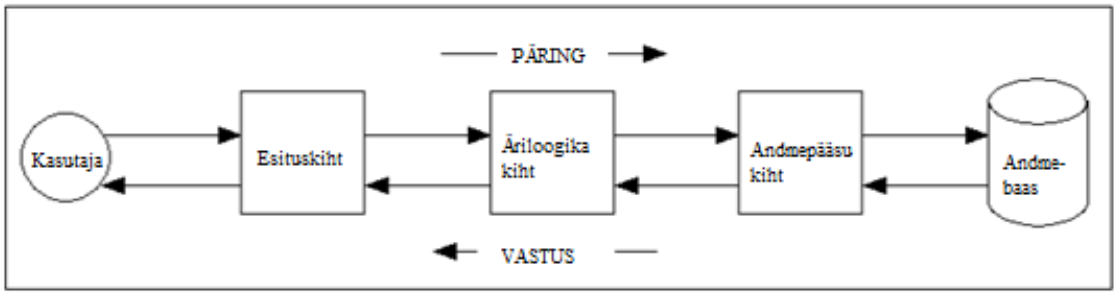
- Ainsa vastutuse printsiip – igal klassil peaks olema ainult üks ülesanne. Kui klass vastutab rohkem, kui ühe funktsionaalsuse eest, siis võivad ühes klassis muudatuste tegemisel tekkida konfliktid teiste klasside ja nende ülesannetega.
- Avatud-suletud printsiip – moodul peaks olema avatud laiendamiseks kuid suletud muutmiseks. Juhul kui tekivad uued vajadused, saab moodulit laiendada, kuid teda ennast ei pea nende realiseerimiseks muutma. Printsiibi eesmärgiks on peamiselt see, et uue funktsionaalsuse tekkimisel peaksime me saame selle implementeerida, kirjutades uut koodi mitte muutes juba eksisteerivat koodi.

- Liskov'i asendus printsiip – rakenduse siseselt peab saama objekte välja vahetada nende alamtüüpidega nii, et rakendus jääks ilma lisamuutusteta korrektselt töötama.
- Liideste eraldatuse printsiip – komponendid ei peaks olema sunnitud sõltuma millestki, mida nad ei kasuta. Ehk siis näiteks kõik klassid, mis implementeerivad mingit liidest peavad reaalselt vajama kõiki liideses kirjeldatud meetodeid. Kui tekib olukord, kus neil mõnda meetodit vaja ei ole, aga nad on sunnitud seda implementeerima, siis see on märk sellest, et kasutatav liides tuleks jagada väiksemateks osadeks.
- Sõltuvuse inversiooni printsiip – moodul ei tohiks sõltuda endast madalama taseme moodulite (sõltuvuste) implementatsioonist vaid nende abstraktsioonidest.

2.3.2 Kolmekihiline API arhitektuur

Peatükis 2.2.5 sai üldiselt tutvustatud kihtide süsteemi printsiipi – vaatame seda nüüd lähemalt. Üks levinumaid API struktuure, mida rakendatakse, eriti Springi rakenduste puhul, on nö kolmekihiline struktuur. Nagu nimi ka ütleb jagatakse API antud põhimõttes kolmeks kihiks. Kõik kihid mängivad oma rolli iga kord, kui kasutaja teeb API pihta päringu selleks, et andmeid küsida või muuta [4].

Järgnevalt seletame lahti need kolm kihti ning kuidas need oma vahel suhtlevad ja koostööd teevad (vt Joonis 1):



Joonis 1. Päringud ja vastused kolmekihilises arhitektuuris [4]

- Esituskiht, enamasti tuntud kui controller, on rakenduse kõige kõrgem kiht. Selle peamiseks ülesandeks on olla nö tõlgiks. Controller võtab vastu kasutaja päringu ja annab selle edasi äriloogika kihile, kust ta saab vastu päringule vastavad andmed, mille ta see järel kasutajale sobival kujul edastab [4].

- Äriloogika kiht, enamasti tuntud kui service, vastutab rakenduse funktsionaalsuse eest. Selles kihis tehakse otsuseid, analüüsitakse ja töödeldakse andmeid ning liigutatakse andmeid esituskihi ja andmepääsu kihi vahel. [4]

- Andmepääsu kiht, enamasti tuntud kui repository, vastutab andmebaasiga või failisüsteemiga suhtlemise eest. Tema ülesandeks on pärida baasist vajalikud andmed ning edastada need äriloogika kihile töötlemiseks, kust need hiljem kasutajani jõuavad. [4]

2.3.3 Kasutuslookeskse arhitektuuri põhimõte

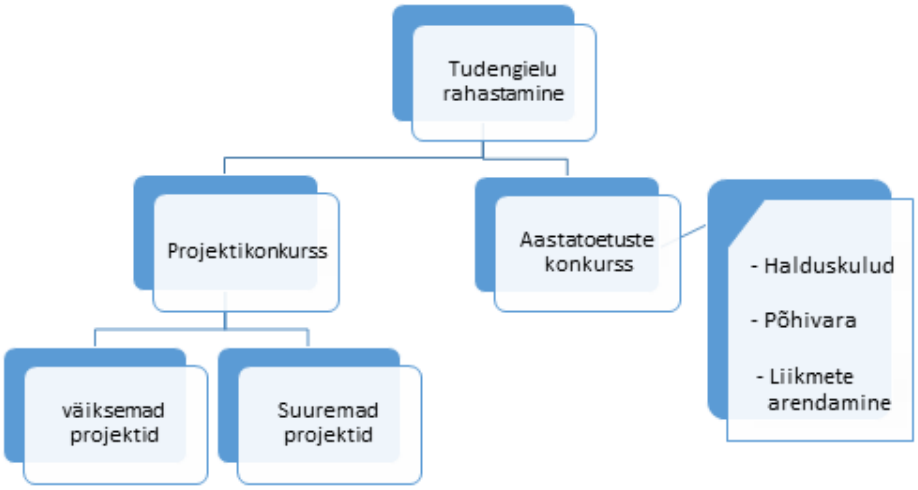
Lisaks üleüldiste puhta arhitektuuri põhimõtete jälgimisele on võimalik oma API arhitektuuri paremaks ja arusaadavamaks teha järgides kasutuslookeskse arhitektuuri põhimõtet.

Kasutuslookeskae arhitektuuri kohaselt on arhitektuuri peamine prioriteet toetada rakenduse funktsionaalsust. Probleem seisneb selles, et rakenduse struktuur ei mõjuta otseselt selle käitumist. Kasutuslookeskse arhitetuuri eesmärk ei ole funktsionaalsuse mõjutamine või juhtimine vaid pigem selle esile tõstmine [3].

Hea arhitektuuri kõige olulisem ülesanne on rakenduse käitumise toetamine ja, mis veel tähtsam, selgitamine. Rakenduse funktsionaalsus, peab olema juba struktuurile peale vaadates selge ja arusaadav. See tähendab, et erinevad kasutuslood on realiseeritud klassidena, mis on kohe nähtavad ning mille nimi väljendab selgelt, mis funktsionaalsus klassis implementeeritakse. Näiteks kui meie rakendus on loodud projektikonkursside haldamiseks ja üks oluline kasutuslugu on uue konkursi loomine, siis selle kasutusloo loogika on realiseeritud klassis nimega LisaKonkurss. Sellise arhitektuuri puhul ei pea me liikuma erinevate failide vahel ja tutvuma nende sisuga selleks, et vastata küsimusele “Mida rakendus teeb?” vaid see on selge juba failide nimesid vaadates [3].

3 Tudengielu rahastamine Tallinna Tehnikaülikoolis

TalTech Üliõpilasesindus pakub tudengitel ja tudengiorganisatsioonidele võimalust taotleda toetusrahasid projektide korraldamiseks ning organisatsioonide töö lihtsustamiseks. Selleks korraldatakse igaaastaselt konkursse, kuhu on võimalik rahastuse taotlemiseks esitada projektijuhi või organisatsiooni esindaja poolt digiallkirjastatud avaldus. Käesolevas peatükis seletatakse lahti tudengielus rahastamise struktuur (vt Joonis 2) ja protsess, mille haldamine on meie rakenduse põhieesmärk.



Joonis 2: Tudengielu rahastamise struktuur [5]

3.1 Konkursid

3.1.1 Aastatoetuste konkurss

Aastatoetuste konkurss toimub 1 kord aastas ning sinna saavad avalduse esitada TalTechi üliõpilasorganisatsioonid. Konkurss on mõeldud organisatsioonide tegevust toetavate varade ja liikmete arendamiseks mõeldud kulude katteks. Aastatoetuste konkursi taotluse täitmine toimub malli alusel (vt Lisa 1) [6].

3.1.2 Projektikonkurss

Projeketikonkursi eesmärgiks on toetada TalTechi üliõpilaste, TalTechi üliõpilasorganisatsioonide ja üliõpilaskogude projekte. Projektikonkurss jaguneb veel omakorda suurte ja väikeste projektide konkursiks. Aastatoetuste konkursi taotluse täitmine toimub malli alusel (vt Lisa 2) [6].

3.1.2.1 Suurte projektide konkurss

Suurte projektide konkursi eesmärk on edendada TalTechi tudengielu ning soodustada traditsioonide tekkimist ja elushoidmist. Konkurss leiab aset 1 kord aastas ning sinna saab esitada projektide taotlusi, mille maksumus ületab TalTechi Üliõpilaskogu juhatuse kinnitatud piiri [6].

3.1.2.2 Väikeste projektide konkurss

Väikeste projektide konkursi eesmärk on toetada tudengite omaalgatust ning edendada ja suunata tudengielu. Konkurss leiab aset 4 korda aastas ning sinna on võimalik esitada projektitaotlusi mille maksumus ei ületa TalTechi Üliõpilaskogu juhatuse kinnitatud piiri [6].

3.2 Hindamine

Taotlusi hinnatakse mitmes etapis. Peale taotluse esitamist kontrollib konkursijuht taotluse vormikohasust ja taotleja abikõlblikkust. Kui esitatud taotluses esineb puuduseid informeeritakse taotlejate ning tal on võimalus need kõrvaldada ja taotlus uuesti esitata. Konkursijuht saab nõuda tehniliste vigade parandamist 3 korda ning kui taotleja ei suuda nende jooksul vigu likvideerida on konkursijuhil õigus avaldus tagasi lükata [7].

Juhul kui esitatud taotlus on vormikohane liigub see edasi lõplikuks hindamiseks konkursi komisjonile, kes annab taotlusele hinnangud kokkulepitud hindamiskriteeriumite alusel. Komisjon võib teha põhjendatud ettepanekuid taotluse täiendamiseks, mis esitatakse taotlejale koos tähtajaga tagasiside andmiseks ja taotluse täiendamiseks [7].

Hindamisprotsess lõppeb komisjoni koosolekuga, mille esimeses osas on kõigil taotluse esitajatel võimalus oma projekti kaitsta. Koosoleku teises osas toimub kinnine arutelu

toetuste suuruste üle. Peale tulemuste selgumist koostatakse tulemustest raport, mis edastatakse kõigile konkursil osalejatele ning kõik komisjoni liikmed peavad konkursi tulemused kinnitama digiallkirjaga [7].

3.3 Aruandlus

Kõik projektide ja aastatoetuste konkursil toetust saanud taotluse esitajad peavad esitama toetusrahade kättesaamiseks digiallkirjastatud aruande. Aruande juurde kuuluvad aastatoetusega või projektiga seonduvad tulude ja kulude kokkuvõte ning kuludokumendid koos maksekorraldustega. Lisaks finantskokkuvõttele (Lisa 5) peab olema esitatud ka tegevusaruandlus (Aastatoetuse konkursi aruande mall Lisa 3, projektikonkursi aruande mall Lisa 4). Aruandluse kontrollimise eest on vastutav ÜE juhatuse poolt määratud isik ning aruannetega on võimalik tutvuda ka ÜE juhatusel ja revisionikomisjonil [6, 8].

4 Kasutatud tehnoloogiad

4.1 Gradle

Gradle on tarkvara ehitust automatiseeriv tööriist, mis põhineb Groovil ja Kotlinil. Selle peamiseks ülesandeks on väliste teekide ja sõltuvuste automaatne allalaadimine ja seadistamine. Gradle võimaldab ligipääsu teekidele Maveni ja Ivy repositooriumites. [9]

Põhjus miks antud projekti arenduses valiti ehituse tööriistaks just Gradle on tema oskus hallata mugavalt ühe ja sama ehituse raames erinevaid projekte või mooduleid. Kuna antud töös on projekt jagatud eraldiseisvateks mooduliteks, millest räägime täpsemalt hilisemas peatükis, siis on antud omadus väga kasulik. Juhul, kui me soovime parasjagu muuta ainult ühte moodulit ja teised ei muutu ei taha me, et projekti uuesti ehitamisel ehitatakse uuesti kõik moodulid – see on aja ja ressursi kulukas. Gradle oskab tuvastada millised ehituse osad ei ole muutunud ja ei hakka neid uuesti ehitama [10].

4.2 Java

Java programmeerimiskeel on platvormist sõltumatu, tüübikindel, objekt orienteeritud keel. Java lähtekood kompileeritakse üldjuhul baitkoodi ning käivitatakse JVM (Java Virtual Machine) poolt [11].

Projektis on kasutusel Java 11 versioon, mis tuli välja 2018. aasta septembris ning on hetkel viimane pikaajalise toega versioon [12].

Java sai valitud lähtuvalt autori enda eelistustest ja kogemustest keelega ning soovist kasutada Spring raamistikku.

4.3 Spring Boot

Spring raamistik on tarkvara rakenduse arendamise raamistik, mille funktsioone saavad kasutada ükskõik millised Javal baseeruvad rakendused, sõltumata platvormist. Spring

sisaldab erinevaid mooduleid, mis pakuvad teenuseid, mis lihtsustavad Java veebirakenduse arendust [13].

Spring Boot on Spring raamistiku osa, mis võimaldab väga lihtsalt luua iseseisvaid Springi rakendusi, mida saab väikese vaevaga käivitada. Spring Booti kasutamine vähendab olulisel määral erinevate konfiguratsioonide hulka, mida muidu oleks vaja selleks, et rakenduse eraldiseisvad osad koos töötaksid [14].

Otsus kasutada Spring Boot raamistiku lähtus sellest, et raamistik teeb tarkvara arenduse protsessi oluliselt mugavamaks ja lihtsustab selle protsessi – need mugavused tulevad eriti selgelt välja mitme mooduliga rakenduses nagu meie oma, kus on tarvis eraldiseisvad moodulid ja klassid omavahel koos toimima panna – Spring aitab lihtsustada klasside instantside loomist ja omistamist.

Käesoleva töö käigus realiseeritud rakenduses kasutame Spring baasmooduli pakutavaid funktsionaalsusi rakenduse erinevate osade sidumiseks – näiteks konfiguratsioonid vajalike klasside instantside loomiseks ja sõltuvuste haldamiseks. Lisaks kasutatakse Spring Security autentimist. Samuti on kasutatakse JDBC-id, et lihtsustada andmebaasiga suhtlemist.

4.4 PostgreSQL

Rakenduse andmebaas on realiseeritud PostgreSQLis, mis on võimas, avatud lähtekoodiga relatsioonilise andmebaasi süsteem. See sai alguse 1965. aastal Berkleley ülikooli POSTGRES-nimelisest projektist ning platvormi arendus on toimunud aktiivselt üle 30. aasta. PostgreSQLil on erinvaid omadusi, mis aitavad loodud rakenduse andmeid hästi ja mugavalt hallata, sõlumata andmemahust ning toetades ligipääsu suure hulga üheaegselt päringud tegevate kasutajate jaoks. Lisaks sellele, et tema kasutamine ei ole tasuline ja tal on avatud lähtekood, on PostgresSQL äärmiselt hästi laiendatav – ta võimaldab luua uusi andmetüüpe, automaatselt uuenevaid vaateid, realiseeritud vaateid, funktsioone, protseduure jpm [15, 16, 17].

PostgresSQL sai valitud seetõttu, et avatud lähtekoodiga tasuta andmebaasisüsteemidest on ta üks mitmekesisemaid ning vastas hästi arendatud rakenduse vajadustele.

4.5 Liquibase

Liquibase on avatud lähtekoodiga tarkvara rakenduse andmebaasi muudatuste haldamiseks. See töötab koos erinevate andmebaasisüsteemidega ning toetab mitmeid faili formaate. Tema peamine funktsionaalsus seisneb võimaluses viia andmebaas kiirelt ja mugavalt kindlasse faasi arendusprotsessis, ilma et me peaksime ise meeles pidama, milliseid muudatuste skripte me oleme konkreetses baasi instantsis juba jooksutanud ja milliseid mitte [18].

Liquibase kasutab skripte, mida nimetatakse changesetideks, et hallata andmebaasi muudatusi. Need skriptid võivad olla realiseeritud erinevate failiformaatidena nagu XML, JSON, YAML või SQL [18]. Töö käigus loodud projektis kasutatakse XML ja SQL faile.

Liquibase lihtsustab oluliselt arenduse käigus andmebaasi ülespanekut ja kindlasse seisu viimist ning on eriti kasulik projektis, kus on suur tõenäosus, et andmebaasi tabeleid on arendusprotsessi käigus tarvis redigeerida. Igakord, kui baasis on tarvis muudatusi teha, lisame juurde changesete. Kokkuvõttes peab eksisteerima üks põhi fail, mis peab järge kõigi changesetide üle ning Liquibase peab järge, millised nendest changesetidest on meie andmebaasis juba jooksutatud ja millised mitte. Liquibase'i baasi uuendamise käsu peale vaadatakse andmebaasi hetkeseisu, tuvastatakse millised skriptid on juba jooksutatud ning seejärel käivitatakse ülejäänud changesetid viies andmebaasi kiirelt ja mugavalt kõige uuemasse olukorda [18].

Just need kirjeldatud omadused ja eelised on põhjuseks, miks autor otsustas ka antud töös Liquibase'I kasutada. Kuigi andmebaasis on mugav ja lihtne teha muudatusi näiteks JetBrainsi rakenduse DataGrip kaudu, siis Liquibase'i eelis on see, et kõik muudatused on meile nähtavad. Kasutades näiteks DataGripi peaksime me eraldiseisvalt salvestama kõik tegevused, mida baasi muutmiseks tehtud on, kui soovime muudatuste üle hästi järge pidada.

4.6 Swagger

Swagger on avatud lähtekoodiga tarkvara raamistik, mis aitab disainida, ehitada, dokumenteerida ja kasutada REST APIsid [19].

Käesoleva töö raames valminud rakenduses kasutatakse Swaggerit selleks, et lihtsustada API kasutamist. Swagger oskab rakenduse HTTP teenuste põhjal genereerida lihtsakoelise kasutajaliidese, läbi mille on võimalik rakendust kasutada.

5 Rakenduse arhitektuur

Realiseeritud API lähtekood on jagatud 6 moodulisse. Järgnevalt on antud ülevaade iga mooduli eesmärgis ja sisesest struktuuris ning põhjendatud, miks moodulid on just selliselt realiseeritud.

5.1 Domain

Domain mooduli eesmärgiks on kirjeldada rakenduse keskmeks olevaid objekte, millega kogu rakendus tööd teeb. Antud moodulis ei ole realiseeritud loogikat. Moodul ise jaguneb omakorda neljaks paketiks:

- *entities* pakett sisaldab objekte, mis vastavad meie andmebaasi tabelitele ehk need on peamised äriobjektid.
- *models* paketis on lisaobjektid, mis on äriobjektide väljadeks, ning mille jaoks ei eksisteeri eraldiseisvaid andmebaasi tabeleid vaid need on Java objektid mis vastavad baasitabelite JSONB tüüpi väljadele.
- *enums* pakett sisaldab rakenduses kasutusel olevaid enumeid, ehk väärtusi, mida baasis hoiame lihtsustamise eesmärgil küll sõnena, kuid millel on kindlad lubatud väärtused.
- Viimasena on pakett *port*, mis sisaldab *entities* paketis kirjeldatud äriobjektidele vastavaid repository ehk andmepääsu kihi objektide liideseid. Nendes klassides ei ole realiseeritud andmepääsu kihi loogika vaid on kirjeldatud, millist funktsionaalsust peab see kiht toetama ehk milliste päringutega saame me andmebaasi poole pöördud. Selliste klasside eksisteerimine lähtub avatud-suletud printsiibist.

Domain moodul on kogu meie rakenduse keskmeks ning seda kasutavad kõik teised moodulid. Domain ise ei sõltu ühestki teisest moodulist.

5.2 Repository

Repository moodul sisaldab andmepääsu kihi loogikat. Antud moodulis on igale äriobjektile vastav DAO klass, mis implementeerib eelnevalt kirjeldatud Domain mooduli *port* paketis leiduvaid repository liidest (vt Joonis 3).

```
public class UserDao implements UserRepository {

    private JdbcTemplate jdbcTemplate;

    public UserDao(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
   ...
}
```

Joonis 3. Dao klassi näide

Meenutades S.O.L.I.D põhimõtteid siis liideste implementeerimine lähtub nii avatud-suletud printsiibist kui ka sõltuvuse inversiooni printsiibist. Kõik klassid ja moodulid, mis kasutavad andmepääsukihti ei tea meie DAO klassidest midagi vaid sõltuvad liidestest, mida need implementeerivad.

See tähendab, et kui me soovime välja vahetada andmepääsukihi loogika täiesti uue vastu – näiteks kui me vahetame andmebaasisüsteemi ning implemnteeritud DAO klassid ei toeta enam baasi poole pöördumist – siis me ei pea muutma kõiki mooduleid, mis repository moodulist sõltuvad. Selle asemel saame luua uued implementatsioonid Domain mooduli liidestele ning muuta konfiguratsiooni, mis vastutab klasside sõltuvuste instantside loomise eest.

Loodud DAO klassid kasutavad baasi päringute tegemiseks JdbcTemplate klassi. Antud klass hoolitseb andmebaasi ühenduse eest ja lihtsustab päringute vastuste tõlgendamist. JdbcTemplate'I kasutamiseks tuleb meil endal koostada SQL laused täpselt nagu andmebebaasis päringuid tehes, kuid antud klass, aitab päringute vatstuseid tõlgendada hiljem java objektideks, millega meie API tegutseda oskab. JdbcTemplate väärtus antakse DAO klassile läbi konstruktori (vt Joonis 3) ning sellest, kuidas DAO klaaside instantse luuakse räägime lähemalt Application mooduli juures.

5.3 Controller

Controller moodul on rakenduse esituskiht. Siin on realiseeritud rakenduse HTTP teenuste kirjeldused, mille poole teised rakendused või kasutajaliides pöörduda saavad. Nagu ka Repositorys on siin igale äriobjektile vastavalt üks controller klass, milles on kirjeldatud erinevad võimalikud HTTP päringud seoses selle objektiga.

Controller moodul on sõltuv UseCase moodulist, millele ta edastab päringute info põhjal käsklused ning saab vastu informatsiooni, mille põhjal koostada päringute vastused. Kasutatavad UseCase mooduli klassid on Controller klassi väljadeks ning nende instantside loomise eest hoolitseb Spring tänu @Resource annotatsioonile (vt Joonis 4). Selleks, et Spring oskaks meie UseCase klassidest instantse luua, eksisteerib meie rakenduses eraldiseisev konfiguratsiooni, mida tutvustame lähemalt

```
@RestController
public class UserController {

    @Resource
    private AuthContext authContext;
    @Resource
    private FindUser findUser;
    @Resource
    private CreateUser createUser;
    @Resource
    private UpdateUser updateUser;

    @GetMapping("/users/{id}")
    public ResponseEntity<?> findById(@PathVariable Long id) {...}

}
```

Joonis 4. Controller klassi näide

Lisaks controller klassidele asub antud moodulis ka AuthContext klass, mis on nö vahelüliks, et suhelda Spring Security klassidega ja hallata autentimist ja aktiivseid kasutajaid.

5.4 UseCase

UseCase moodul täidab äriloogika kihi rolli. Küll aga ei ole anud juhul äriloogika kiht realiseeritud Service klassidena nagu tüüpilises kolmekihilises arhitektuuris, mida varasemalt ka kirjeldasime. Struktuur, kus iga äriobjekti jaoks eksisteerib üks Service klass, kuhu on koodantud kogu andmete töötlemise loogika rikub ainsa vastutuse printsiipi.

Vaatame näitena ühte rakenduse äriobjektidest – Kasutaja ehk User. Tüüpilise controller-service-repository struktuuri juures tekiks meil äriloogikakihis üks klass UserService. See klass oleks vastutav igasuguse andmete töötlemise ja valideerimise eest nii juhul kui me soovime rakenduselt pärida millised kasutajad hetkel eksisteerivad, kui ka siis, kui me soovime lisada süsteemi täiesti uue kasutaja. Kasutaja lisamine ja pärimine on aga kaks täiesti eraldiseisvat ülesannet ning ainsa vastutuse printsiibi kohaselt peaks nad seega olema eraldi klassides. Üks UserService klass vastutaks liiga suure hulga funktsionaalsuse eest ning muutuks liiga keerukas, et selle sisu selgelt arusaadav oleks.

Seega oleme realiseeritud rakenduses jaganud UserService klassi kasutuslugude põhjal mitmeks osaks nii nagu kirjelda kasutuslookeskse arhitektuuri põhimõte. Eksisteerivate kasutajate pärimiseks vajaliku loogika eest, vastutab nüüd FindUser (vt Joonis 5) klass ning uue kasutaja loomise eest CreateUser klass.

```
public class FindUser {

    private UserRepository repository;

    public FindUser(final UserRepository repository) {
        this.repository = repository;
    }
   ...
}
```

Joonis 5. UseCase klassi näide

Kasutuslugude klassid on jaotatud äriobjektide põhjal pakettidesse – nii on Usecase mooduli struktuuri vaadates kohe arusaadav, milliste objektidega rakendus töötab ning milliseid tegevusi antud objektidega sooritada saab, mis on sellise arhitektuuri peamine eesmärk.

UseCase moodul sõltub ainult Domain moodulist. Kuigi UseCase pöördub kokkuvõttes Repository moodulis kirjeldatud DAO klasside poole, siis nende olemas olust ei tea ta tegelikult mitte midagi. Nagu varasemalt seletatud implemneteerivad kõik DAO klassid mõnda äriobjektile vastavat repository liidest, mis annab meile vabaduse teha muudatuse Repostory moodulis, ilma et need mõjutaksid UseCase moodulit, nagu peatükis 5.2 ka seletati.

Repository tüüpi klassid, millest UseCase klass sõltub antakse edasi läbi konstruktori – selle eest, millise Repository implementatsiooniga täpsemalt tegu on, hoolitseb Application mooduli konfiguratsioon.

Erinevalt Controller moodulist, ei sõltu UseCase moodul Spring Bootist, mis tähendab, et soovi korral on meil võimalik see siduda hoopis teistsuguse rakendusega. Seega kui tulevikus soovitakse sama funktsionaalsuse jaoks luua rakendus, mis Spring booti ei kasuta, siis äriloogika funktsionaalsust ei ole tarvis uuesti arendada – piisab sellest, kui teised rakenduse osad panna juba eksiteerivat UseCase moodulit kasutama.

5.5 Application

Application moodul on see, mis kõik ülejäänud osad koos tööle paneb. Hetkel on realiseeritud API Spring Boot rakendusena ning rakendust käivitav klass asub just siin moodulis. Ühtlasi asuvad selles moodulis kõik konfiguratsiooni ja seadete klassid. Kõige olulisem neist on SpringConfig (vt Joonis 6) klass, milles on realiseeritud loogika kasutusloo klasside instantside loomiseks ning mis otsustab ühtlasi selle üle, milliseid repository implementatsioone meie kasutuslugude klassid kasutama hakkavad.

```
@Configuration
public class SpringConfig {

	@Resource
	private JdbcTemplate jdbcTemplate;


	@Bean
	public FindUser findUser() {
	    return new FindUser(new UserDao(jdbcTemplate));
	}
}
```

Joonis 6. SpringConfig klassi lõik

Konfiguratsiooni klassi jaoks luuakse @Resource annotatsiooni abil JdbcTemplate klass, mille instantsi oskab Spring meie andmebaasi seadistuste abil iseseisvalt luua, ilma et me vajaksime lisanduvat seadistamist. Tuletades meelde Repository moodulit, siis JdbcTemplate oli seal vajalik DAO klasside loomiseks. UseCase'id loomiseks omakorda on meil vaja DAO klasside instantse. SpringConfig klassis eksiteerib iga kasutusloo klassi jaoks eraldi meetod, milles luuakse *new* märksõna abil selle klassi instants ning kõik DAO klasside, instantsid, mis selleks vajalikud on. @Bean annotatsioon nende meetodite ees ütleb Springile, et mujal tuleb nende klasside instantside loomiseks kasutada just neid meetodeid. Tänu selle oskab Spring luua vajalikud instantsid Controller klassidele läbi @Resource annotatsiooni nagu me peatükis 5.3 nägime.

5.6 Database

Database moodul on teistest eraldi, ning ei ole otseselt seotud rakenduse funktsionaalsusega vaid rakenduse andmebaasiga. Selle eesmärk on pigem arendusprotsessi ja rakenduse ülespanemise lihtsustamine mitte funktsionaalsuse täitmine. Antud moodulis leiduvad Liquibase'i skriptid ja seadistused selleks et luua baasitabelid ja viia baas soovitud olukorda.

Iga baasi tabeli ehk rakenduse äriobjekti jaoks eksisteerib SQL fail, milles on skriptid esialgse tabeli loomiseks ning kõikideks muudatusteks, mis arenduse käigus tabeliga sooritatakse. Sinna SQL faili saamegi jooksvalt lisada uusi skripte. Lisaks SQL failidele eksisteerib changelog.xml fail, kus asuvad vajalikud seadistused liquibase'i

jooksutamiseks ning kus defineeritakse, milliseid SQL skripte tuleb baasi uuendamisel arvestada.

Liquibase jooksutamiseks on loodud Gradle käsud, mida saab käsurealt käivitada ning, mis uuendavad vastavalt changelog failile andmebaasi. Lisaks baasi uuendamisel, saab liquibase käske kasutada ka näiteks andmebaasi tühjendamiseks.

6 Funktsionaalsed nõuded

Rakenduse eesmärgiks on võimaldada tudengielu rahastamise protsessi haldamist veebipõhisel platvormil. Käesolevas peatükis on lahtiseletatud rahastusplatvormi API funktsionaalsus lähtuvalt äriobjektidest ja kasutusjuhtudest. Peatüki 6.2 alapeatükkide nimede sulgudes olev inglisekeelne osa viitab äriobjekti või kasutusloo klassi nimele rakenduse lähtekoodis.

6.1 Privileegid

Rakenduse kasutajate tegevused on piiratud sõltuvalt nende privileegidest. Rakenduses eksisteerib 3 privileegi:

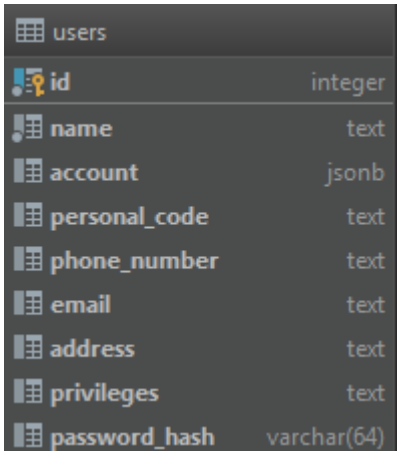
- Rakenduse administraator
- Konkursijuht
- ÜE juhatuse või revisionikomisjoni liige

Kasutajal võib olla rohkem kui üks privileeg. Kasutaja kellel pole ühtegi privileegi on tavakasutaja. Tavakasutajatel võib olla võimalik sooritada erinevaid tegevusi sõltuvalt nende seosest konkursside, avalduste või organisatsioonidega. Võimalikud tegevused on lahtiseletatud järgmises alapeatükis.

6.2 Funktsionaalsus

6.2.1 Kasutaja (User)

Rakenduse funktsionaalsuse kasutamiseks on vaja registreerida end kasutajaks. Kasutajale vastav andmebaasi tabel on kujutada Joonisel 7.



Joonis 7: Kasutaja äriobjektile vastav andmebaasitabel

Kasutajatega on seotud 3 kasutuslugu:

6.2.1.1 Kasutaja loomine (CreateUser)

Kasutaja loomiseks ei ole vaja privileege ega autentimist. Iga rakenduse külastaja saab endale luua kasutaja.

6.2.1.2 Kasutaja muutmine (UpdateUser)

Kasutajat saab muutmiseks on kaks olukorda:

- Kasutaja soovib ise redigeerida oma andmeid. Tal on võimalik muuta oma pangakonto infot, telefoninumbrit, meiliaadressi ja aadressi.
- Rakenduse administraator saab redigeerida kasutaja privileege – neid lisada või eemaldada. Administraator ei saa muuta iseenda õigusi.

6.2.1.3 Kasutaja pärimine (FindUser)

Kasutaja andmete pärimiseks on kaks olukorda:

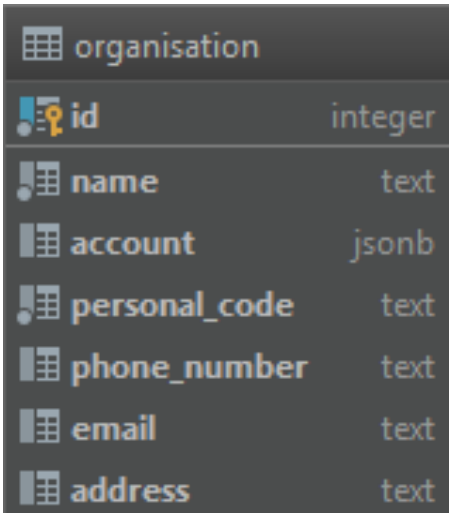
- Kasutaja ise saab pärida kõiki oma andmeid, et näha, mis info tema kohta süsteemis eksisteerib ning veenduda selle korrektsuses.
- Kõik kasutajad pärida kõigi teiste kasutajate kohta unikaalse id, nime ja isikukoodi. Tavakasutaja vajab seda projektitaotluse loomisel, et märkida projekti

liikmeid. Konkursijuht kasutab antud funktsionaalsust konkursikomisjoni koostamisel ning kasutaja, kellel on õigus muuta organisatsiooni liikmelisust, vajab seda uute liikmete lisamisel.

6.2.2 Organisatsioon (Organisation)

Tudengielu rahastamine on väga suures osas suunatud just tudengiorganisatsioonidel ning rakenduse funktsionaalsus on nendega tihedalt seotud.

Organisatsiooniga seotud andmebaasitabel on kujutad Joonisel 8.



Joonis 8: Organisatsiooni äriobjektile vastav andmebaasitabel

Organisatsiooniga on seotud 3 kasutuslugu.

6.2.2.1 Organisatsiooni lisamine (AddOrganisation)

Uue organisatsiooni saab lisada iga kasutaja. Organisatsiooni looja märgitakse automaatselt organisatsiooni liikmeks ning talle antakse organisatsiooni administraatori ja jälgija õigused.

6.2.2.2 Organisatsiooni muutmine (UpdateOrganisation)

Organisatsiooni andmeid saab muuta ainult kasutaja, kes on märgitud organisatsiooni liikmeks ning, kellele on märgitud organisatsiooni adminni õigused. Kõiki organisatsiooni andmeid (v.a unikaalne id) saab redigeerida.

6.2.2.3 Organisatsiooni pärimine (FindOrganisation)

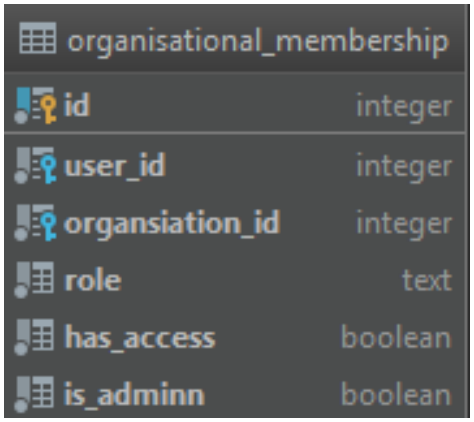
Organisatsiooni andmete pärimiseks on kolm olukord:

- Kõik kasutajad saavad pärida kõikide organisatsioonide kohta nende nime, selleks et kontrollida, millised organisatsioonid süsteemis registreeritud on.
- Kõik kasutajad saavad pärida kõiki organisatsiooni andmeid, mille liikmeks nad ise määratud on. Seda on vaja taotluse loomisel.

6.2.3 Organisatsiooni liikmelisus (OrganisationalMembership)

Organisatsiooni liikmelisus seob omavahel kasutajat ja organisatsiooni. Liikmelisuse info sisaldab seotud kasutaja ja organisatsiooni identifikaatoreid, kasutaja rolli organsatsioonis ning infot, kas kasutajal on jälgimisõigus ehk õigus näha kõiki organisatsiooniga seotud avaldusi, aruandeid ja liikmeid ning kas ta on organisatsiooni admin ehk, kas tal on õigus redigeerida organisatsiooni andmeid ning liikmelisust.

Organisatsiooni liikmelisusega seotud andmebaasitabel on kujutatud Joonisel 9.



Joonis 9: Organisatsiooni liikmelisuse äriobjektile vastav andmebaasitabel

Organisatsiooni liikmelisusega on seotud 4 kasutuslugu.

6.2.3.1 Organisatsiooni liikmelisuse lisamine (AddOrganisationalMembership)

Organisatsiooni liikmelisust saab lisada kasutaja, kes on ise organisatsiooni liige ning talle on määratud administraatori õigus.

6.2.3.2 Organisatsiooni liikmelisuse muutmine (UpdateOrganisationalMembership)

Organisatsiooni liikmelisust saab muuta ainult kasutaja, kes on määratud organisatsiooni administraatoriks. Redigeerida saab liikme rolli ja õigusi seos organisatsiooniga. Kasutaja ei saa endalt eemaldada administraatori õigusi.

6.2.3.3 Organisatsiooni liikmelisuse pärimine (Find OrganisationalMembership)

Organisatsiooni liikmelisuse pärimise jaoks on kaks olukord:

- Kasutaja saab pärida kõiki endaga seotud organisatsiooni liikmelisusi.
- Konkreetse kasutaja ja organisatsiooniga seotud liikmelisuse kohta saab pärida andmeid kasutaja ise või teine kasutaja, kes on märgitud organisatsiooni administraatori või jälgija rolli.
- Kasutaja, kellel on organisatsioonis jälgija või administraatori roll, saab pärida kõiki andmeid organisatsiooniga seotud liikmelisuste kohta.

6.2.3.4 Organisatsiooni liikmelisuse kustutamine (DeleteOrganisationalMembership)

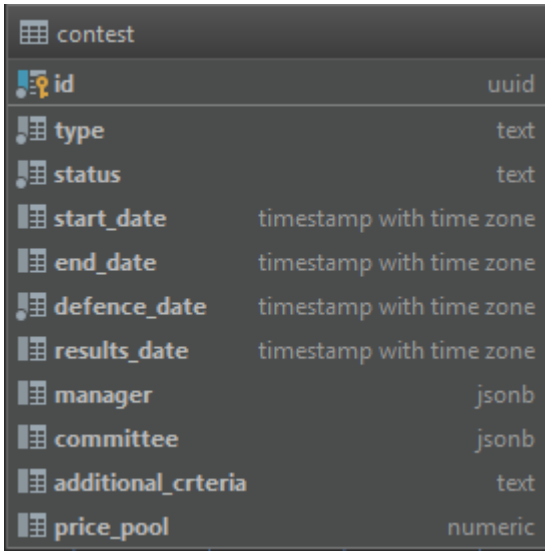
Liikmelisuse kustutamine toimub kahel juhul:

- Kasutaja ise soovib organisatsioonist lahkuda ning kustutada oma liikmelisuse. Kasutaja ei saa enda liikmelisust kustutada, kui ta on organisatsiooni ainus liige.
- Organisatsiooni liikmelisuse saab kustutada kasutaja, kellel on organisatsiooni administraatori õigused.

6.2.4 Konkurss (Contest)

Rakenduse põhieesmärgiks on konkursside haldamine. Konkursse on kokku 3 erinevat tüüpi: suurte projektide, väikeste projektide ja aastatoetuste konkurss, nagu Tudengielu rahastamise peatükis välja toodud.

Konkursile vastav andmebaasitabel on kujutatud Joonisel 10.



Joonis 10: Konkursi äriobjektile vastav andmebaasitabel

Konkurssidega on seotud 4 kasutuslugu.

6.2.4.1 Konkursi loomine (AddContest)

Uue konkursi saab luua konkursijuhi privileegidega kasutaja. Konkursi looja määratakse automaatselt loodud konkursi juhiks.

6.2.4.2 Konkursi muutmine (UpdateContest)

Konkursi saab muuta ainult konkreetse konkursi juht. Konkursi muutmiseks on kaks olukorda:

- Konkursi juht saab muuta kõiki konkursi andmeid – v.a unikaalne id, staatus ja konkursi juht – juhul, kui konkurss ei ole staatuse järgi märgitud lõpetatuks.
- Konkursi juht saab muuta konkursi staatust.

6.2.4.3 Konkursi pärimine (FindContest)

Kõik kasutajad saavad pärida kõiki konkursse, mis on staatuse järgi avalikud.

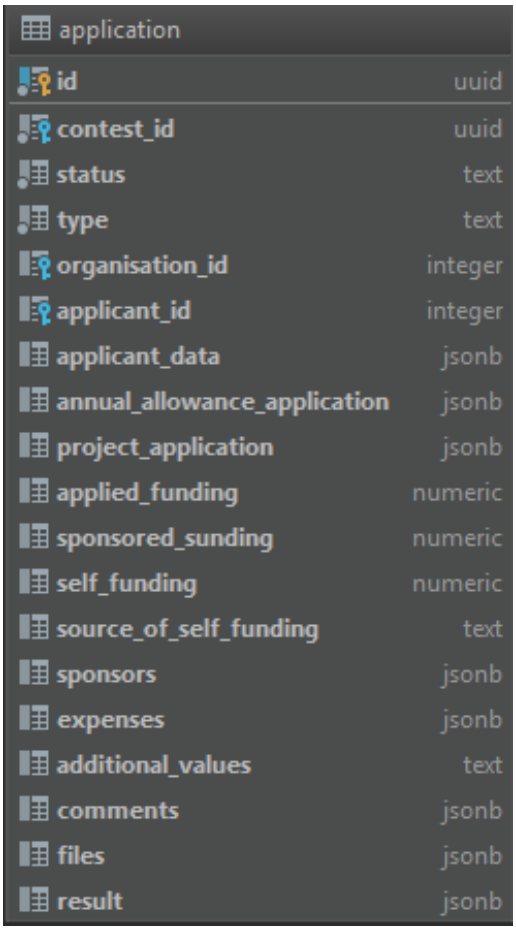
6.2.4.4 Konkursi kokkuvõtte genereerimine (GenerateResultsReport)

Kui konkursi tulemused on kinnitatud, saab iga konkursile esitatud avalduse tulemuste põhjal genereerida koondraporti.

6.2.5 Avaldus (Application)

Avaldused on alati seotud täpselt ühe konkursiga. Sõltuvalt avaldusega seotud konkursi tüübist on avaldusega seotud ka kas projektikonkurssi avalduse andmed või aastatoetuse avalduse andmed (vt ka Lisa 1 ja Lisa 2).

Avaldusele vastav andmebaasitabel on kujutaud Joonisel 11.



Joonis 11: Avalduse äriobjektile vastav andmebaasitabel

Avaldustega on seotud 3 kasutuslugu:

6.2.5.1 Avalduse loomine (AddApplication)

Uue avalduse saab luua iga kasutaja. Projektikonkursile avalduse esitamisel märgitakse avalduse looja automaatselt projektijuhiks. Aastatoetuse konkursile avalduse esitamisel märgitakse kasutaja automaatselt avalduse täitjaks.

6.2.5.2 Avalduse muutmine (UpdateApplication)

Avalduse muutmiseks on kaks juhtu:

- Avalduse esitaja saab muuta avalduse andmeid juhul, kui avaldus ei ole staatuse järgi lõplikult kaitsmisele esitatud.
- Konkursijuht saab muuta avalduse staatust ja lisada avaldusele kommentaare, kui avaldus on esitatud tema hallatava konkursi alla.

6.2.5.3 Avalduse pärimine (FindApplication)

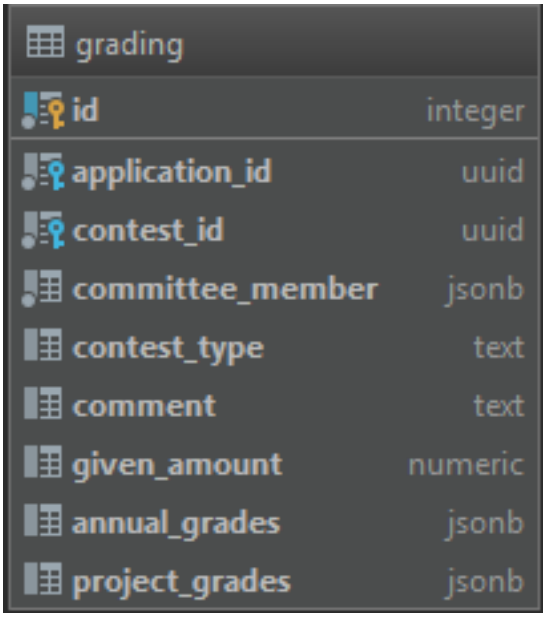
Avalduste pärimiseks on neli juhtu:

- Avalduse detailandmeid saab id pärida avalduse esitaja, kasutaja, kes on avaldusega seotud organisatsioonis jälgija õigustega, ÜE juhatuse või revisjonikomisjoni liige, avaldusega seotud konkursi juht ja komisjoni liikmed.
- Kasutaja saab pärida nimekirja kõikidest avaldustest, mille esitaja ta on.
- Organisatsioonis jälgimisõigustega kasutaja saab pärida nimekirja kõikidest avaldustest, mis organisatsiooniga seotud on.
- Konkursijuht ja komisjoni liikmed saavad pärida nimekirja kõikidest avaldustest, mis konkreetse konkursiga seotud on ja, mis ei ole staatuse järgi esitamata mustandid.

6.2.6 Hindamine (Grading)

Iga konkursi komisjoni liige annab kõigile konkursile edastatud avaldustele hinnangu.

Hindamisele vastav andmebaasitabel on kujutatud joonisel 12.



Joonis 12: Hindamise äriobjektile vastav andmebaasitabel

Hindamisega on seotud 3 kasutuslugu.

6.2.6.1 Hindamise lisamine (AddGrading)

Uue hindamise saab lisada ainult kasutaja, kes on märgitud hindamisega seotud konkursi komisjoni liikmeks ning, kes ei ole mingil põhjusel eemaldatud seotud avalduse hindamisest. Hindamist saab lisada ainult avaldusele, mis on staatuse järgi parasjagu hindamise järgus.

6.2.6.2 Hindamise muutmine (UpdateGrading)

Hindamist saab muuta ainult hindamise loonud kasutaja. Muuta saab ainult kommentaari, antud summat või avalduse punkte. Hindamist saab muuta ainult juhul kui sellega seotud avaldus on staatuse järgi hindamise faasis.

6.2.6.3 Hindamise pärimine (FindGrading)

Hindamise pärimiseks on kaks juhtu:

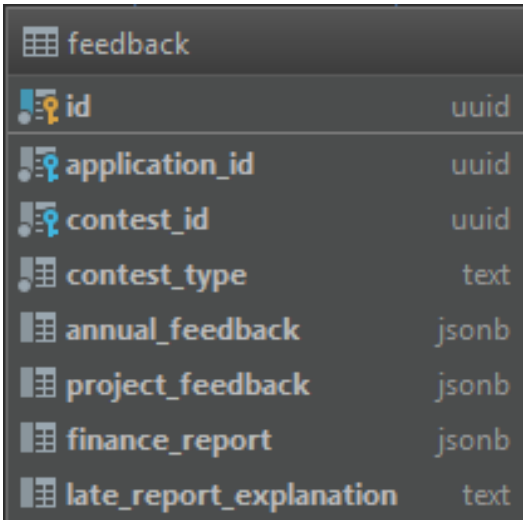
- Komisjoni liige, kes on hindamise loonud, saab avalduse id järgi pärida oma hindamist.

- Konkursijuht saab konkursi järgi pärida kõiki konkursi avaldustega seotud hindamisi.

6.2.7 Aruanne (Feedback)

Iga aruanne on seotud täpselt ühe avaldusega ja iga valdusega on seotud täpselt üks aruanne. Aruanne sisaldab endas aastatoetuse aruande või projektitaotluse aruande informatsiooni (vt Lisa 3 ja Lisa 4) sõltuvalt temaga seotud projekti tüübist, finants aruannet (vt Lisa 5) ja selgitust juhul, kui aruanne esitati peale tähtaega.

Aruandele vastav andmebaasitabel on kujutatud Joonisel 13.



Joonis 13: Aruande äriobjektile vastav andmebaasitabel

Aruandega on seotud 3 kasutuslugu.

6.2.7.1 Aruande loomine (AddFeedback)

Aastatoetuse aruande saab luua kasutaja, kes on aruandega seotud avalduse looja. Aruande saab luua ainult siis, kui sellega seotud avaldus on staatuse järgi toetuse saanud ja aruande esitamise faasis.

6.2.7.2 Aruande muutmine (UpdateFeedback)

Aruande muutmisega on seotud kaks juhtu:

- Aruande andmeid saab muuta selle loonud kasutaja. Andmeid saab muuta ainult juhul, kui aruanne ei ole esitatud staatuses.

- Aruande staatust muuta ja sellele kommentaare lisada, saab aruandega seotud konkursi juht.

6.2.7.3 Aruande pärimine (FindFeedback)

Aruande detailandmeid saab avalduse id järgi pärida aruande esitaja, kasutaja, kes on aruandega seotud organisatsioonis jälgija õigustega, ÜE juhatuse või revisjonikomisjoni liige, ning aruandega seotud konkursi juht. ÜE juhatuse või revisjonikomisjoni privileegiga kasutajad ja konkursijuht saavad aruannet pärida vaid juhul, kui see ei ole esitamata mustandi staatuses

7 Rakenduse edasi arendamine

Käesoleva töö käigus valminud rakendusliides täidab rahastusplatvormi põhifunktsionaalsust, kuid see ei ole piisav reaalselt kasutatava rahastusplatvormi rakenduse realiseerimiseks. Selleks, et rakendus oleks tulevikus reaalselt kasutatav tuleks sellele juurde arendada kasutajaliides. Kasutajaliidese arenduse käigus võib tulla välja erinevaid viise, kuidas eksisteerivat API-t oleks mõistlik täiendada. Üldine funktsionaalsus peaks jääma samaks, kuid see võib vajada mõningaid täiendusi ja täpsustusi eriti seoses erinevate validatsioonidega ja sellega, kui palju informatsiooni päringutele vastuseks tagastatakse – rakenduses leidub kohti, kus hetkel tagastatakse ilmselt rohkem infot, kui kasutajaliides tegelikult vajab.

Lisaks sellele leiab autor, et tulevikus võiks mõelda selle peale, kas valmiva veebiplatvormi jaoks piisab ainult sellest, et rakendus toetab praegust rahastusüsteemi või tuleks mõelda viisidel, kuidas see võiks olla paindlikum juhul, kui teatud tingimused rahastussüsteemis muutuvad – niiviisi ei peaks kõigi süsteemi muudatuste toetamiseks tegema tingimata jätkuarendust rakendusele. Selleks tuleks võimalused läbi arutada ÜE liikmetega ning üheskoos analüüsida, kas ja kuidas saaks rakendust muutuste osas paindlikumaks muuta ning millised on kõige tõenäolisemad muudatused süsteemis, millega rakendus kindlasti kohaned võiks.

8 Kokkuvõte

Töö eesmärgiks oli luua TalTechi Üliõpilasesinduse veebipõhise rahastusplatvormi jaoks uus rakendusliides, mis täidaks nõutud funktsionaalsusi ning vastaks puhta ja kasutuslookeskse arhitektuuri põhimõtetele. Loodud API on üks osa veebipõhisest rakendusest, mis lihtsustaks ÜE, tudengiorganisatsioonide ja tudengite jaoks rahastuse taotlemise protsessi.

Töö esimeses osa anti ülevaate API arenduse ja arhitektuuri põhimõtetest, millele rakenduse arhitektuuri välja töötades tugineti. Samuti tutvustati tudengielu rahastamise süsteemi Tallinna Tehnikaülikoolis. Selle osa eesmärgiks oli anda selge arusaam rakenduse taustast.

Teises osa eesmärgiks oli kirjeldada valminud rakenduse arhitektuuri ja funktsionaalsust. Samuti toodi välja selle edasi arendamise võimalused.

Töö koostamisel sai autor rohkelt uusi teadmisi puhta koodi ja arhitektuuri põhimõtetest ning õppis neid rakendama. Samuti sai autori uusi kogemusi mitme mooduliga projektide ehitamisest.

Töö tulemusena valmis töötav rakendusliides, mis täidab tudengielu rahastamise protsessi jaoks vajalikku funktsionaalsust. Selleks, et veebipõhine rahastusplatvorm aga tudengite ja organisatsioonide jaoks kasutatav oleks vajab see tulevikus kasutajaliidese arendust.

Kasutatud kirjandus

[1] M. Stowe, Undisturbed REST: A guide to designing the perfect API, San Francisco: MuleSoft, 2015.

[2] „Representational State Transfer," [Võrgumaterjal]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Kasutatud 20 5 2019].

[3] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software and Design, Pearson Education, Inc, 2018

[4] „3-Tier-Architecture," [Võrgumaterjal]. Available: http://www.tonymarston.net/php-mysql/3-tier-architecture.html. [Kasutatud 16 5 2019].

[5] „Tudengielu rahastamise struktuur," [Võrgumaterjal]. Available: https://www.ttu.ee/organisatsioonid/uliopilasesindus/tegevused/toetame/. [Kasutatud 1 5 2019].

[6] „Tudengielu rahastamise eeskiri," [Võrgumaterjal]. Available: https://www.ttu.ee/public/u/uliopilasesindus/Tegevus/Toetame/Projektikonkurss/Lisa_4 _Tudengielu-rahastamise-eeskiri.pdf. [Kasutatud 27 4 2019].

[7] „Rahastuse hindamisjuhend," [Võrgumaterjal]. Available: https://www.ttu.ee/public/u/uliopilasesindus/Tegevus/Toetame/Projektikonkurss/Rahastuse-hindamisjuhend.pdf. [Kasutatud 17 4 2019].

[8] „Aastatoetuste ja projektikonkursi aruandluse juhend," [Võrgumaterjal]. Available: https://www.ttu.ee/public/u/uliopilasesindus/Tegevus/Toetame/Aastatoetused/Aastatoetuste_ja_projektikonkursi_aruandluse_juhend.pdf. [Kasutatud 17 4 2019].

[9] „Gradle," [Võrgumaterjal]. Available: https://www.vogella.com/tutorials/Gradle/article.html. [Kasutatud 15 5 2019].

[10] „Miks Gradle on hea tööriist?," [Võrgumaterjal]. Available: https://www.theserverside.com/opinion/4-reasons-why-Gradle-may-be-the-right-Java-build-tool. [Kasutatud 15 5 2019].

[11] „Java programmeerimiskeel," [Võrgumaterjal]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html. [Kasutatud 5 5 2019].

[12] „Java versioonid," [Võrgumaterjal]. Available: https://www.oracle.com/technetwork/java/java-se-support-roadmap.html. [Kasutatud 5 5 2019].

[13] „Spring," [Võrgumaterjal]. Available: https://spring.io/projects/spring-framework. [Kasutatud 10 5 2019].

[14] „Spring Boot," [Võrgumaterjal]. Available: https://spring.io/projects/spring-boot. [Kasutatud 10 5 2019].

[15] „PostgreSQLi eelised I,“ [Võrgumaterjal]. Available: https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases/. [Kasutatud 10 5 2019].

[16] „PostgreSQLi eelised II,“ [Võrgumaterjal]. Available: https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases-part-ii/. [Kasutatud 10 5 2019].

[17] „PostgreSQL,“ [Võrgumaterjal]. Available: https://www.postgresql.org/about/. [Kasutatud 10 5 2019].

[18] „Liquibase,“ [Võrgumaterjal]. Available: https://blogs.oracle.com/shay/introduction-to-liquibase-and-managing-your-database-source-code. [Kasutatud 7 5 2019].

[19] „Swagger,“ [Võrgumaterjal]. Available: https://swagger.io/docs/specification/20/what-is-swagger/ . [Kasutatud 20 5 2019].

Lisa 1 – Aastatoetuse konkursi taotluse mall

I TAOTLEJA ANDMED		
Organisatsiooni või taotleja nimi		
Arvelduskonto omaniku nimi		
Arvelduskonto number		
Organisatsiooni või taotleja aadress (tänav/küla, linn/vald, postiindeks, maakond)		
Isiku- või registrikood		
Telefon	**E-posti aadress**	**Veebiaadress** (kui on)
Taotleja lühikirjeldus (Ülevaade taotleja tegevusest sh. seotus teiste organisatsioonidega ja/või kuulumine tudengiühendusse ning muu oluline informatsioon)		
Lühikokkuvõte taotleja strateegiast (visioon, pikaajalised eesmärgid)		
Plaanitavad ja/või käimasolevad projektid* (nimetada plaanitavate ja/või käimasolevate projektide nimed ja neid lühidalt kirjeldada)		
Plaanitava projekti nimi	**Toimumise ajavahemik**	**Lühikirjeldus**
	pp.kk.aaaa - pp.kk.aaaa	
Kirjeldada, millist lisandväärtust pakub taotleja tegevus TTÜ üliõpilaskonnale		

Finantseerimine	**Summa** (EUR)
Kogumaksumus	
Toetuseks taotletav summa	
Teistelt toetajatelt taotletav summa	
Omafinantseeringu summa	
Nimetada/tuua välja omafinantseeringu katmise allikad	

II AASTATOETUSE TEGEVUSED JA KULUD					
Aastatoetuse eelarve kululiikide lõikes (eurodes)					
Nr	**Kululiigid**	**Kulutuse põhjendus**	**Kulutüüp** (halduskulu, põhivara vms)	**Taodeldav summa**	**Kogu maksumus**
1					
2					
Aastatoetuse maksumus kokku					

III RUUMI TAOTLEMINE (täidetakse vaid juhul, kui organisatsioon soovib taodelda TTÜ üliõpilasesinduse halduses oleva ruumi)	
Kas teie organisatsioonil on oma ruum (kontor/tuba)?	**Kus teie ruumid paiknevad (ruumi tähis/number)?**
JAH/EI	
Ruumi kasutamise eesmärk (kirjeldada, milleks ruumi kasutatakse, missugune on ruumi kasutamise vajadus ja/või sobiliku ruumi omadused)	
Ruumi kasutamise otsene mõju organisatsiooni arengule (missuguseid eesmärke aitab täita, nimetada tegevused, mis ei saaks ruumi olemasoluta toimuda või kui teil oma ruumi veel ei ole, siis mille poolest muutub järgmise aasta tegevuskava ruumi saamisel)	
Liikmete ligipääs ruumile (kui palju inimesi ruumi kasutab, kellel on ligipääs ruumile ja/või kui teil oma ruumi veel ei ole, siis kellele plaanite ligipääsu anda)	
IV KOHUSTUSLIKUD LISADOKUMENDID (esitatakse digitaalselt taotlusvormiga koos)	
Lisadokumendid	
1	Organisatsiooni kuulutavate inimeste loend (kirjeldada ka arvuliselt ja protsentuaalselt, kui palju liikmeid on TTÜ tudengid ning kui palju neist on aktiivsed, passiivsed, vilistlased, välistudengeid jne)
2	Organisatsiooni põhitegevused tegevuskavana
V TAOTLUSE KINNITAMINE	
Taotleja kinnitus	
Kinnitan oma allkirjaga alljärgnevat:	
1) taotleja aktsepteerib ja täidab aastatoetuste konkursi eeskirja ning kõiki TTÜ ÜE otsuseid tingimusteta;	
2) kõik käesolevas taotluses esitatud andmed on õiged ja täielikud;	
3) kõik käesolevas taotluses esitatud dokumendid on kehtivad ja ehtsad.	

Taotleja nimi	**Allkiri**	**Kuupäev**
	Allkirjastatud digitaalselt	pp.kk.aaaa

VI TAOTLUSE REGISTREERIMINE (täidab taotluse vastuvõtja)	
TTÜ üliõpilasesinduse töötaja nimi	**Kuupäev**
	pp.kk.aaaa

Lisa 2 – Projektikonkursi taotluse mall

I TAOTLEJA ANDMED			
Organisatsiooni või taotleja nimi			
Arvelduskonto omaniku nimi			
Arvelduskonto number			
Taotleja või organisatsiooni juriidiline aadress (tänav/küla, linn/vald, postiindeks, maakond)			
Telefon	**E-posti aadress**	**Veebiaadress** (kui on)	
Projektijuht			
Nimi:			
Isikukood:			
Telefon:			
E-post:			
Projekti meeskond (projekti teostajad, kes on kaasatud projekti elluviimisesse)			
Nimi	**Vastutusala**	**Kas TalTech tudeng? (jah/ei)**	**Organisatsiooniline kuuluvus**
Projektijuhi ja -meeskonna kirjeldus (ülevaade projektijuhi ja peamiste projektimeeskonna liikmete varasematest tegevustest, sh ülevaade milliseid projekte on projektimeeskond varem korraldanud)			
II PROJEKTI ANDMED			
Projektikonkurss:			
Projekti nimi:			
Projekti sisu kokkuvõte (kirjeldada mis tegevusi projekt sisaldab, võimalusel lisada päevakava ning muu oluline informatsioon ürituse kohta)			
Projekti sihtgrupp			
Projekti eesmärgid ja oodatavad tulemused (kirjeldada projekti elluviimise eesmärke ja oodatavaid tulemusi, kuidas te nendeni jõuate ja millist muutust ja pikemaajalist mõju antud projekti elluviimise tulemusena soovitakse saavutada)			
Kirjelda, millist lisandväärtust pakub projekt TalTech üliõpilaskonnale? (Lähtuge TalTech Üliõpilaskonna arengusuundadest)			
Projekti alguskuupäev (tegevuste ja kulude abikõlblikkuse algus)		**Projekti tegevuste lõppkuupäev** (tegevuste ja kulude abikõlblikkuse lõpp, projekti maksimaalne kestvus on 12 kuud)	
pp.kk.aa		pp.kk.aa	
NB! Projekti alguskuupäev ei saa olla varasem kui projektitulemuste kinnitamiste kuupäev!			

NB! Enne projekti algust ja peale projekti lõppu ei tohi teha ühtegi projektiga seotud finantstehingut!	
Finantseerimine	**Summa** (EUR)
Kogumaksumus	
Toetuseks taotletav summa	
Teistelt toetajatelt taotletav summa	
Omafinantseeringu summa	
Nimetada/tuua välja omafinantseeringu katmise allikad	

Andmed käesoleva projektiga otseselt seotud teiste toetajate kohta (märkida ära ka need taotlused, mis on mistahes organisatsioonidele või firmadele esitatud, ent mille kohta pole veel finantseerimisotsust tehtud)		
Toetuse andja	**Rahalise toetuse puhul toetuse summa**	**Mitterahalise toetuse korral toodete/teenuste kirjeldus**
Kokku (EUR):		

III PROJEKTI TEGEVUSED JA KULUD			
Nr	**Projekti eelarve kululiikide lõikes (vajadusel lisada märkused, tähelepanekud)**	**Taodeldav summa**	**Kogu maksumus**
1			
2			
	Projekti maksumus kokku:		
Vajadusel selgita eelarvet			

Tegevuse nimetus (tegevused panna loogilises ajalises järjestuses)	**Tegevuse läbiviimise oodatav tulemus** (tulemus peab olema mõõdetav)	**Tegevuse periood**

V TAOTLUSE KINNITAMINE		
Taotleja kinnitus		
Kinnitan oma allkirjaga alljärgnevat:		
1) taotleja aktsepteerib ja järgib projektikonkursi eeskirju ning kõiki projektikonkursiga seotud TalTech ÜE otsuseid;		
2) kõik käesolevas taotluses esitatud andmed on õiged ja täielikud;		
3) kõik käesolevas taotluses esitatud dokumendid on kehtivad ja ehtsad.		
Projektijuhi nimi	**Allkiri**	**Kuupäev**
	Allkirjastatud digitaalselt	pp.kk.aaaa

Lisa 3 - Aastatoetuse konkursi aruande mall

I ARUANDE REGISTREERIMINE (täidab TTÜ üliõpilasesindus)		
Aruande registreerimise kuupäev:	pp.kk.aaaa	
Aruande registreerija nimi:		
II AASTATOETUSE ANDMED		
Aruande täitja nimi ja ametikoht	**Toetuse saaja nimi**	
Aruande täitja telefon	**Organisatsiooni või taotleja aadress**	
Aruande täitja e-posti aadress	**Taotlejaga seonduv veebiaadress (kui on)**	
Aruande esitamise kuupäev	**Finantsjuhi andmed (kui on)**	
pp.kk.aaaa		
Juhul kui aruanne ei ole TTÜ üliõpilasesindusele esitatud tähtaegselt, siis palun põhjendage aruande hilinemist.		
Ülevaade, kuidas toetus on aidanud TTÜ tudengielu edendada, tudengite aktiivsust, omaalgatust suurendada, tudengeid arendada jne		
Kinnitan, et kõik käesolevas aruandes esitatud andmed on õiged.		
Aruande täitja nimi	**Allkiri**	**Kuupäev**
	Allkirjastatud digitaalselt	pp.kk.aaaa

Lisa 4 – Projektikonkursi aruande mall

I PROJEKTI ANDMED		
NB! Aruannet peab täitma projektijuht!		
Aruande esitamise kuupäev		
pp.kk.aaaa		
Juhul kui aruanne ei ole TalTech üliõpilasesindusele esitatud tähtaegselt, siis palun põhjendage aruande hilinemist.		
II TEGEVUSARUANNE		
Kokkuvõte projekti tegevustest		
Tegevus (vastavalt taotlusele)	**Planeeritud tulemus** (vastavalt taotlusele)	**Tegelik tulemus**
Hinnang projekti elluviimisele (Kuidas kulges projekti elluviimine? Kas toimusid muudatused, missugused probleemid tekkisid ja kuidas neid lahendati? Milliseid uusi ideid tekkis?)		
Projekti sihtgrupini jõudmine (Kuidas jõuti projekti sihtgrupini, mida järgmine aasta paremini teha?)		
Eesmärgi saavutamine, tulemuste kokkuvõte (Kuidas projekt oma eesmärgi(d) täitis ja teie hinnang saavutatud tulemusele. Kuidas saaksite parandada tegevus(t)e või projekti elluviimist järgnevatel aastatel?)		
Projekti tulemuste jätkusuutlikkus, finantssuutlikus ja mõju (Millist mõju avaldas tehtud investeering organisatsiooni arengule? Milliseid jätkutegevusi on vaja tulemuste mõju kestmiseks? Kas taotluses kavandatud eelarve (tulud ja kulud) osutus realistlikuks?)		
Millist lisandväärtust pakkus projekt TalTech tudengkonnale?		
Kinnitan, et kõik käesolevas aruandes esitatud andmed on õiged.		
Aruande täitja nimi	**Allkiri**	**Kuupäev**
	Allkirjastatud digitaalselt	pp.kk.aaaa

Lisa 5 – Finants kokkuvõtte mall

I PROJEKTI KULUDE ARUANNE					
Aruanne palun täita eurodes 1 sendi täpsusega (st. kaks kohta peale koma)					
Nr	**Kululiigid**	**Esialgses eelarves ettenähtud kulud EUR**	**Esialgses eelarves ettenähtud kulud EUR**	**Tehtud kulud KOKKU EUR**	**Eelarve jääk EUR**
1.					
	Kokku				
Kommentaarid					

II PROJEKT FINANTSEERIJATE LÕIKES				
	Algselt taotletud finantseerimine EUR	Tegelikult eraldatud toetus EUR	Kasutatud toetus KOKKU EUR	Eelarve jääk EUR
TTÜ ÜE-lt taotletav finantseerimine				
Teistelt toetajatelt taotletav finantseerimine				
Omafinantseerimine				
Finantseerimine kokku				

	Algselt ettenähtud finantseerimine %	Tegelik finantseerimine KOKKU %	Erinevused %
TTÜ ÜE-lt taotletav finantseerimine			
Teistelt toetajatelt taotletav finantseerimine			
Omafinantseerimine			
Kokku			

Enne allkirjastamist veenduge aruandes esitatud andmete õigsuses!		
Kinnitan, et kõik käesolevas aruandes esitatud andmed on õiged.		
Kinnitan, et käesolevas aruandes kajastatud kulud on eranditult aastatoetuse põhised, täies osas tasutud ning selgelt eristatavad ja nõuetekohaselt dokumenteeritud aastatoetuse taotleja raamatupidamises.		
Aruande täitja nimi	**Kuupäev**	**Allkiri**
	pp.kk.aaaa	Allkirjastatud digitaalselt