TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl-Erik Hein 185753IAIB
Artur Kerb 185127IAIB
Robin-Kevin Koppa 163917IAPB

TalTechi seiklusmäng

Bakalaureusetöö

Juhendaja: Evelin Halling
PhD

Tallinn 2021

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Karl-Erik Hein, Artur Kerb, Robin-Kevin Koppa

18.05.2021

Annotatsioon

Antud bakalaureusetöö eesmärgiks oli luua funktsionaalne GPS-baasil põhinev seiklusmäng. Seda rakendust saab kasutada, et luua ning osaleda mängudes, millel on mitu erinevat mängu tüüpi. See projekt on peamiselt loodud Java, Spring Booti, PostgreSQLi, Angulariga, OpenLayersi ning Bootstrapiga.

Käesolev dokument koosneb tehtud töö ülevaatest, kasutatud tööriistadest, rakenduse haldamisest ja selgitusest ning sisemisest loogikast kui ka tulevikuplaanidest.

Rakenduse eesmärk on lahendada probleeme ning täita lünki, mis jättis eelnevalt kasutatud sarnane rakendus, mis oli tasuline ning ei omandanud teatud funktsionaalsust. Meie rakenduses on lisa funktsionaalust, nagu iga tiimile oma punktide määramine. Samas on võimalik seda tulevikus edasi arendada.

Selle projekt töö tulemusena oleme saavutanud valmis rakenduse, kus tiimid saavad osaleda mängudes, kus nad peavad kaardi peal punkte korjama vastavalt sellele, mis mängutüübiga on tegemist.

Käesolev lõputöö on valminud meeskonnatöö tulemusena, kus iga liige panustas võrdselt ning osales rakenduse loomisel, kujundamisel ning dokumentatsiooni kirjutamisel. Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 93 leheküljel, 9 peatükki, 55 joonist, 0 tabelit.

Abstract

TalTech Adventure Game

The objective of this bachelor's thesis was to create a functional GPS-based game. This application can be used to create and play a game with several different game modes. This project was developed mainly using Java, Spring Boot, PostgreSQL, Angular, OpenLayers and Bootstrap.

This thesis contains an overview of the work that has been done, an overview of the tools used, the explanations and the inner mechanics of the application and what else could be implemented in the future.

This application solves the problem that the previous application that Tallinn University of Technology had used was costly and lacked some notable features such as assigning different points to teams. The application can also be developed further in the future according to the wants and needs of TalTech.

The outcome of this project allows the user to log in, create, delete and customize a room, create, delete and update locations, questions and their respective sets. The user can bind these sets together to pairs and create pair sets, which can be assigned to playing teams. The user can activate the room and teams can join using a generated code. The teams will have to go to different points and answer different types of questions. The user has an overview of the current game and can see the history of different games.

This thesis was written as a team project, where every member has contributed equally to different parts of the project. Every member has been a part of designing, creating and documenting the created game.

The thesis is in Estonian and contains 93 pages of text, 9 chapters, 55 figures, 0 tables.

Lühendite ja mõistete sõnastik

Account	Kasutaja objekt
Active Room	Aktiveeritud ruumi objekt
API	*Application Programming Interface*
Back-end	Andmete juurdepääsu kiht rakenduses
Boolean	Tõeväärtus andmetüüp
C	Programmeerimiskeel
C++	Objektorienteeritud programmeerimiskeel
CI/CD	Pidev integratsioon
Commit	Git käsklus
DELETE	*HTTP* päring
DTO	*Data transfer object*
DPI	*Dots per inch*, punkti tolli kohta
Endpoint	Andmete ligipääsu aadress
Entity	Entiteet, üksus
Front-end	Visuaalne ning esindav kiht rakenduses
GET	*HTTP* päring
Git	Versioonhaldussüsteem
GitLab	Rakenduse arendamise platvorm
GPS	*Global Positioning Service*
HTTP	*Hypertext Transfer protocol*
Issue	Ülesanne
JavaScript	Programeerimiskeel
List	Nimekiri, Java objekt
Long	Java andmetüüp
Merge	Git käsklus
Milestone	Verstapost
MVC	*Model, View, Controller*
Stack	Tehnoloogiate kuhi
Optional	Java objekt
POST	*HTTP* päring
Push	Git käsklus
PUT	*HTTP* päring
Service	Teenus

String	Java objekt
REST	*Representional State Transfer*
Reverse proxy	Aitab muidu kättesaamatut *endpointi* kätte saada
Request	Päring
Repositoorium	Varamu, kus säilitatakse asju
Runner	GitLabi käskjalg
TalTech	Tallinna Tehnikaülikool
TTÜ	Tallinna Tehnikaülikool
TypeScript	JavaScriptil põhinev programmeerimiskeel
Workflow	Töövoog

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tallinna Tehnikaülikool ehk TTÜ kasutas tihti tiimi mängudeks 360kraadi poolt väljaarendatud GPS-il põhinevat seiklusmängu, kuid sooviti mängule lisa funktsioone, aga 360kraadi loojad ei suutnud neid soove ellu viia. Selleks et neid soove ellu viia pöörduti tudengite poole *Tarkvara meeskonnaprojekti* aine raames, kus anti 6-liikmelisele tudengi rühmale antud ülesanne, et luua 360kraadi sarnane seiklusmäng.

Tarkvara meeskonnaprojekti aines ei suudetud antud projektis implementeerida kõiki funktsionaalsusi ning seetõttu oli meil võimalus alles jäänud 3-liikmelisest tiimist edasi arendada seiklusmängu TTÜ-le.

Tallinna Tehnikaülikoolis Informaatika eriala mängitakse esmakursusel kaardil põhinevat mängu, kus pead ringi liikuma ning punktides küsimustele vastama. Kõige kiiremad ning õigesti vastanud mängijad võidavad. Tallinna Tehnikaülikoolile ehk TalTechile tuleb see mäng aga teiselt firmalt, 360kraadi, mis osutus kalliks ning võib jääda oma omaduste ja funktsioonide poolest ka puudulikuks. Me nägime võimalust teha enda versioon sellest, mis on just mõeldud TalTechi õppejõududele ning on vastavalt nende vajadustele tehtud.

1.1 Probleem ja projekti eesmärk

TTÜ kasutas tudengite jaoks tiimi ühtsuse loomiseks seiklusmängu 360kraadi, kuid see muutus TTÜ-le piiravaks nimelt sooviti teatuid funktsionaalseid osi lisada mängudele, kuid 360kraadi ei suutnud seda pakkuda ning ei olnud enam taskukohane. Lisaks ei saanud TTÜ ise mänge luua, vaid pidid juhiseid edasi andma 360kraadile.

1.2 Lühiülevaade realiseeritud funktsionaalsusest

Projekti käigus loodi seiklusmäng, mis koosneb veebirakendusest, andmebaasist ning taga taustal olevast loogika kihist.

Projekti käigus realiseeritud funktsionaalsused:

- Projekt on seatud üles leheküljel *https://skoop.cs.taltech.ee/avaleht*.
- Kasutaja saab sisse logida Tallinna Tehnikaülikooli kasutajaga.
- Kasutaja saab luua, muuta ning kustutada ruumi.
- Kasutaja saab kolmest erinevast mängutüübist valida ühe ning seadistada seda ruumile.
- Kasutaja saab lisada kaardistikke, kus asuvad küsimuste punktid (markerid).
- Kasutaja saab lisada küsimustikke, millega on seotud küsimused.
- Kasutaja saab luua kas automaatselt või manuaalselt paare, millega on seotud küsimused ning kaardistikud.
- Kasutaja saab lisada paaridele kogumikud, mis saab tiimidele hiljem määrata.
- Kasutaja saab aktiveerida ruumi, selleks et mängijad saaksid ruumiga ühineda ja mängida.
- Mängulooja saab jälgida mängu progressi, vaadates oma seadme kaardilt tiimide liikumist ning jälgides tiimide vahelist hetkelist punktiskoori.
- Kasutaja saab vaadata arhiveeritud mänge ning sellega seotud informatsiooni.
- Mängija saab mänguga liituda ning endale tiimi teha.
- Mängija saab vastata küsimustele ning koguda punkte.
- Mängija näeb enda ja punktide asukohta kaardil.
- Mängija saab võistelda teiste tiimide vastu.

1.3 Töö edasine struktuur

Töö on jaotatud 9 osaks. Peatükis 1 esitatakse sissejuhatus. Peatükis 2 kirjeldatakse projekti täpsemalt. Peatükis 3 esitatakse ülevaade kasutatud tööriistadest. Peatükis 4 esitatakse töö põhiosa. Peatükis 5 esitatakse valideerimine. Peatükis 6 esitatakse tulemused. Peatükis 7 esitatakse kommentaarid ning peatükis 8 järeldused ning edasised sammud. Viimases peatükis 9 tehakse kokkuvõte.

2 Projekti kirjeldus

Tarkvara meeskonnaprojekti aines alustas 6-liikmeline meeskond projekti. Peale seda jätkas kolmeliikmeline meeskond seda edasi. Projekt ei olnud mängitavas seisus. Üles oli seatud GitLab CI/CD, mis ehitas projekti Tallinna Tehnikaülikooli serverisse. Projekti oli paika pandud erineva raamistikud ja programmeerimis keeled ehk *stack*.

Taga taustal oleva kihi ehk *back-endi* poole pealt oli tehtud algne andmebaas, kuhu oli baas loogika sisse ehitatud. Olid tehtud repositooriumid, mis suhtlesid teenustega. Samuti olid tehtud kontrollerid, mis vahendasid REST API ning teenuste vahel informatsiooni. Lisatud olid ka *data transfer objectid*, mis andsid informatsiooni edasi kontrollerist ning võtsid seda vastu. Algsed testid oli paika pandud ning testiti projekti muudatuste korral. OAuth2 võtmed olid kätte saadud ning autentimisest oli tehtud esimene versioon, mis vajas veel tööd.

Visuaalsel kihil ehk *front-endi* poolel oli tehtud algne lehekülg, kus oli võimalik luua küsimusi, asukohti, küsimustikke ning GPS-i asemel oli võimalik mängimisel kaardi peale vajutada ning see kuvas küsimuse. Oli olemas väga algne mäng, kus oli puudu palju funktsionaalsust. Kasutajaliides ei olnud kõige moodsam ning vajas palju muudatusi. Mängu loomise töövoog ehk *workflow* oli ka muutunud võrreldes algse variandiga ning hetkeseis on palju arusaadavam.

2.1 Eesmärgid

Peale ainet *Tarkvara meeskonnaprojekt* oli tarvis teha palju muudatusi.
Meie meeskonna liikmete arv kahanes ning töö maht oli suurem, kuid seadsime iga nädal uusi ülesandeid, mida üritasime lahendada. Tekkisid iganädalased sprindid, kus jaotasime tööd võrdselt ära ning igaüks tegi oma osa. Tehtud tööd näitasime iga nädal juhendajale ette ning vastavalt tagasisidele tegime muudatusi.

Üks põhieesmärkidest oli saada mäng sellisesse faasi, kus oleks juba võimalik telefoniga välja minna ning vastavalt asukohale aktiveerida punkti, mis võimaldaks kasutajal vastata küsimusele.
Kui põhieesmärk oli saavutatud, saime hakata vastavalt kliendi soovidele teha suuri muudatusi mänguga. Kliendi soove rahuldasime mängutüüpide ning uue vaadete moel.

Esimeseks suuremaks muudatuseks oli veebilehe disain, kuna tegemist oli TalTech-ile loodava mänguga, siis oli vajalik ka TalTechi sarnast värvikombinatsiooni.

Teiseks sooviks oli ühekordselt võetav punkt. Kui üks tiim mängus vastab punkti ära, siis see kaob ka teiste mängu kaardilt ära ning seda enam vastata ei saa. Klient soovis, et siin oleks kaks valikuvarianti. Esimese variandina võiks punkt kaduda ära valesti ainult vastates ning teise variandina punkt kaoks ära nii õigesti kui ka valesti vastates. Selle lahenduseks meie tegime *rush* mängu tüübi. Hetkeseisuga on see mängu tüüp seadistatud nii, et küsimused kaovad ära nii õigesti kui ka valesti vastates.

Kliendi kolmandaks sooviks oli, et iga tiim saaks punkti võtta ühe korra. Selle lahenduseks oli *normal* mängu tüüp. Iga tiim saab mängu minnes eraldi kaardi ning eraldi punktid. See on nii-öelda individuaalmäng tiimi peale. Lõpptulemusena tiim siiski võistleb teiste tiimide vastu õigesti vastatud küsimustega ning ajaga.

Neljandaks kliendi sooviks oli, et punkte saaks määrata konkreetsetele tiimidele. Meie lahendus sellele oli *preassigned* mängu tüüp. Sellel mängu tüübil on tiimide piirang, sest iga tiim nõuab individuaalselt punktide kogumikku. Punkte peab eraldi seadistatama erinevatele tiimidele. Antud projektis on võimalik teha punktide kogumikust koopiad ning vastavalt muuta neid koopiad oma tahtmisele. Tiimidele antakse koopiad paaride kogumitest mängu minnes.

Klient lisaks soovis, et oleks võimalik saada ülevaade hetkel toimuvast mängust. Antud projektis on see võimaldatud. Mängu loojal on võimalik vaadata aktiivses ruumis olevate tiimide viimati salvestatud asukohti, punktide arvu ning kõik mängus olevaid markereid koos küsimustega.

Kuna mängu püsti panemine on algselt üsnagi keerukas, siis klient soovis, et oleks mingisugune õpetus, mis näitab mida ning kuidas midagi tegema peab. Õpetus alustab sellega, et tutvustatakse kasutajale kuidas oma esimest mängu luua.
Esmaks luuakse ruum seejärel lisatakse kaardi peale punktid, kus hakatakse küsimusi näitama. Järgnevalt õpetuse abil luuakse küsimustik, kuhu sisestatakse vastav arv küsimusi. Kui küsimustik on loodud, suundutakse punktide genereerimise lehele, kus seatakse igale punktile vastav küsimus. Kui see on tehtud, minnakse aktiveerimise lehele, kus genereeritakse kasutajal vastav ühinemiskood, selleks et mängijad saaksid mängima hakata.

Samuti on klient soovinud, et me teeme oma kasutajaliidest ning *user flowi* ehk kasutajavoogu projekti jooksul selgemaks ning arusaadavamaks. Me oleme üleliigseid nuppe kustutanud ning asju koondanud suurematesse kogumitesse

Vastavalt kliendi soovile lisasime võimaluse ise luua paare kaardile. Selleks lõime uue kaardi, kuhu saab kasutaja kaasa anda kaardistiku ning seal olevatele asukohtadele/markeritele saab kaasa anda küsimuse või ära kustuda neid. Kui automaatsel punktide genereerimisel saab anda valida küsimusi ühe küsimustikust, siis manuaalsel saab valida kõigest küsimustikust, mida kasutaja on loonud.

Soovisime implementeerida võimalust kiiresti luua ruum ning kohe mängima hakata. Selleks lõime automaatse paaride genereerimise. Kui anda automaatsele genereeriale üks küsimustik ja kaardistik, siis vastavalt nendele genereerib see kasutajale valmis paarid ning ta saab kohe mängida hakata.

2.2 Projekti haldus

Projektis on olnud tähtsal kohal tehtud töö dokumenteerimine ning korrektne versioonihaldus koos erinevate harude tegemisega ning *merge requestide* üle vaatamisega. Meie projektis on *merge requeste* peaaegu alati teine tiimiliige üle vaadanud enne peaharusse lisamist.

Projekti tegemise jooksul on tehtud GitLabi repositoorumitesse iga nädalaselt uusi harusid, uusi ülesandeid *issue'sid* ning *milestone'e* ehk pikaajalisemaid eesmärke. Töö on

jaotatud niimoodi, et kaks inimest tegelevad visuaalse ning esinduse kihiga ehk *front-endiga* ning üks inimene tegeleb andmete päringu ning töötluse kihiga ehk *back-endiga.*

Projekti jooksul on toimunud projekti liikmete vahel pidev kommunikatsioon. Kui *front-endi* tiimil on olnud midagi vaja, siis see on kiiresti *back-endi* implementeeritud. Kõik projekti liikmed on aktiivselt osa võtnud projekti arendamisest ning selle haldamisest. Iga tiimi liige on aktiivselt pannud *issue'dele* töö jaoks kulunud aega ning on sidunud oma tööaja ära vastava GitLabi haruga ning *milestone'iga*.

Projektis on olnud kommunikatsioon tähtsal kohal ning suuremate probleemide lahendamine on toimunud tavaliselt tiimiliikmete aruteluna.

Projektis on *merge requestid* üle vaadatud teiste tiimi liikmete poolt ning nende poolt kinnitatud ja peaharudesse *merge'itud.*

Iganädalaselt on seatud peale kliendi ning juhendajaga kohtumist uus eesmärke. Iganädalaselt on tehtud nende eesmärkide jaoks uued *milestone'id* GitLabi, kuhu saab märkida *issue'sid*.

Iga tiimiliige loob endale *issue* ning vastutab, et see tehtud saab. Tihti koos *issue* loomisega, genereeritakse ka uus haru peamisest harust, kus hakatakse tööd tegema. *Issue´dele* on ka võimalik märkida ennustatud ajakulu ning ka tegelik ajakulu. See on peamine tundide kirja panemise viis. Aja kirja panemisel arvestame ka arutelu ning ka dokumentatsioonide lugemist, mitte ainult puhta koodi kirjutamist.

3 Kasutatud tarkvara/tööriistad

Projekti realiseerimise käigus on läinud vaja erinevaid tarkvara, programmeerimiskeeli, raamistikke, teeke ning lisasid. Projekti algseadistus pandi paika aines *Tarkvara meeskonnaprojekti* ning see on suuresti samaks jäänud, sest antud projekti jooksul pole siiamaani olnud vajadust neid muuta ega midagi suurt juurde lisada.

Valisime all nimetatud tehnoloogiad just sellepärast, et olime enne tuttavad sellega, kuidas nad töötavad ning otsustasime, et oleks antud projektis keskendus arendamisel, mitte uute tehnoloogiate õppimisel.

3.1 Angular

Angular [1] on arendusplatvorm, mis on üles ehitatud TypeScript'is. Platvormina, Angular sisaldab:

- Komponendipõhist raamistikku järk-järsult laienevate veebirakenduste loomiseks
- Hästi integreeritud teekide kogu, mis hõlmab paljusid erinevaid funktsioone, sealhulgas vormide haldamine, klient-server suhtlus ja muu
- Arendaja tööriistade komplekt, mis aitab koodi arendada, koostada, testida ja värskendada.

Angulari abil saab kasutada platvormi, mis võib ulatuda ühe arendaja projektidest ettevõtte taseme rakendusteni. Angular on loodud värskendamise võimalikult lihtsaks muutmiseks, nii et oleks võimalik kasutada kõige uuemaid võimalusi minimaalse vaevaga. Angulari ökosüsteemi kuulub üle 1,7 miljoni arendaja, raamatukogu autori ja sisu looja.

3.2 TypeScript

TypeScript [2] on avatud lähtekoodiga programmeerimiskeel, mis on põhineb JavaScriptil, mis on üks maailma enim kasutatavaid tööriistu, sest ta lisab staatilise tüübi määratlused. *Types* ehk tüübid võimaldavad kirjeldada objekti kuju, pakkudes paremat dokumentatsiooni ning võimaldades TypeScriptil kontrollida, kas kood töötab õigesti.

3.3 Java

Java [3] programmeerimiskeel on üldotstarbeline, samaaegne, klassipõhine, objektorienteeritud keel. See on loodud piisavalt lihtsaks, et paljudel programmeerijatel oleks võimalik keelt sujuvalt mõista. Java programmeerimiskeelega on lähedalt seotud C ja C++.

3.4 OpenLayers

OpenLayers [4] on puhas JavaScripti teek kaardi andmete kuvamiseks enamikus kaasaegsetes veebibrauserites, ilma serveripoolsete sõltuvusteta. OpenLayers rakendab rikaste veebipõhiste geograafiliste rakenduste loomiseks JavaScripti API-d, mis on sarnane Google Mapsi API-dega, ühe olulise erinevusega - OpenLayers on vaba tarkvara, mis on välja töötatud avatud lähtekoodiga tarkvara kogukonna jaoks.

3.5 Gradle

Gradle [5] on avatud lähtekoodiga projekti ehitus tööriist, mis on loodud piisavalt paindlikuks, et ehitada peaaegu igat tüüpi tarkvara. Järgnev on kõrgetasemeline ülevaade selle mõnest olulisemast funktsioonist:

- Suur jõudlus ning kiirus
- JVM - ehitatud suuresti Java peal ning Java platvormi jaoks
- IDE tugi

3.6 Spring Boot

Spring Boot [6] on avatud lähtekoodiga Java raamistik. Sinna on kerge lisada lisa raamistikke, mis lihtsasti ühilduvad ning seal on olemas stardikomplekte, et rakendus saada võimalikult kiiresti üles. Samuti on seal sisseehitatud serveri võimalus, sisseehitatud testimise võimalused ning erinevad kasulikud andmete mõõdikud.

3.7 PostgreSQL

PostgreSQL [7] on avatud lähtekoodiga objekt-relatsiooniline andmebaas, mida on üle 30 aasta aktiivselt arendatud ning mis on tuntud oma kindluse, töökiiruse ning mitmekesiste funktsioonide poolest.

3.8 Flyway

Flyway [8] [9] on avatud lähtekoodiga andmebaasi migratsiooni tööriist. See on tuntud oma kasutuslihtsuse ning konventsioonide poolest. Sellel on 7 baas-käsklust ning käsklusi on võimalik saab teha nii SQL keeltes kui ka näiteks Javas. Siin projektis on see kasutuses Gradle lisana.

3.9 Docker

Docker [10] [11] on avatud platvorm rakenduste arendamiseks, saatmiseks ning jooksutamiseks. See lubab eraldada iseseisvaid rakendusi ehk aplikatsioone (näiteks eraldiseisev andmebaas) programmi infrastruktuurist, et oleks võimalik kiiresti ja mugavalt enda koodi kasutusele võtta. Docker pakendab rakendusi eraldi isoleeritud keskkonnas, mida kutsutakse konteineriks. Isolatsioon ning turvalisus võimaldab jooksutada mitmeid konteinerid samaaegselt ühel masinal. Konteinerid ei ole ressursi nõudlikud ning nad sisaldavad juba kõike, mida on jooksutamiseks vaja, nii et pole vaja muretseda eelnevalt installitud programmide pärast.

3.10 Git

Git [12] on tasuta avatud lähtekoodiga versioonihaldussüsteem, mis võimaldab käsitleda nii väikseid kui ka väga suuri projekte kiirelt ning efektiivselt.

3.11 GitLab

GitLab [13] [14] on arendamise platvorm, mida kasutab Tallinna Tehnikaülikool. Seal on sisseehitatud Giti funktsionaalsust, lisaks saab paigaldada sinna *pipeline'i, issue'sid* ning *milestone'e,* mis on kasulikud projekti haldamises ning dokumenteerimises.

GitLabis [15] on sisseehitatud tööriist GitLab CI/CD [16], mis võimaldab koodi pidevalt ehitada, testida ning kasutusele võtta uut koodi muudatustega. Selline iteratiivne protsess aitab vähendada võimalust uute *bugide* ehk vigade tekkimiseks.

3.12 Bootstrap

Bootstrap [17] [18] on tasuta avatud lähtekoodiga väga populaarne CSS raamistik, et arendada dünaamiline ning mobiili-toega veebilehti. Bootstrapis on suur erinevate komponent hulk, mis teeb lehekülgede disainimise ning nende komponentide asukohtade nihutamise väga mugavaks. See kasutab ka JavaScripti erinevate komponentide töötamiseks ning on lihtsasti kasutatav nii Angulariga kui ka teiste raamistikega.

3.13 Ubuntu

Ubuntu [19] on Linuxi operatsioonisüsteem, mis on tasuta saadaval. See on avatud lähtekoodiga, saadaval mitmetest erinevates versioonides ning on kasutuses nii serverites kui ka lauaarvutites. See toetab erinevaid protsessorite arhitektuure ning sellele on saadaval palju erinevat tarkvara, mis on kasulik tarkvaraarenduse protsessis.

Ubuntu operatsioonisüsteem on kasutuses tiimile antud Tallinna Tehnikaülikooli serveriga.

3.14 NGINX

NGINX [20] on avatud lähtekoodiga tarkvara, mis on kujundatud kasutuseks veebi serverina, *reverse proxyna* (suudab mitte kättesaadavaid aadresse kättesaadavaks teha) ning koormuse haldajana maksimaalse jõudluse ning stabiilsuse eesmärgil. See on väga populaarne, sest see on tasuta, lihtsasti ülesseatav ning see skaleerub hästi minimaalsete riistvara nõuetega. NGINX serveerib tiimi veebirakendust ja API-t ning haldab turvalisust.

3.15 Spring Web

Spring Web [21] on Springi raamistik, kuhu on sisse ehitatud Spring MVC, REST API ning Tomcat vaikimisi manustatud server. See aitab projekti kiiresti testida ning tööle saada. Selles projektis peamiselt kasutatakse seda REST API ning serveri jaoks.

3.16 Spring Security

Spring Security [22] on Springi raamistik. See lisab Spring-baasil põhinevatele rakendustele väga kohandatava autentimise ning turvalisuse. See raamistik keskendub Java rakendustele volituste ning autentimise jagamisele.

3.17 Spring Data JPA

Spring Data JPA [23] on osa Spring Datast, mis on Springi raamistik. See teeb lihtsaks JPAl põhinevate repositooriumite implementeerimise. Need repositooriumid võimaldavad lihtsalt suhtlemist andmebaasidega ilma manuaalsete SQL päringute kirjutamiseta.

3.18 OAuth 2.0

OAuth 2.0 [24] on tööstuse-standard protokoll volitamiseks. See võimaldab projektil saada volitust näiteks Microsoft Azurest või GitHubist vastavate võtmete abil. Kasutaja andmed ei ole nähtavad ning on seega turvalised.

3.19 JUnit

JUnit [25] on populaarne testimise raamistik Java jaoks. See on populaarne testide-keskse arendamise metoodikaga ning koosneb mitmest ühik testimiste raamistikust, mida kollektiivselt kutsutakse xUnitiks. See on tasuta ning avatud lähtekoodiga ning pakub mitmeid mugavaid võimalusi testide kirjutamiseks ning soovitud tulemuste saamiseks ja kontrollimiseks.

3.20 Swagger

Swagger [26] on avatud lähtekoodiga tööriist API arendamise jaoks. Selle komponent Swagger-UI [27] võimaldab visualiseerida ja suhelda olemasoleva API-ga. See on kollektsioon HTML, Javascriptist ning CSSi varadest, mille abil genereeritakse automaatselt lihtsasti arusaadav ning interaktiivne dokumentatsioon.

3.21 Alternatiivid

Siin on erinevad raamistikud, mida tiim kasutusele ei võtnud koos põhjendustega. Peamiste põhjenduste hulka kuulusid, et tiimil puudus varasem kokkupuude loetletud raamistikega.

3.21.1 React

React [28] on deklaratiivne, tõhus ja paindlik JavaScripti teek kasutajaliideste loomiseks. See võimaldab teil koostada keerukaid kasutajaliideseid väikestest ja eraldatud koodi juppidest, mida nimetatakse komponentideks. React on *front-end* raamistik JavaScript baasil, mis arendati välja Facebook' poolt aastal 2011 selleks, et luua kasutajaliideseid.
React kasutatakse üheleheliste rakenduste loomiseks. See järgib komponendi põhist lähenemist, mis aitab luua korduvkasutatavaid kasutajaliidese komponente.
React kasutatakse keeruka ja interaktiivse veebi- ja mobiilse kasutajaliidese väljatöötamiseks. Reactil on suur toetajate kogukond.

Reacti mitte kasutusele võtt oli tingitud sellest, et keegi meeskonnas polnud sellega varem kokku puutunud. Tiimiliikmed olid põhiliselt kokku puudunud Angulariga ning nagu varem mainitud ei tahtnud aega kulutada uue tehnoloogia õppimisel, vaid seda kulutada projekti arendamisele.

3.21.2 Vue.js

Vue [29] on progressiivne raamistik kasutajaliideste loomiseks. Erinevalt teistest monoliitsetest raamistikest on Vue algusest peale kavandatud järk-järgult kasutatavaks. Põhi teek on keskendunud ainult vaatekihile ning seda on lihtne kätte saada ja integreerida teiste raamatukogude või olemasolevate projektidega. Teisest küljest on Vue suurepäraselt võimeline töötama ka keerukaid ühe lehe rakendusi, kui seda kasutatakse koos kaasaegsete tööriistade ja tugi teekidega.

Vue.js ei võetud kasutusele peamiselt sellepärast, et tiimiliikmetel puudus sellega suurem kokkupuude ning tiimiliikmed ei pidanud vajalikuks hakata seda õppima hakata projekti töö käigus. Võrreldes ka funktsionaalsusi, pakkus Angular rohkem, kui Vue.js.

3.21.3 Aurelia

Aurelia [30] on avatud lähtekoodiga kasutajaliidese JavaScripti raamistik, mis on loodud ühe lehe rakenduste loomiseks, mis ei käitu raamistikuna. See on ehitatud algusest peale, kasutades tänapäevaseid tööriistu, toetades täielikult TypeScripti. Selle arhitektuur on rida koostööd tegevaid teeke ja veebikomponente, mis on nii-öelda kootud lihtsa kaasaegse JavaScripti abil. Aurelia eesmärk on aidata teil luua rakendusi brauseri, mobiilseadmete või töölaua jaoks.

Aurelia mitte kasutusele võtt oli tingitud selles, et tegemist on uudse *frameworkiga*, millel pole niivõrd suurt kogukonda võrreldes näiteks Angulariga ja kokkupuude on vähene tiimiliikmete seas.

3.21.4 Node.js

Node.js [31] võimaldab JavaScripti käivitada väljaspool brauserit. Selle tagajärjed on JavaScripti jaoks üsna suured. Node võimaldab kasutada JavaScripti kohalikus arvutis toimingute tegemiseks, milleks muidu läheks vaja, näiteks Python, C ++ või Java, kuna JavaScripti pole enam brauseriga seotud.

Node.js ei võetud kasutusele sellepärast, et tahtsime luua *backendi* Java baasil. Tiimi liikmed arvasid, et Javat on kergem hiljem hallata teiste arendajate poolt, sest see on tüübi range keel, erinevalt JavaScriptist. Tiimi liikmete arvates on Javal on suur kogukond ning sellel on saadaval palju erinevaid raamistikke ning teeke arendamiseks. Nad olid varem sellega kokku puutunud erinevate kursuste jooksul ning teadsid, et see on töökindel programmeerimiskeel ning nad saaksid keskenduda projekti ehitamisele, olles keelega ja selle nüanssidega juba varem tuttavad.

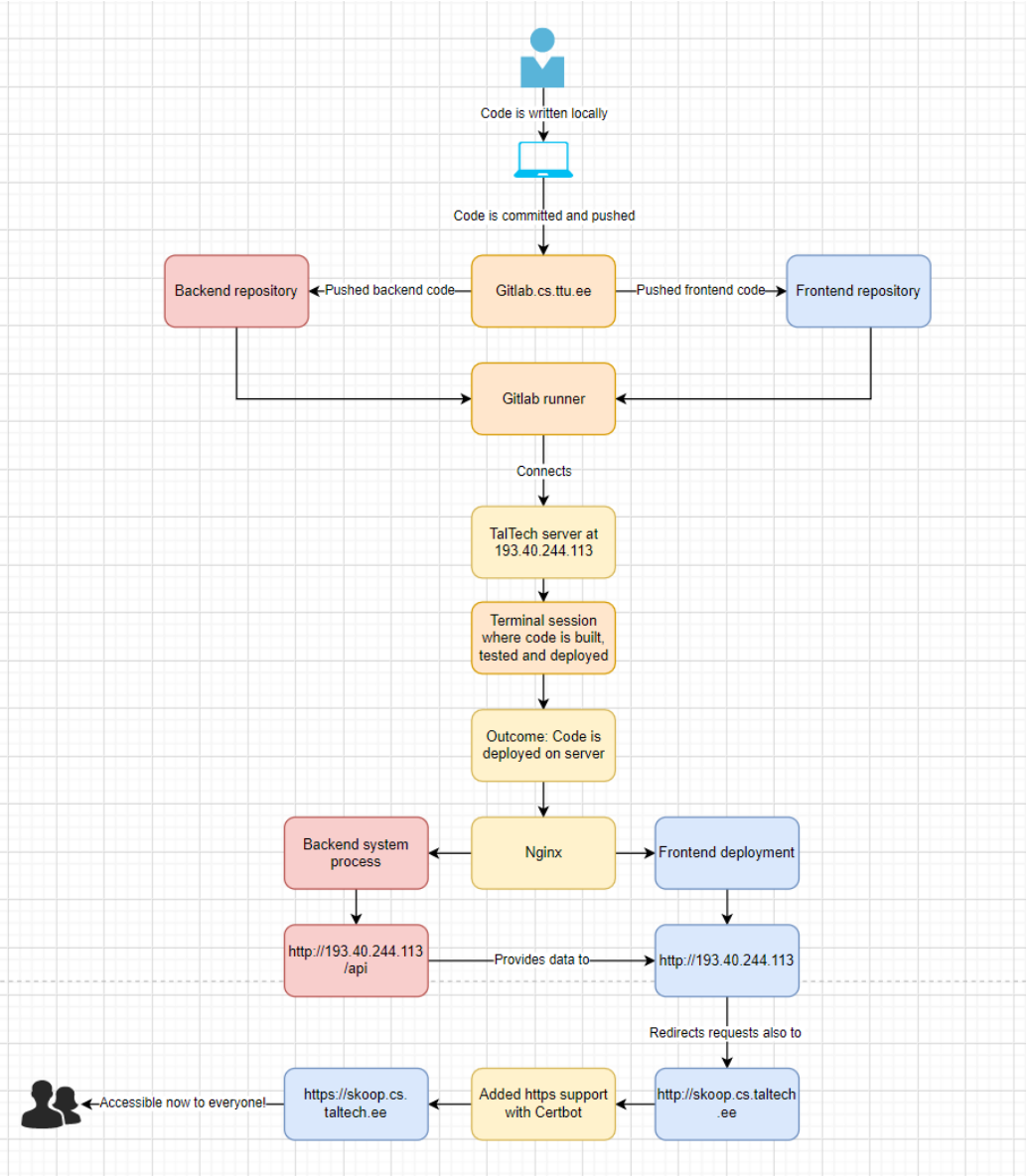
4 Projekti sisu

Antud peatükis tutvustatakse lähemalt *front-end* ja *back-endiga*. *Front-end* sisaldab erinevaid veebilehe vaateid, kuhu on lisatud iga elemendi tegevused. *Back-endis* on detailselt lahti seletatud, kuidas on erinevad raamistikud üles seadistatud, kuidas on üldine struktuur tehtud, kuidas toimib sisene ja väline suhtlus *front-endiga* ning mis erinevate objektidega tegeletakse ja nende nüansid.

4.1 Projekti seadistus

Kõik kood on saadaval kahes repositooriumis. *Back-end* on saadaval repositooriumis: *https://gitlab.cs.ttu.ee/360kraadi/360kraadi-back-end* ning *front-end* on saadaval repositooriumis: *https://gitlab.cs.ttu.ee/360kraadi/360kraadi-front-end*.

Koodile kasutusele võtt asub Joonisel 1.



Joonis 1. Koodi kasutusele võtmise protsess.

Projekti kasutusele võtmine ehk *deployimine* seati üles aine *Tarkvara meeskonnaprojekti* raames. Kood *commititakse* ning *pushitakse* kas *back-endi* või *front-endi* GitLabi repositooriumisse. Koodi kohale jõudas vaatab GitLabi *runner* (kummalgi repositooriumil on eraldi *runner*) vastavalt oma repositooriumile .gitlab-ci.yml faili, kus on kirjas täpne tegevusjuhend, et projekti kasutusele võtta. Kui kood on töödeldud eelnevalt mainitud GitLab *runneri* juhendi järgi, siis NGINX serveerib *front-endi* aadressile *https://skoop.cs.taltech.ee/* eelseadistatud kaustast ning *back-endi* aadressile *https://skoop.cs.taltech.ee/api* Linuxi *service'i* abil.

4.2 Front end

Antud peatükis käsitletakse lähemalt *front-end* raamistike, mida kasutusele võeti. Lisaks tutvustatakse veebilehe disaini osa, millele peamiselt lähtuti nagu värvide kombinatsioone ning milline näev välja *user flow* kui kasutaja navigeerib veebis.

4.2.1 Kasutajaliidese disain

Kasutajaliidese realiseerimiseks võtsime natukene innovatsiooni TalTech kodulehelt. Arvestasime peamiselt esinevate värvidega nagu roosa ja valge, mis on TalTechile omaseks saanud.

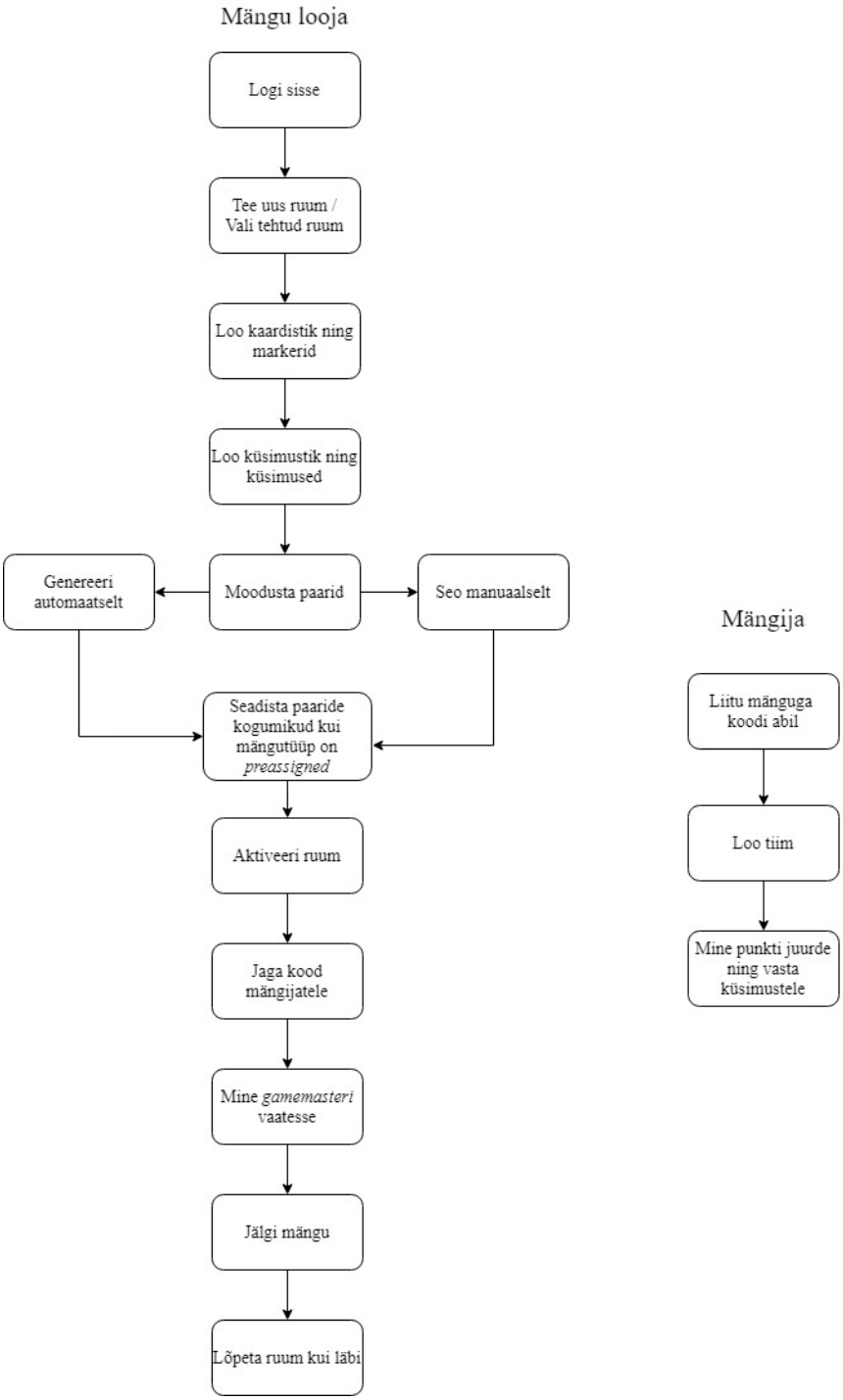
Mängijatele proovisime muuta kasutajaliidese võimalikult lihtsaks ja mugavaks, et mänguga ühinemine ja lisada tiimide moodustamine oleks võimalikult arusaadav. Mängimise ajal on neil võimalik vaadata oma hetkest punktiskoori koos alles jäänud mänguajaga, navigeerimiseks saavad näha oma seadmest kaarti, mis näitab seadme hetkest asukohta ja kõike saadaval punkte. Vajadusel saavad nad kaardil vajutada nupule, et mängija oleks kaardi keskel, kui liigutakse ringi.

Kasutajaliidese loomise käigus lähtusime sellest, et *workflow* oleks enamjaolt mugav ja arusaadav võrreldes eelmise versiooniga ning seetõttu oli tarvis ellu viia palju muudatusi. Mänguloojat tervitatakse ruumi loomise lehel varakult õpetusega, mille abil on võimalik samm-sammult järgida, kuidas luua ruumi kuni selle aktiveerimiseni.

Peamised muudatused võrreldes eelmise versiooniga oli uuem disain, mis sobituks rohkem TalTechi stiiliga, kasutaja *flow* oleks arusaadavam, et nupud oleksid selgesti nähtavad ja mida need teevad.

4.2.2 User flow

Kasutajate tegevusel on kindel järjekord ehk *user flow*, et oleks võimalik luua mängitav mäng ja et mängijal oleks võimalik loodud mängu mängida. See järjekord on kirjeldatud joonisel 2.



Joonis 2. User flow

4.2.3 Mobiilivaade (*Responsive*)

Mobiilivaade on ülesehitatud sedasi, et kasutaja kes mängib mängu või on ühinemas mänguga saab seda teha oma seadmest. Mängija saab sisestada ühinemiskoodi.

Seejärel lisada oma tiiminimi ja tiimiliikmed nimeliselt.

Mängu on võimalik luua ka mobiilivaates, kuid soovituslikult oleks parem teha seda arvutivaatest, kuna nii on mugav jälgida loomise protsessi suuremalt ekraanilt.

4.2.4 Tutorial

Tutorial ehk õpetus on selleks, et kui mängulooja esmakordselt soovib mängu luua, siis tervitatakse teda õpetus *pop-up*'iga, kus kirjeldatakse ruumi loomise etappe ning seejärel tehakse samm-sammult läbi erinevad etapid.

Esmalt luuakse ruum, kuhu tuleb sisestada ruumi nimi, mängutüüp, vajadusel tiimiliikmete maksimum arv ja ajalimiit, et kaua võib mängida.

Teiseks koos õpetuse juhendamisel liigutakse ruumi muutmise lehele, kus antakse mänguloojale info mida antud lehel on võimalik teha ning tehakse koos järgmiselt ära kaardistik, kuhu tuleb lisada punktid, millesse tulevad hiljem küsimused.

Järgnevalt, kui kaardistik on loodud suunatakse küsimustike lehele, kus luuakse *question set* ehk küsimustik, mis hoiab endas küsimusi, sellele antakse nimi ja minnaks küsimuste loomise juurde. Seal on kiire õpetus, kuidas ja mis tüüpe küsimusi on võimalik luua ning nendeks on valikvastustega, mitme vastuse variandiga ja tavaline vasta oma sõnadega küsimus.

Kui küsimustik on loodud õpetuse juhendamisel, suundutakse paaride genereerimise lehele, kus on võimalik siduda punktid küsimustega ära. Kahe variandi vahel on võimalik valida genereerimiseks kas automaatselt ehk sinu eest tehakse suurem osa tööst ära ja seotakse juhuslik küsimus ära punktiga, või manuaalselt ehk saad ise oma küsimused valikuliselt panna mingile punktile.

Punktidele küsimused lisatud, minnakse taaskord juhendamisel aktiveerimise lehele, mis aitab mängu viia mängijateni ehk genereeritakse vastav kood mängule, aga ennem on vaja anda aktiivsele ruumile nimi selleks, et kasutajal oleks tulevikus paremini aru saada, mis mäng hetkel on aktiveeritud. Aktiveeritud mängu on võimalik pealt vaadata ja hiljem ära lõpetada.

4.2.5 Kaart (OpenLayers)

Mängu realiseerimiseks on vajalik kaart, mis suudab kuvada kõik vajalikud asukohad ning ka näidata mängija/kasutaja asukohta. Iga kord, kui keegi avab kaardi ükskõik mis vaates, siis küsib rakendus luba kasutajalt tema hetke asukohta saada. Rakendus ei tööta korrektselt, kui ei saada seda kätte. Mängimisel on isegi võimatu liikuda, kuna rakendus ei saa uusi asukohti kätte.

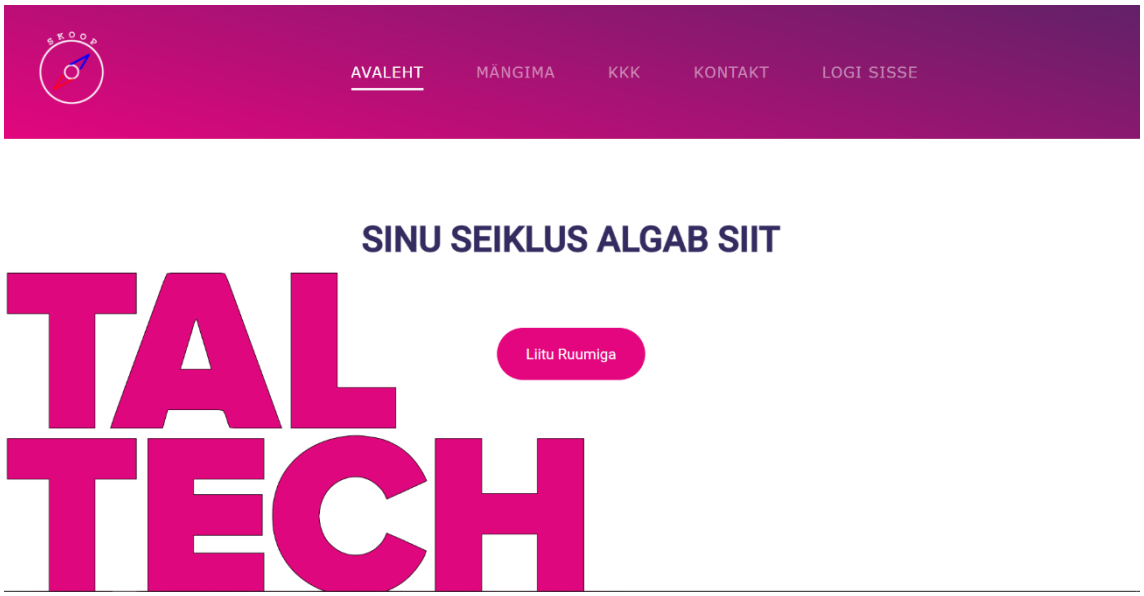
Kaardil on ka eristatud mängija ning markerid. Mängimisel iga kord rakendus kontrollib, kas mängija hetkeasukohal on markereid lähedal, mis vastavad paaride aktiveerimis kaugusele.

Kaart pakub ka lisa funktsionaalsusi. On võimalik nii sisse kui ka välja suumida, saab vabalt lohistada kaarti, et uurida ümbrust. Mõnel kaardivaatel on mingi funktsionaalsus, mida teistel ei ole. Mängimisel saab hoida enda asukohta kaardi keskel, mängu vaatlemis saab vajutada meeskonna peale ning see viib tema asukohani, paaride genereerimisel saab kaardil markeri peale vajutades kas luua paare või neid kustutada.

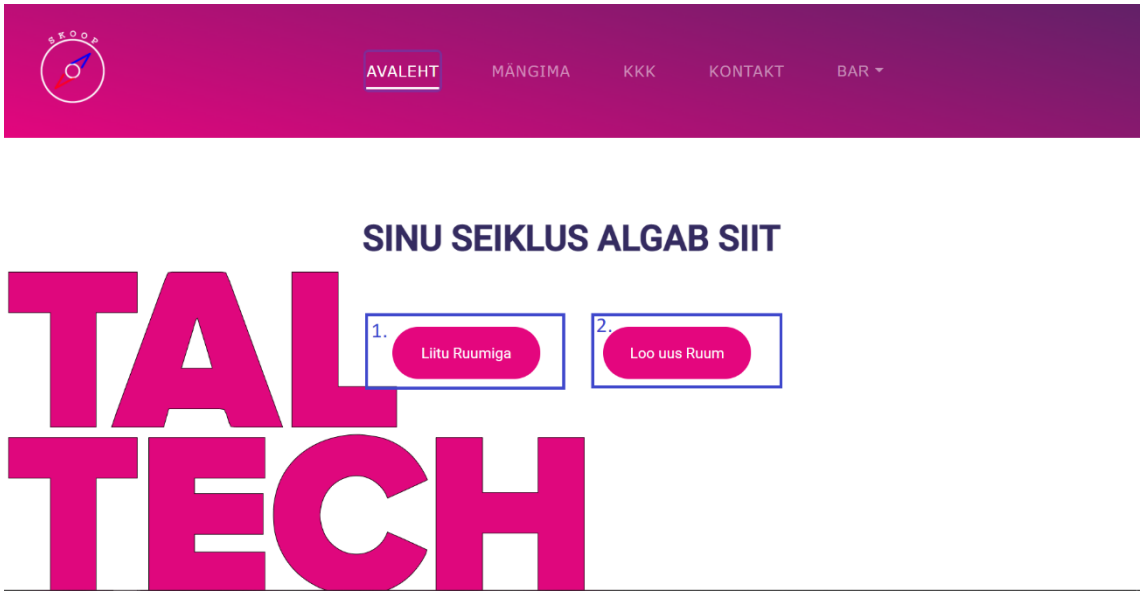
OpenLayers ei ole täiuslik, ning ka seal esineb tõrkeid. Miskipärast ei suuda see mõnes veebilehitsejas leida mängija asukoha ning paneb alati sama asukohta. Seda probleemi on esinenud nii Microsoft Edgis kui ka Brave Browseris. Kuid kui telefonis avada kaart, siis on kõik töökorras.

4.2.6 Avaleht

Avaleht on vaade, mis koosneb tavakasutajale ühise mänguga nupuga. Mänguloojale kuvatakse mängu ühinemise kõrvale ruumi loomise nupp. Joonistel 3 ja 4 on vastavalt näidatud Avalehe vaade.



Joonis 3. Tavakasutaja avalehe vaade

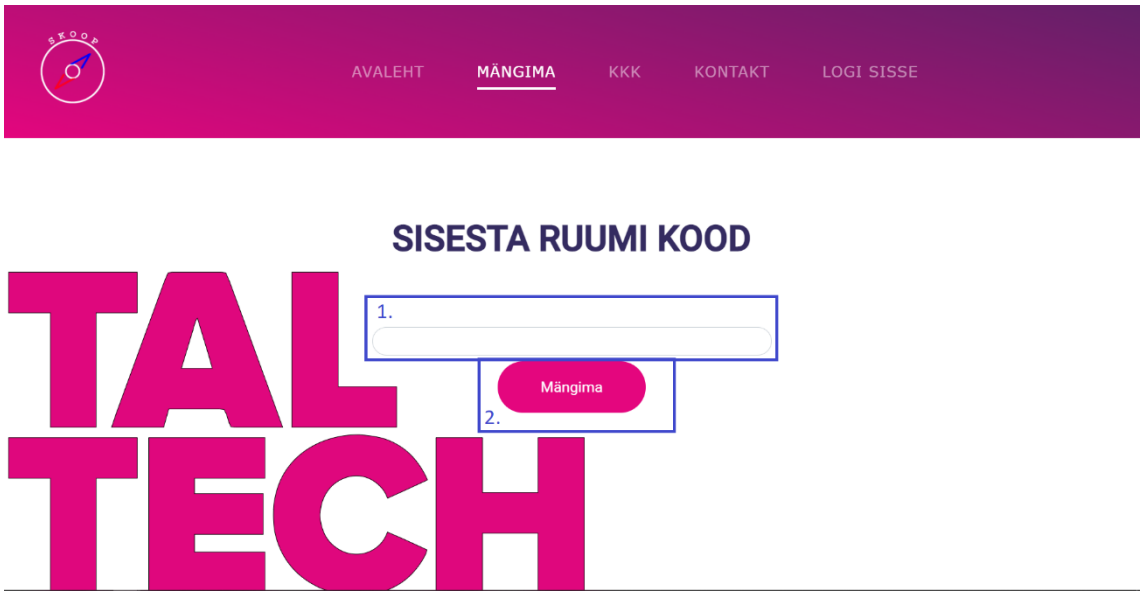


Joonis 4. Sisselogitud kasutaja avalehe vaade

1. Liitu ruumiga – kasutaja suundub ühine ruumiga lehele
2. Loo uus ruum – kust saab luua ruumi mängude jaoks

4.2.7 Mängima

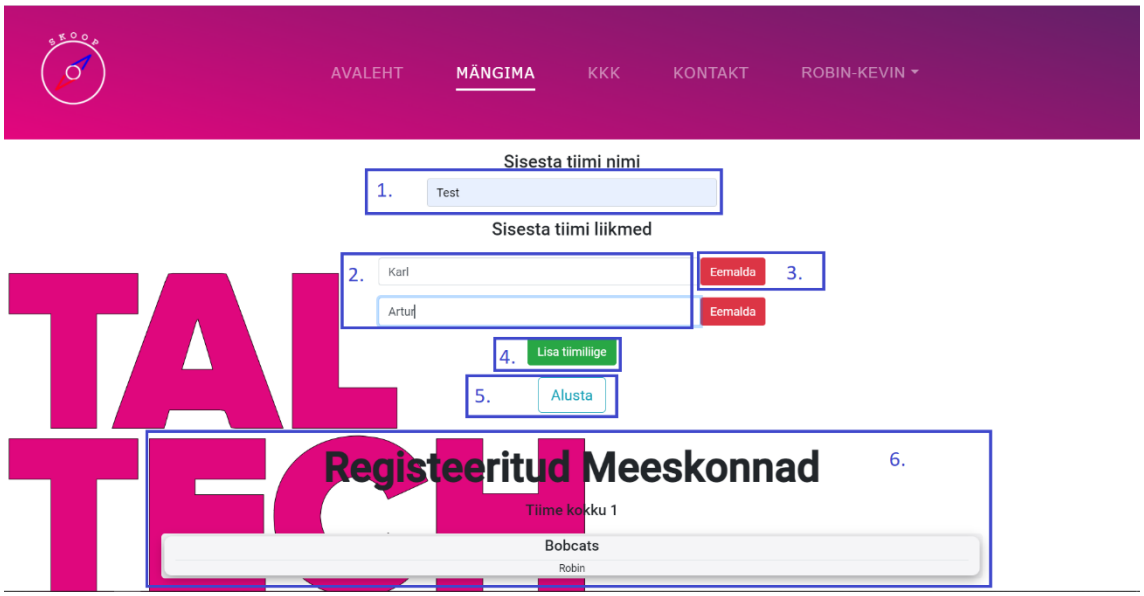
Mängima on vaade, kus kasutaja saab sisestada ühinemis koodi. Joonis 5. on näidatud kuidas on võimalik ühineda mänguga.



Joonis 5. Tavakasutaja mängima vaade

1. Ühinemiskood – kood, mille saab mängu loojalt, et liituda mänguga
2. Mängima – viib kasutaja tiimi registreerimisele

Tiimi registreerimis lehel on võimalik registreerida tiimi nimi koos nende liikmetega. Joonisel 6. on näidatud tiimi registreerimise vaadet.

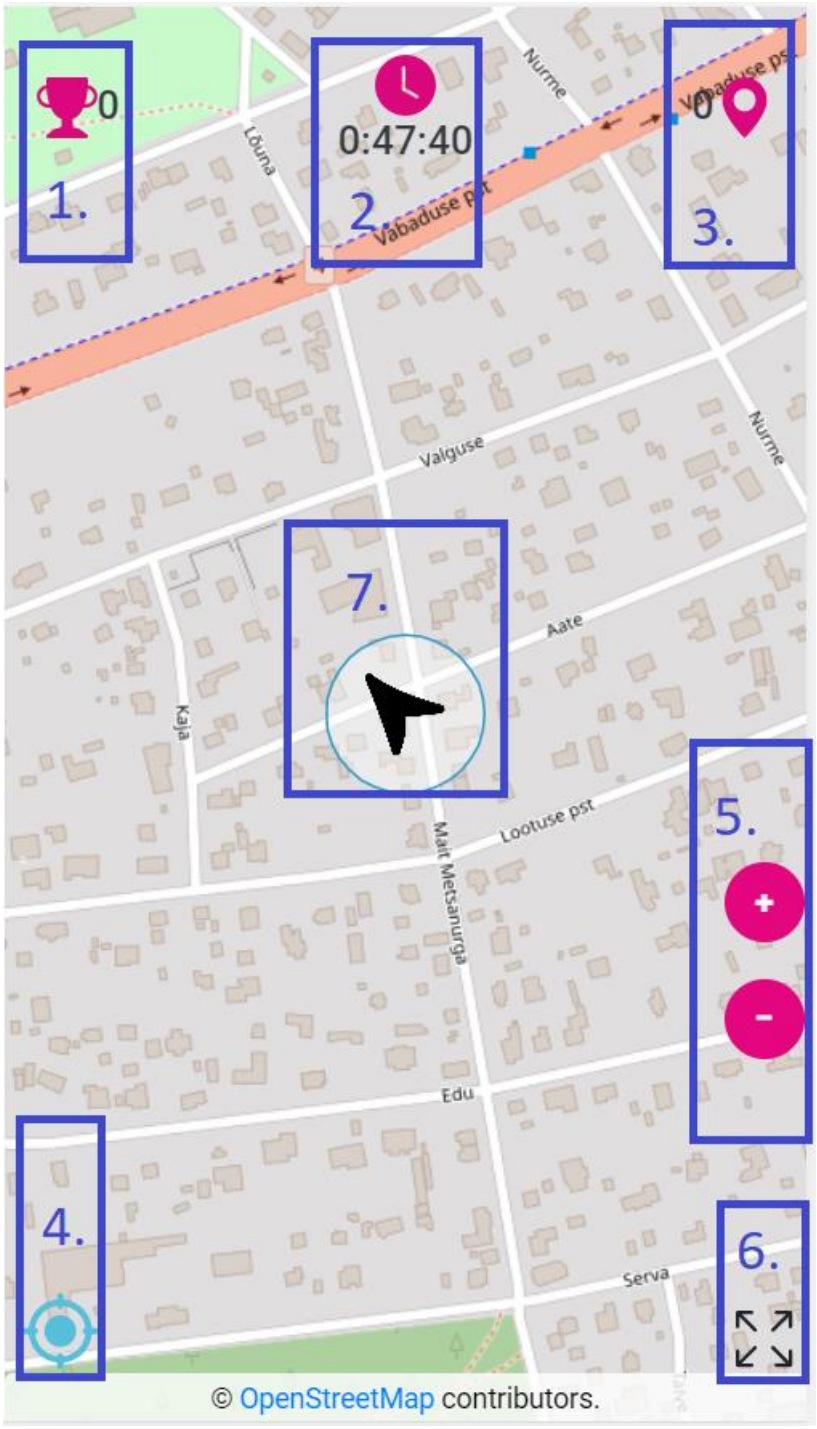


Joonis 6. Tiimi registreerimis lehe vaade

1. Sisesta tiimi nimi – siia sisestatakse oma tiimi nimi, millega hakatakse mängima

2. Siseta tiimi liikmed – siia sisestatakse tiimi liikmete nimed
3. Eemalda – võimalusel on tiimi liiget eemaldada
4. Lisa tiimiliige – soovitakse liikmeid juurde lisada
5. Alusta – suundutakse mängima
6. Registreeritud Meeskonnad – vaade kus on näha hetkel aktiivseid olevaid time, kes mängivad.

Joonis 7. on näidatud, milline näeb välja mängimise mobiilivaade, kui hakatakse mängima mängu



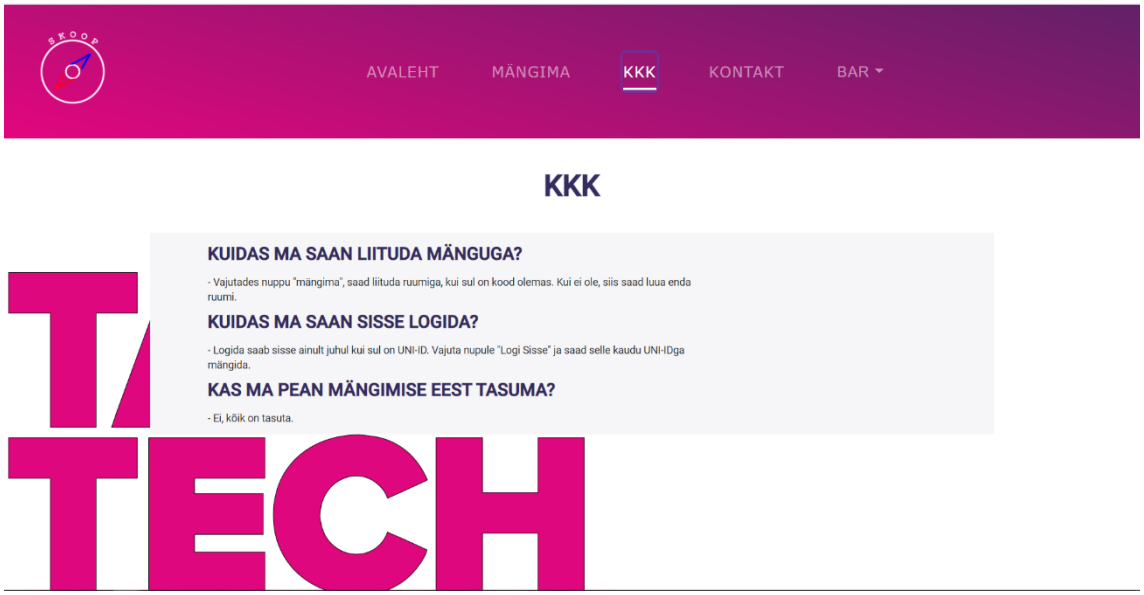
Joonis 7. Mängimise mobiilivaade

1. Hetkene punktiskoor
2. Mänguaeg
3. Palju on alles jäänud tiimil markereid

4. Asukoha lukustus ehk kui kasutaja soovib kaardil vabalt ringi vaadata, siis deaktiveerib lukustuse
5. Sisse ja välja suumine kaardil
6. Seadmes saab minna täisekraanile
7. Kasutaja hetke asukoht koos täpsusega

4.2.8 KKK

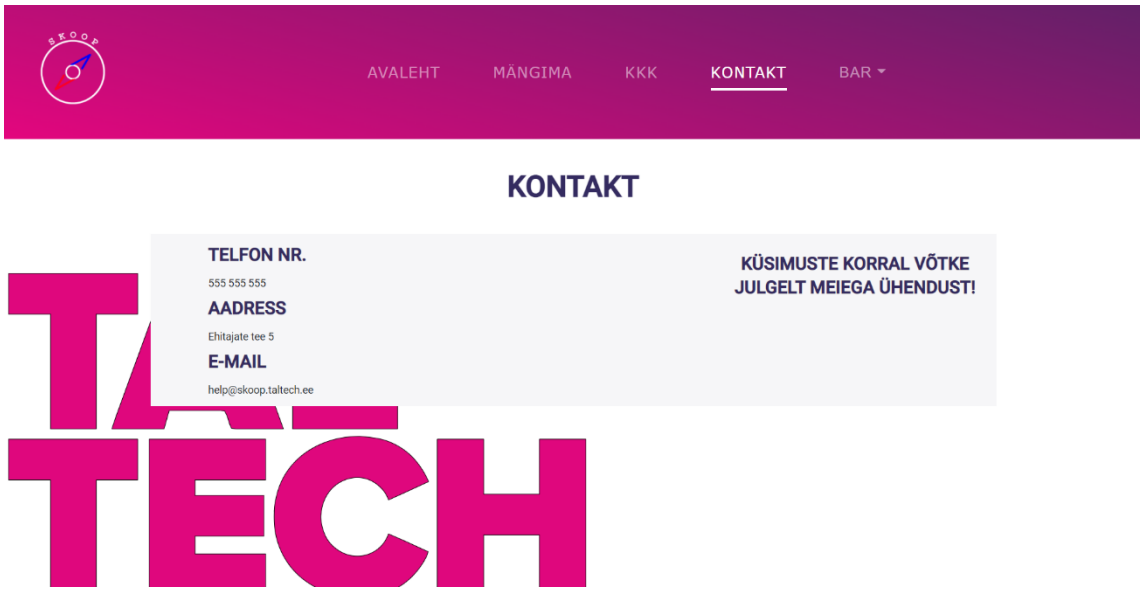
KKK ehk korduma kippuvad küsimused on vaade kus kuvatakse erinevate mängjate või kasutajate poolt tekkinud küsimused. Joonis 8. on näidatud KKK lehe vaadet.



Joonis 8. KKK lehe vaade

4.2.9 Kontakt

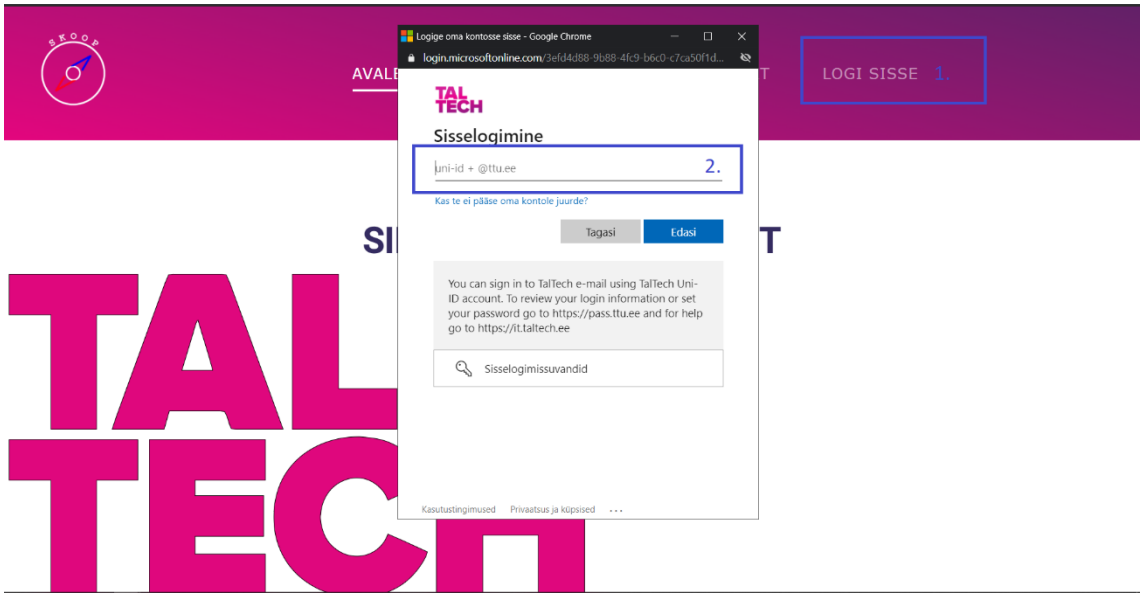
Kontakt vaade on vajalik selleks, et kui soovitakse ühendust võtte tehnilisetoega, kui satutakse mängimisel või loomisel hätta. Joonis 9. on näidatud Kontakt lehe vaadet.



Joonis 9. Kontakt lehe vaade

4.2.10 Sisselogimine

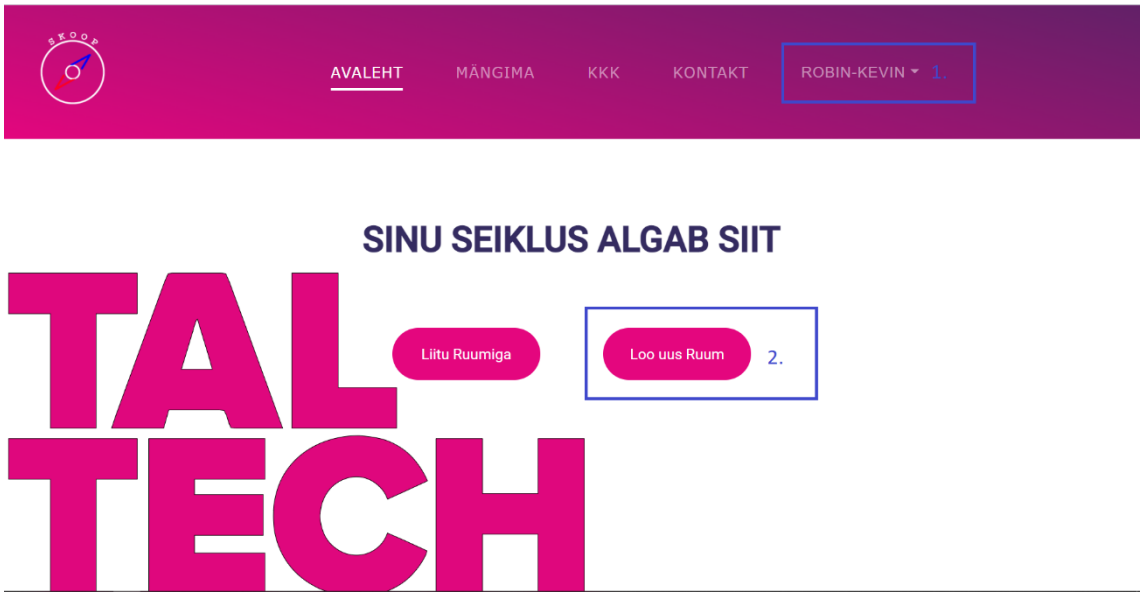
Sisselogimise vaatega saab kasutaja ennast sisselogida kasutades TalTechi poolt antud kasutaja abil. Sisselogimisega on võimalik hakata luua ruume. Joonis 10. on näidatud sisselogimise akent.



Joonis 10. Sisselogimise vaade

1. Logi Sisse – kuvatakse kasutajale sisselogimis aken
2. uni-id + @ttu.ee – kasutaja sisestab oma TalTechi konto

Õnnestunud sisselogimisel kuvatakse kasutaja nimi menüüribale seda on võimalik näha Joonisel 11.

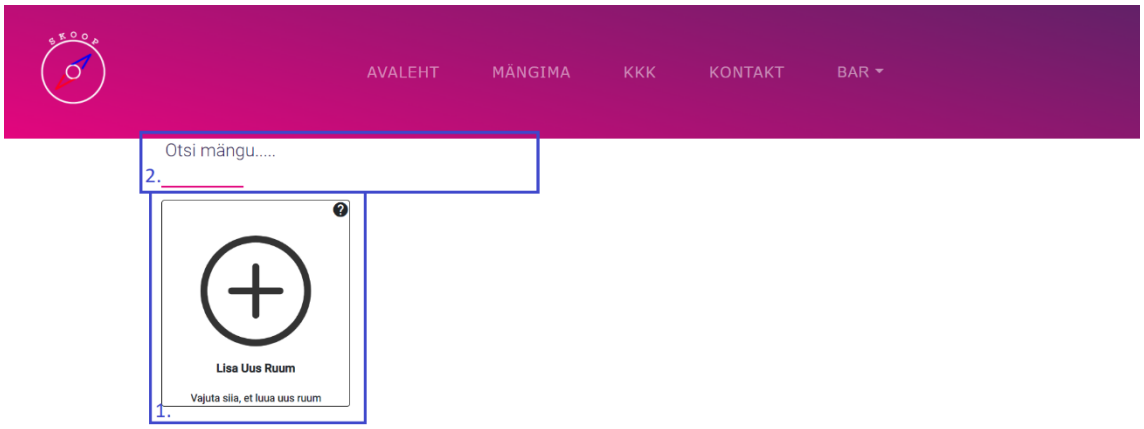


Joonis 11. Õnnestunud sisselogimise vaade

1. Robin-Kevin – kuvatakse kasutaja eesnimi
2. Loo uus Ruum – kasutaja õnnestunud sisselogimisel annab võimaluse luua ruumi

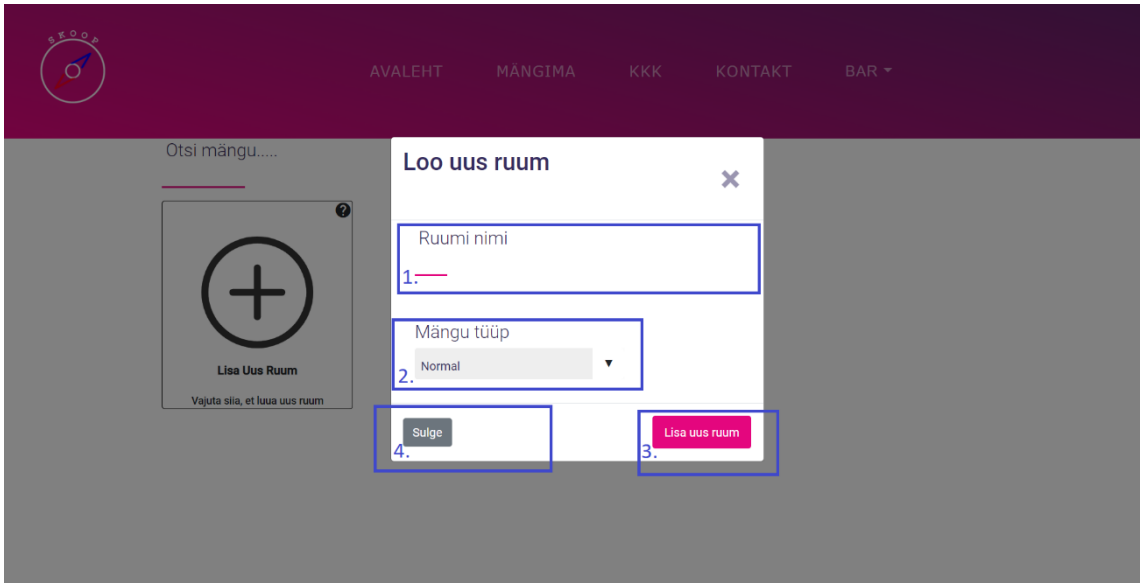
4.2.11 Ruumi loomine

Ruumi loomine on vaade, kus kasutaja saab luua ruumi sedasi, et annab nime ruumile, seejärel valib mängu kestvuse ajaliselt, mängu tüübi ja tiimide maksimaalne arv. Joonis 12. on näidatud ruumi loomise vaadet ja lisaks Joonis 13. on näidatud, mis juhtub kui lisada ruum kus on näha valikuid.



Joonis 12. Ruumi loomise lehe vaade

1. Lisa Uus Ruum – luuakse ruum kasutaja poolt
2. Otsi mängu – kui kasutajal on mitu ruumi, siis vajadusel saab otsida



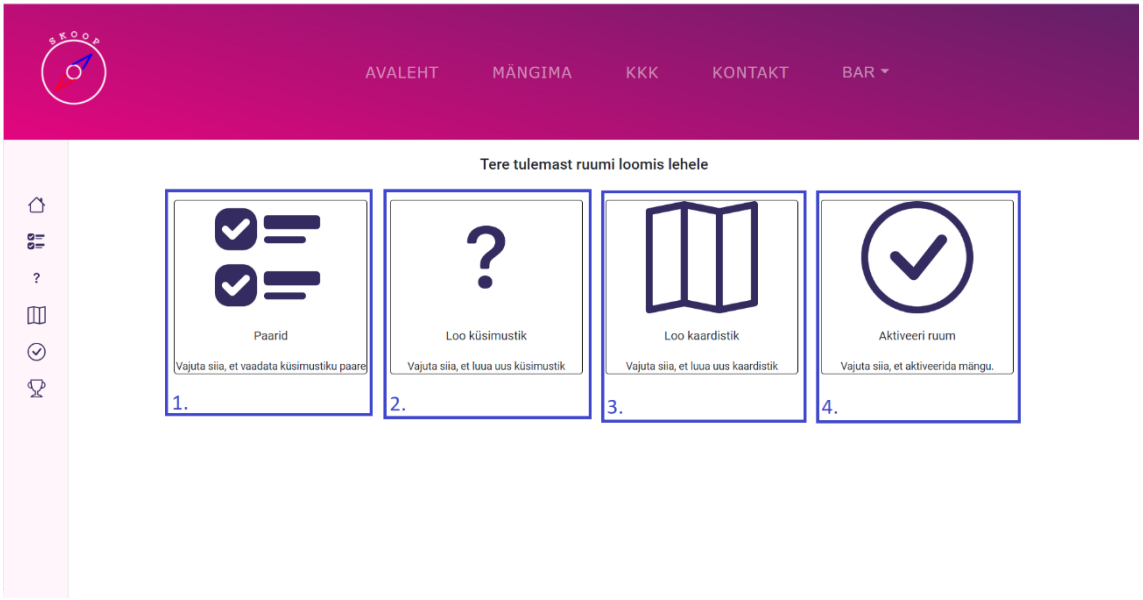
Joonis 13. Ruumi loomise vaade

1. Ruumi nimi – kasutaja annab ruumile nime
2. Mängu tüüp – kasutaja saab valida kolme tüübi vahel, et millist ruumi luua
3. Lisa uus ruum – kui sätted paigas lisatakse uus ruum

4. Sulge – kasutaja ei soovi luu uut ruumi, siis sulgub ruumi loomise vaate *pop-up*

4.2.12 Ruumi muutmine

Ruumi muutmine on vaade, kus kasutaja saab navigeerida enda loodud ruumi sisu ehk mis kaardistik on seal, küsimustiku paarid, aktiveeritud ruumid ja küsimustikud. Joonis 14. on näidatud ruumi loomise vaadet.

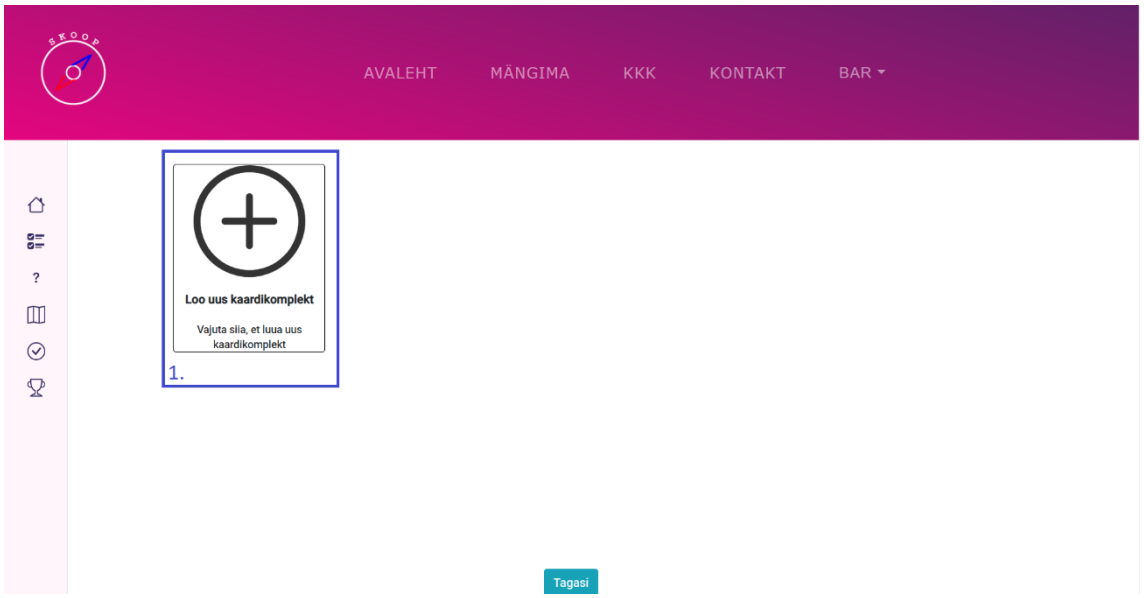


Joonis 14. Ruumi muutmise lehe vaade

1. Paarid – kasutaja saab luua uusi paare ja näha loodud paare
2. Loo küsimustik – kasutaja saab luua küsimustiku
3. Loo kaardistik – kasutaja saab luua kaardistikku
4. Aktiveeri ruum – kasutaja saab aktiveerida ruumi

4.2.13 Kaardistiku loomine

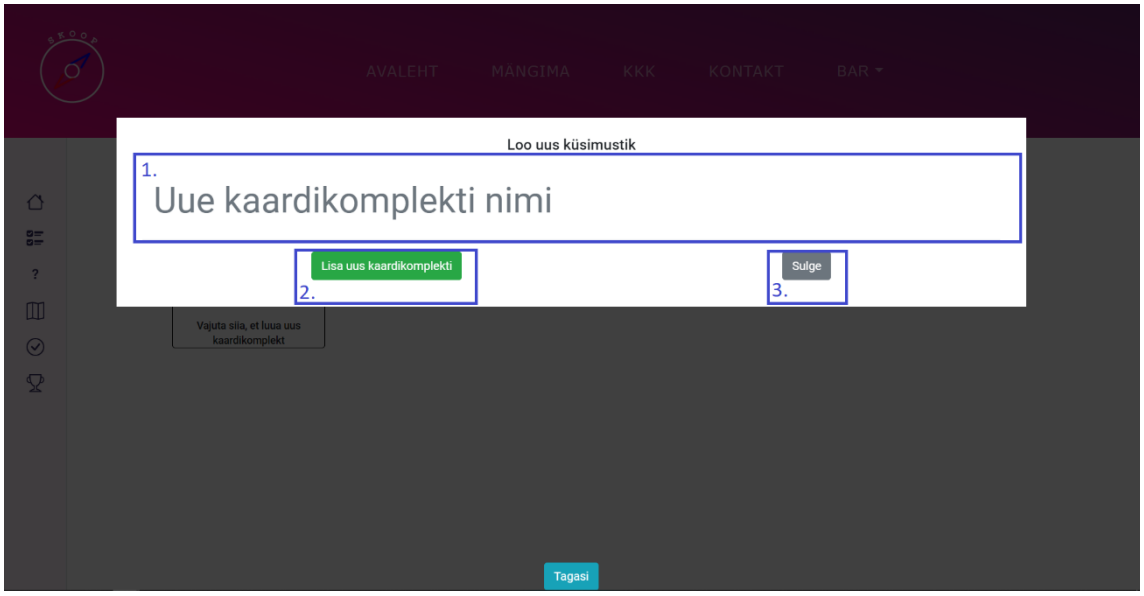
Kaardistiku loomine on vaade, kus kasutaja lisab punktid kaardile ja hiljem seob need küsimutega. Joonis 15. on näidatud kaardistiku loomise vaadet.



Joonis 15. Kaardistiku loomise vaade

1. Loo uus kaardikomplekt – kasutaja saab luua kaardikomplekti

Joonis 16. on näidatud kuidas luua uut kaardikomplekti.

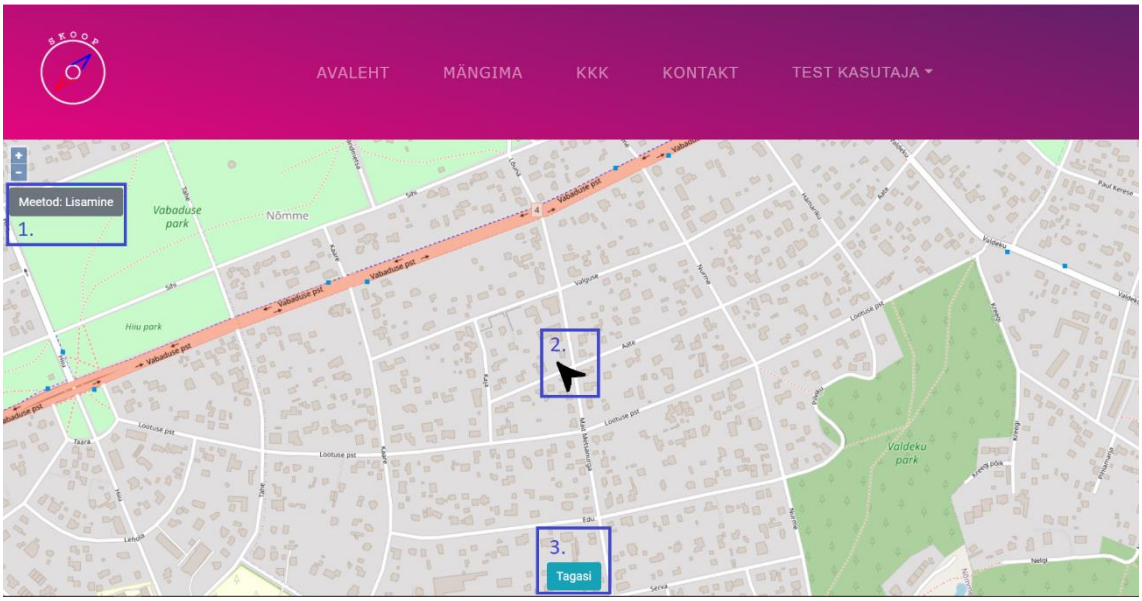


Joonis 16. Loo uus kaardikomplekt vaade

1. Uue kaardikomplekti nimi – kasutaja annab nime loodavale kaardikomplektile
2. Lisa uus kaardikomplekt – kasutaja lisab uue kaardikomplekti

3. Sulge – kasautaja sulgeb uue kaardikomplekti loomise

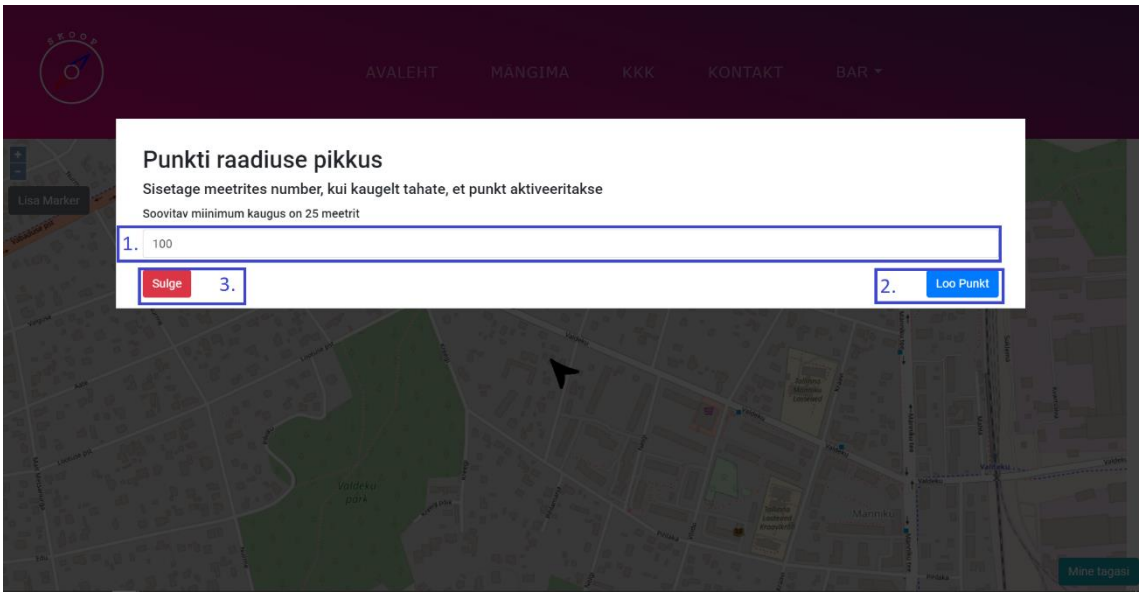
Joonis 17. on näidatud kuidas lisada loodud kaardikomplektile markereid, millele on hiljem võimalik siduda küsimused.



Joonis 17. Kaardi muutmise vaade

1. Meetod: Lisamine – kasutaja saab lisada punkti vabalt valitud kohale kaardile
2. Kasutaja hetkene asukoht
3. Tagasi – kasutaja suundub tagasi kaardikomplekti vaatesse

Joonis 18. on näidatud, kui lisada kaardile marker. Saab lisada raadiuse antud markerile.

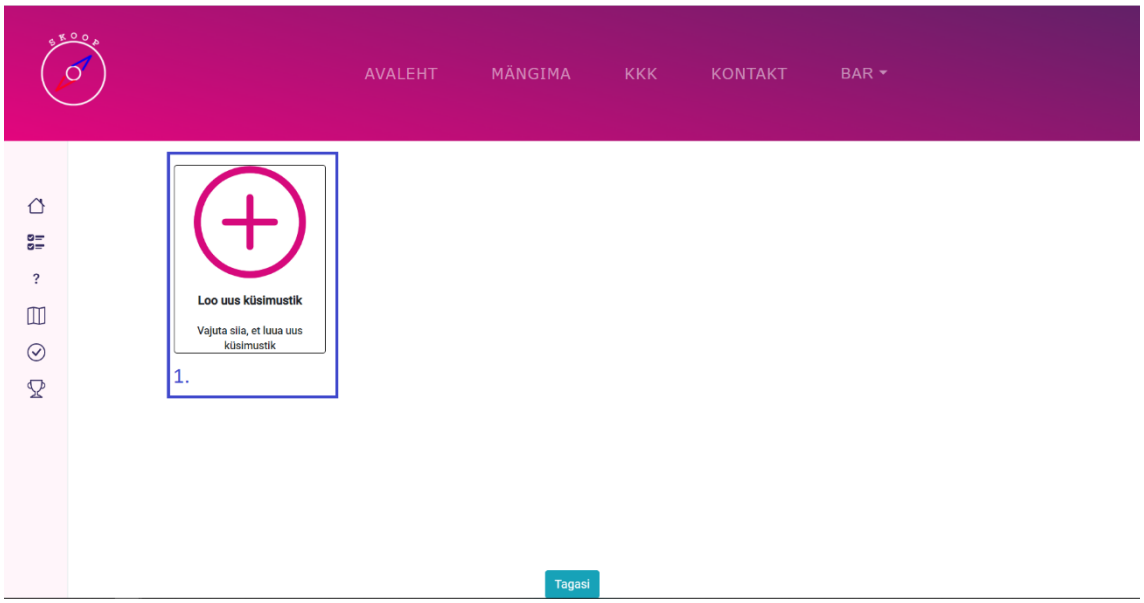


Joonis 18. Punkti lisamise vaade

1. Punkti raadius – kasutaja lisab punktile raadiuse, kui lähedalt on võimalik punkt kätte saada
2. Loo Punkt – lisab punkti kaardile
3. Sulge – ei lisa punkti kaardile ja sulgeb

4.2.14 Küsimustiku loomine

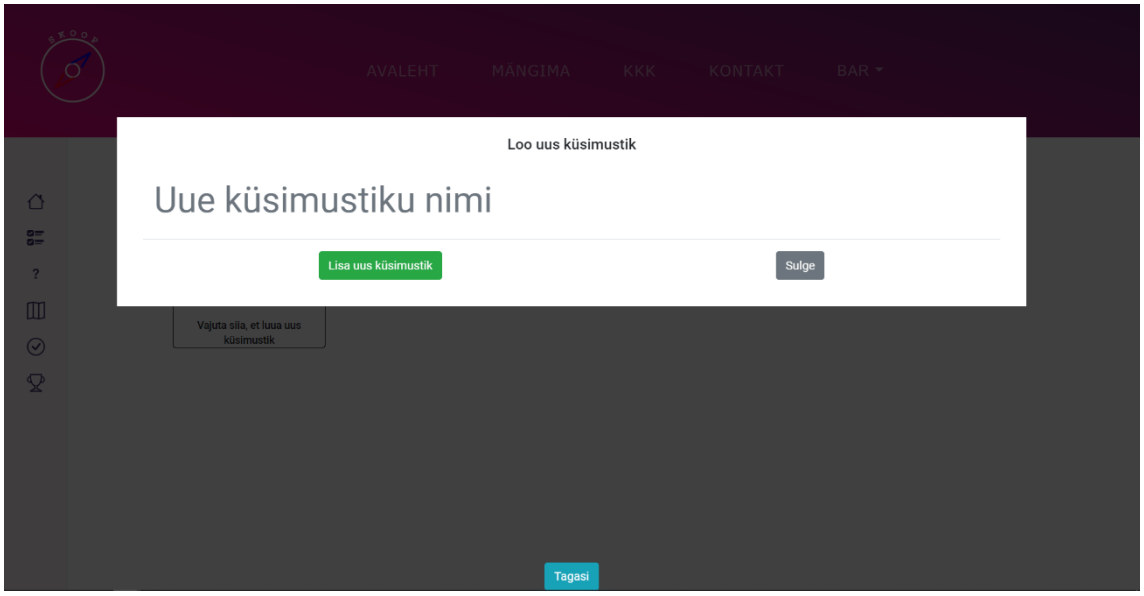
Küsimustiku loomine on vaade, kus kasutaja näeb ja saab lisada küsimustikke, mis omakorda sisaldavad küsimusi. Joonis 19. on näidatud küsimustiku loomise vaadet.



Joonis 19. Küsimustiku loomise vaade

1. Lisa uus küsimustik – kasutaja lisab uue küsimustiku

Joonis 20. on näidatud uue küsimustiku loomise vaadet, mis kuvatakse kasutajale.

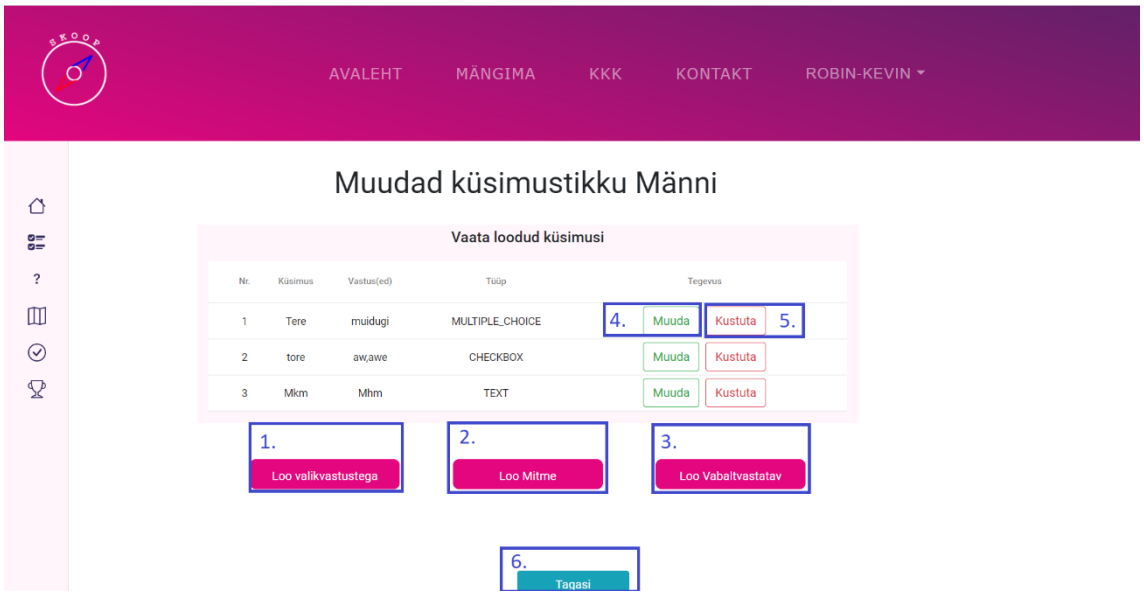


Joonis 20. Lisa uus küsimustik vaade

1. Uue küsimustiku nimi – kasutaja annab uuele küsimustikule nime
2. Lisa uus küsimustik – kasutaja lisab uue küsimustiku
3. Sulge – kasutaja sulgeb ja ei lisa küsimustikku

4.2.15 Küsimuste loomine

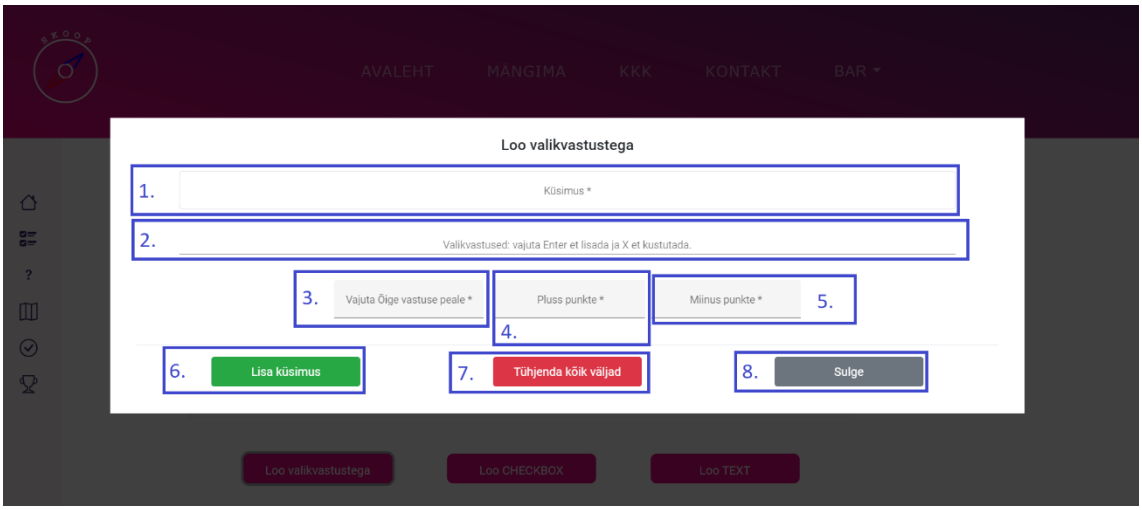
Küsimuste loomine on vaade, kus kasutaja saab luua küsimuse või kustutada. Joonis 21. on näidatud küsimuste loomise vaadet.



Joonis 21. Küsimuste loomise lehe vaade

1. Loo valikvastustega – kasutaja saab luua valikvastustega küsimuse
2. Loo Mitme - kasutaja saab luua mitme valikvastustega küsimuse
3. Loo Vabaltvastav – kasutaja saab luua tekst küsimuse
4. Muuda – kasutaja saab muuta valitud küsimust
5. Kustuta – kasutaja saab kustutada valitud küsimust
6. Tagasi - kasutaja suundub tagasi küsimustikke vaatesse

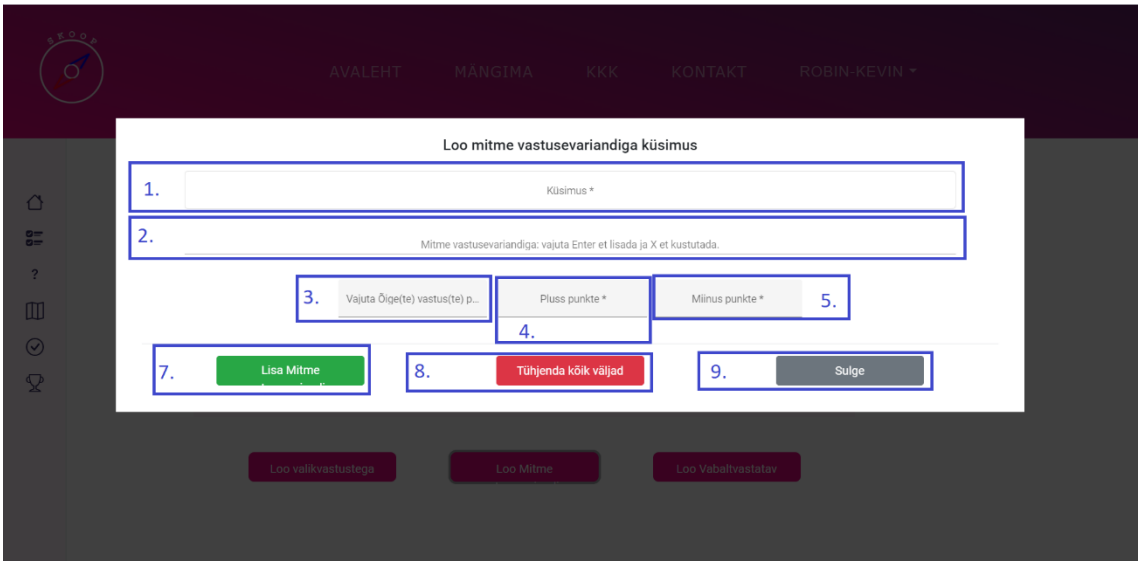
Joonis. 22 on näidatud kuidas luuaks valikvastusega küsimust.



Joonis 22. Loo valikvastustega vaade

1. Küsimus – kasutaja lisab küsimuse
2. Valikvastused: vajuta Enter et lisada ja X et kustutada. – kasutaja lisab valikvastused
3. Vajuta Õige vastuse peale – kasutaja vajutab õigele vastusele
4. Pluss punkte – kasutaja lisab pluss punkte õigest vastatud küsimuse eest
5. Miinus punkte – kasutaja lisab miinus punkte valesti vastatud küsimuse eest
6. Lisa küsimus – kasutaja lisab küsimuse
7. Tühjenda kõik väljad – kasutaja tühjendab kõik väljad, et alustada otsast peale küsimuse loomisega
8. Sulge – kasutaja sulgeb kui ei soovi lisada küsimust

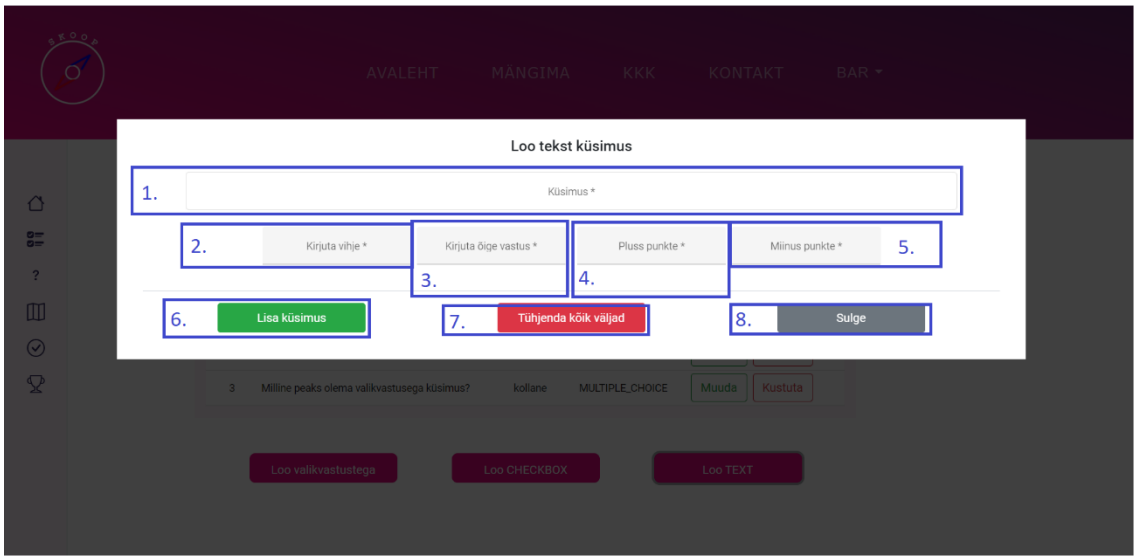
Joonis 23. on näidatud kuidas saab luua mitme vastusevariandiga küsimust.



Joonis 23. Loo mitme vastusevariandiga vaade

1. Küsimus – kasutaja lisab küsimuse
2. Mitme vastusevariandiga: vajuta Enter et lisada ja X et kustutada. - kasutaja lisab vastuse variandi
3. Vajuta Õige(te) vastus(te) peale – kasutaja vajutab õigetele vastusetele
4. Pluss punkte – kasutaja lisab pluss punkte õigest vastatud küsimuse eest
5. Miinus punkte – kasutaja lisab miinus punkte valesti vastatud küsimuse eest
6. Lisa küsimus – kasutaja lisab küsimuse
7. Tühjenda kõik väljad – kasutaja tühjendab kõik väljad, et alustada otsast peale küsimuse loomisega
8. Sulge – kasutaja sulgeb kui ei soovi lisada küsimust

Joonis 24. on näidatud kuidas luua vabaltvastatav küsimust.

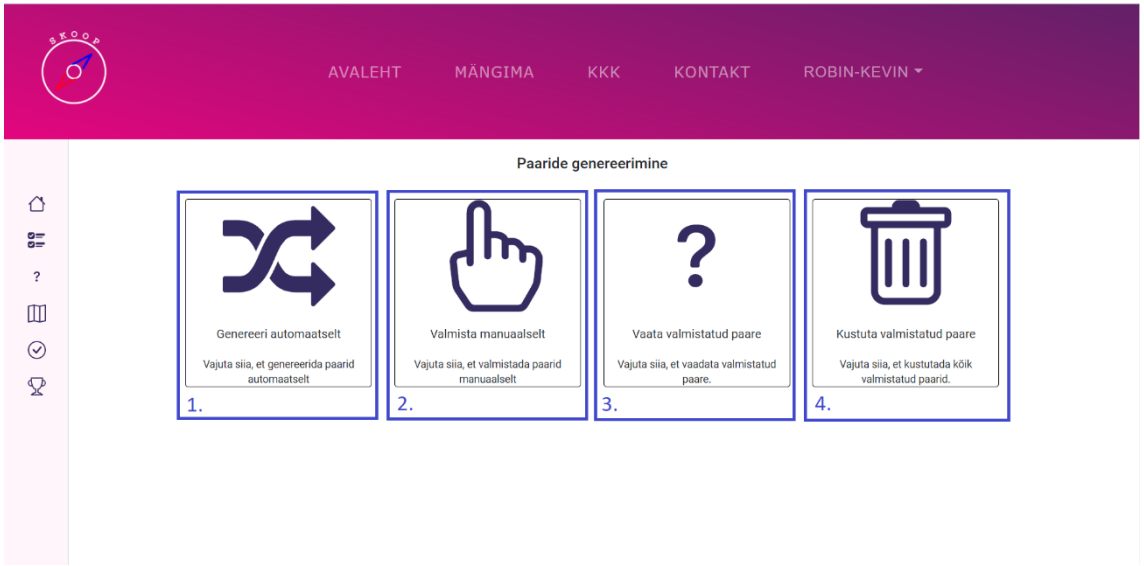


Joonis 24. Loo vabaltvastatav küsimus vaade

1. Küsimus – kasutaja lisab küsimuse
2. Kirjuta vihje - kasutaja lisab vihje küsimuse kohta
3. Kirjuta õige vastus – kasutaja kirjutab õige vastuse
4. Pluss punkte – kasutaja lisab pluss punkte õigest vastatud küsimuse eest
5. Miinus punkte – kasutaja lisab miinus punkte valesti vastatud küsimuse eest
6. Lisa küsimus – kasutaja lisab küsimuse
7. Tühjenda kõik väljad – kasutaja tühjendab kõik väljad, et alustada otsast peale küsimuse loomisega
8. Sulge – kasutaja sulgeb kui ei soovi lisada küsimust

4.2.16 Paaride genereerimine

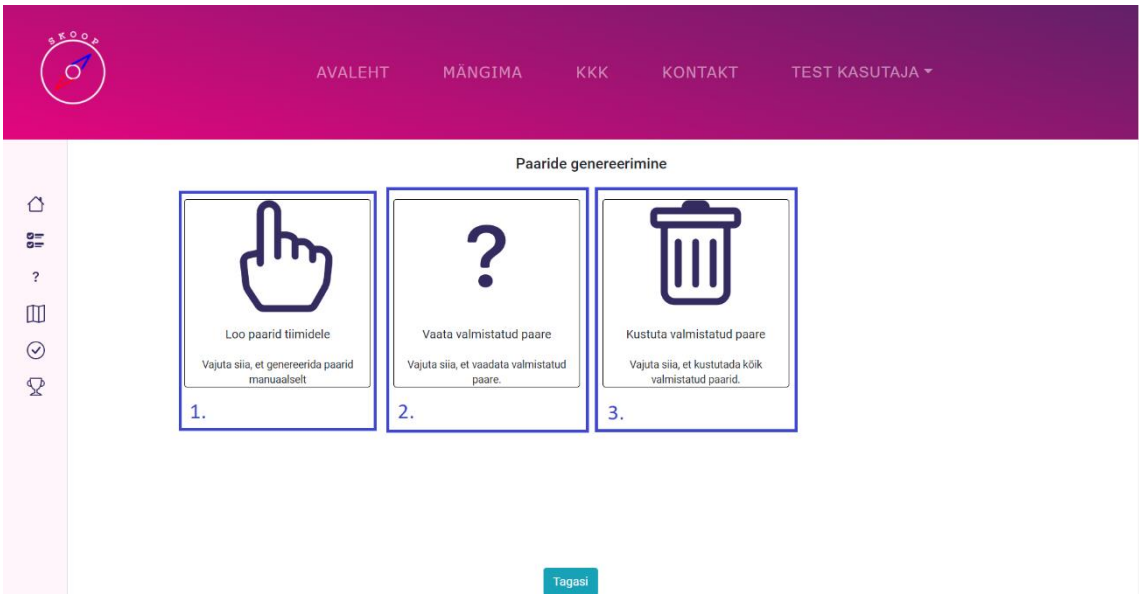
Paaride genereerimine on vaade, kus kasutaja seob kaardil asuvad punctid küsimutega ära kahel variandil, kas automaatselt või käsitsi. Joonis 25. on näidatud paaride genereerimise vaadet, kui luuakse ruumi *normal* või *rush* tüübil.



Joonis 25. Paaride genereerimise lehe vaade *normal* ja *rush* tüüpidel

1. Genereeri automaatselt – kasutaja saab automaatselt genereerida punktidele küsimused, mis on loodud
2. Valmista manuaalselt - kasutaja saab manuaalselt lisada punktidele küsimused, mis on loodud
3. Vaata valmistatud paare – kasutaja saab vaadata hetkel valmistatud paare
4. Kustuta valmistatud paare – kasutaja saab kustutada kõik paarid mis on punktidega seotud

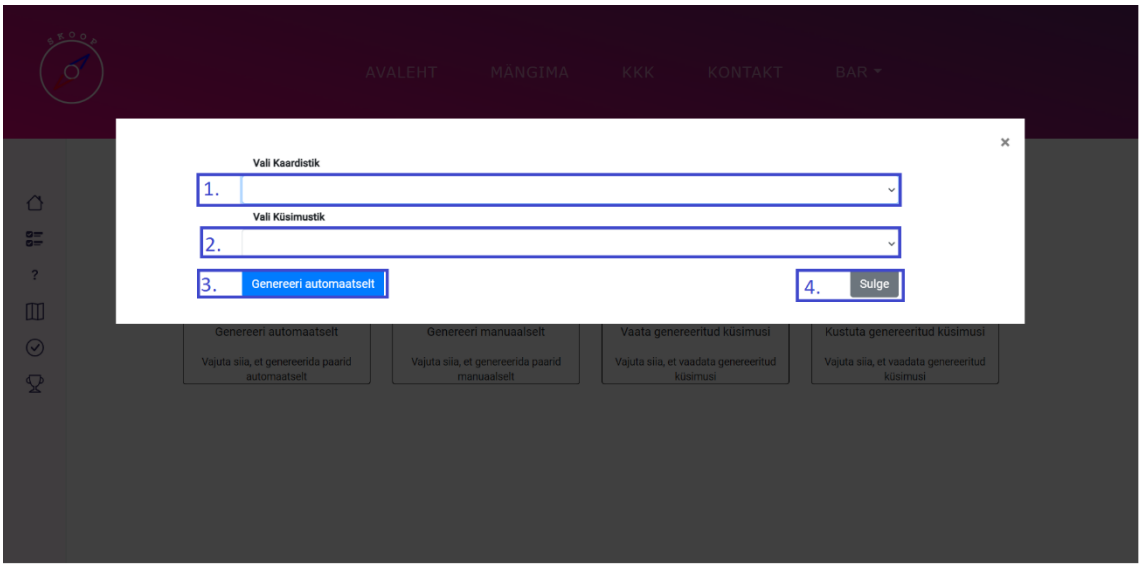
Joonis 26. on näidatud paaride genereerimis vaadet, kui luuakse *preassigned* ruum.



Joonis 26. Paaride genereerimise vaade *preassigned* tüübil

1. Loo paarid tiimidele – kasutaja saab valmistada paarid tiimidele eraldi
2. Vaata valmistatud paare – kasutaja saab vaadata hetkel genereeritud küsimusi
3. Kustuta valmistatud küsimusi – kasutaja saab kustutada kõik küsimused mis on punktidega seotud

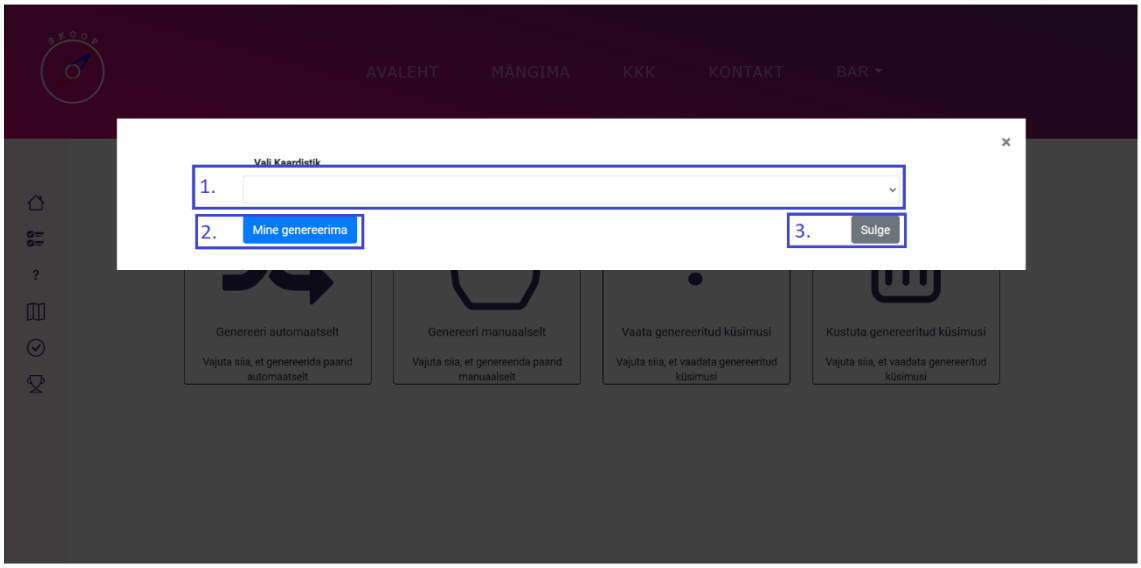
Joonis 27. on näidatud kuidas saab genereerida paarid automaatselt.



Joonis 27. Automaatse genereerimise vaade

1. Vali kaardistik – kasutaja valib kaardistiku kuhu soovib genreerida küsimustikud
2. Vali küsimustik – kasutaja valib küsimustiku
3. Genereeri automaatselt – kasutaja genereerin automaatselt küsimustiku
4. Sulge – kasutaja sulgeb automaatse genereerimise ja ei genereerita küsimustikud

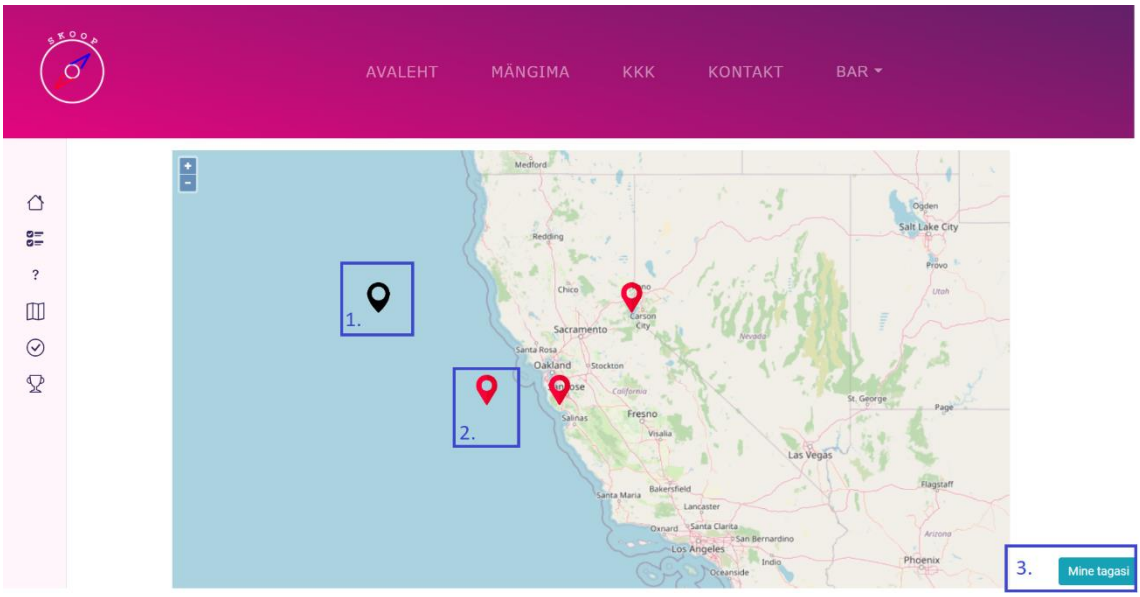
Joonis 28. on näidatud kuidas manuaalselt lisatakse markerid kaardile, aga ennem tuleb valida kaardikomplekt.



Joonis 28. Genereeri manuaalselt vaade

1. Vali kaardistik – kasutaja valib kaardistiku kuhu soovib genereerida küsimustikud
2. Mine genereerima – kasutaja asub manuaalselt genereerima kaardi peale
3. Sulge – kasutaja sulgeb ja ei soovi manuaalselt genereerida

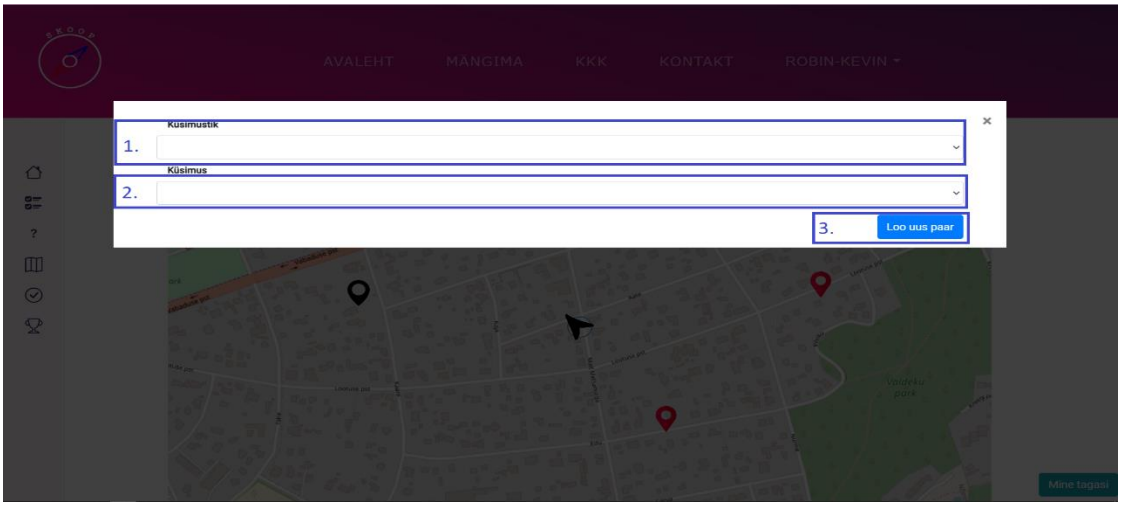
Joonis 29. on näidatud, kuidas lisatakse markeritele küsimused.



Joonis 29. Manuaalse genereerimine kaardil vaade

1. Punkt millel puudub küsimus
2. Punkt mille on küsimus olemas
3. Mine tagasi – kasutaja suundub tagasi paaride vaatesse

Joonis 30. on näidatud akent, kus kuvatakse küsimustik ja sealt saab valida oma küsimuse, mille saab siduda markeriga.

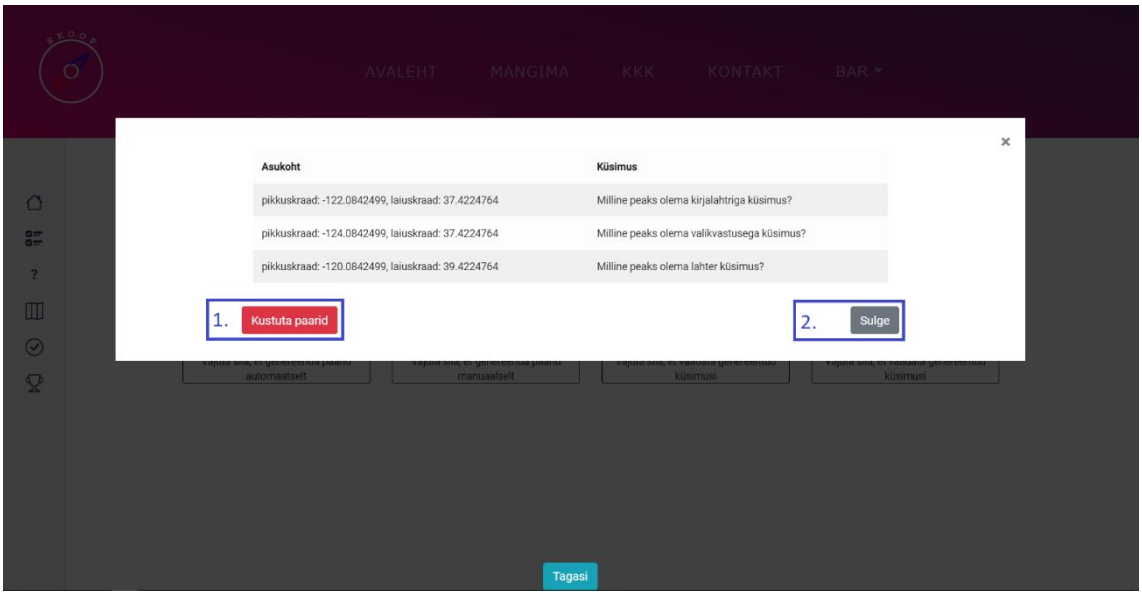


Joonis 30. Lisa punktile küsimus vaade

1. Küsimustik – kasutaja valib küsimustiku

2. Küsimus – kasutaja valib küsimuse, mida hakatakse kuvama antus punktis
3. Loo uus paar – kasutaja loob seose

Joonis 31. on näidatud valmistatud paare.

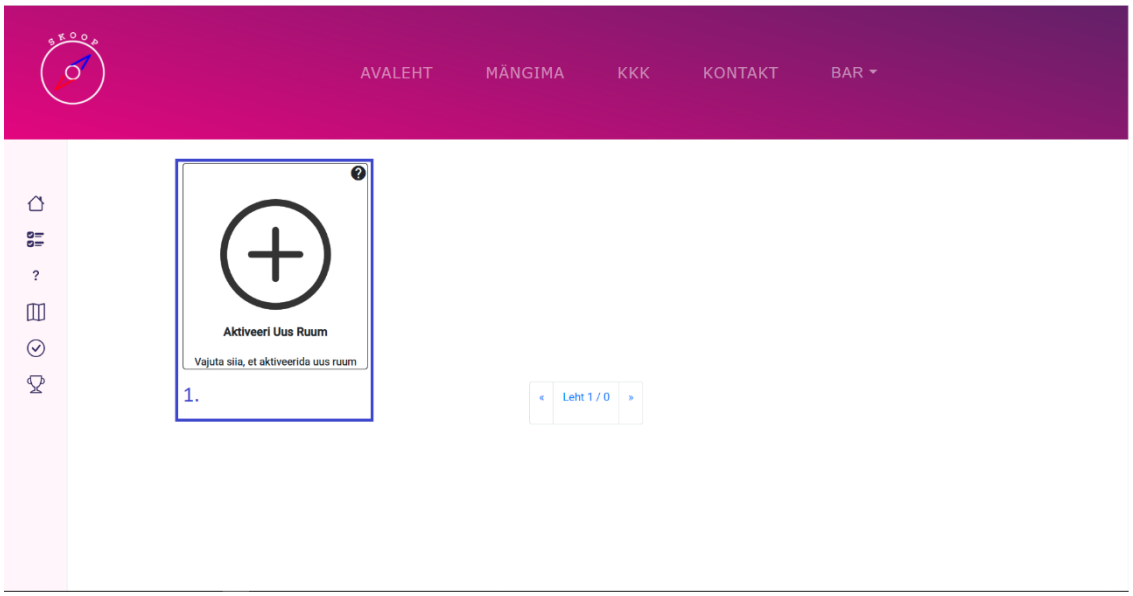


Joonis 31. Vaata valmistatud paare vaade

1. Kustuta paarid – kasutaja saab kõik seotud punktidega küsimused kustutada
2. Sulge – kasutaja ei soovi kustutada vaid vaadelda loodud küsimusi

4.2.17 Aktiveerimine

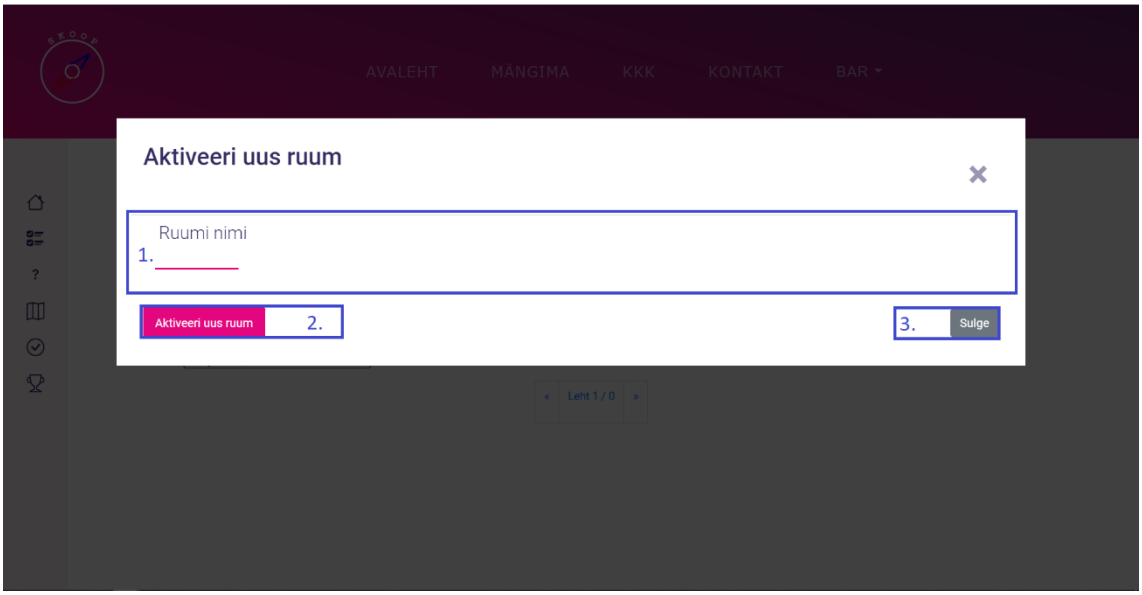
Aktiveerimine on vaade, kus kasutaja näeb hetkel aktiveeritud ruume ja saab uue ruumi aktiveerida mängimiseks. Joonis 32. on näidatud aktiveerimise vaadet.



Joonis 32. Aktiveerimis lehe vaade

1. Aktiveeri Uus Ruum – kasutaja aktiveerib uue ruumi, millega saab pärast mängijad ühineda

Joonis 33. on näidatud ruumi aktiveerimise akent, kuhu tuleb lisada aktiveeritavale ruumile nimetus.



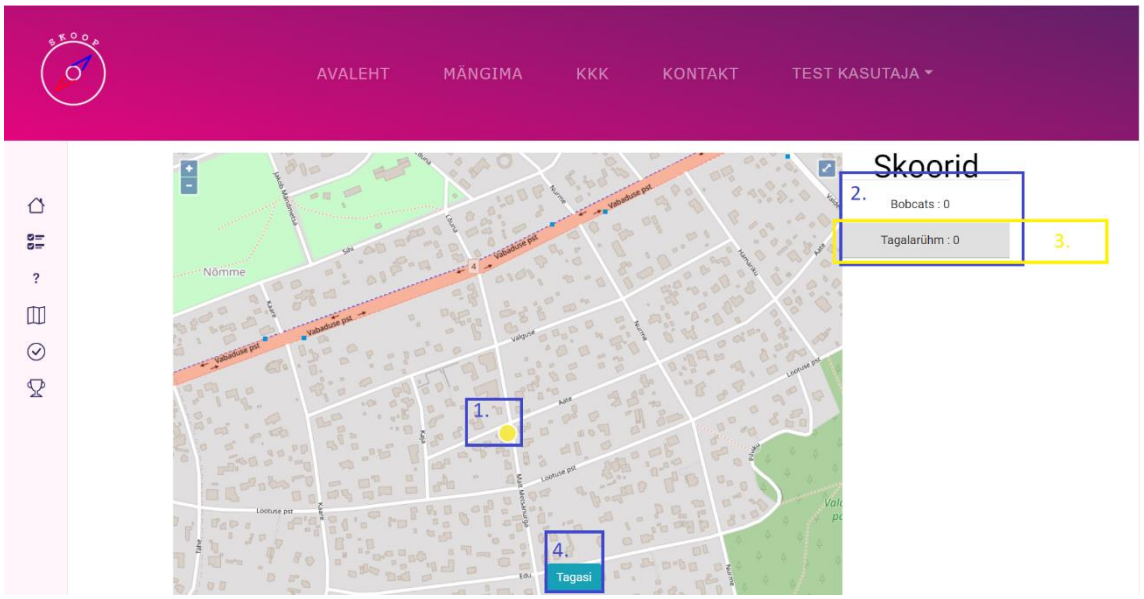
Joonis 33. Aktiveeri uus ruum vaade

1. Ruumi nimi – kasutaja annab aktiveeritavale ruumile nime

2. Aktiveeri Uus Ruum – kasutaja aktiveerib uue ruumi
3. Sulge – kasutaja ei soovi uut ruumi aktiveerida

4.2.18 GameMaster vaade

GameMaster vaade on, kus mängu/ruumi looja näeb hetkel toimuvat mängu reaalajas. Joonis 34. on näidatud GameMaster vaadet.



Joonis 34. GameMaster vaade

1. Tiim „Tagalarühm“ hetkene asukoht kaardil
2. Skoorid – hetkel aktiivsed olevad tiimid, kes mängivad ja nende hetke punktiskoori
3. Tagalarühm: 0 – peale klikades kuvatakse kaardil antud tiimi asukoht

4.3 Back-end

Tagataustal olev kiht ehk *back-end* on nii-öelda projekti selgroog. Sinna lähevad kõik andmed sisse, kus nendega tehakse erinevaid protsesse. Neid töödeldakse, vajadusel muudetakse, uuendatakse, kustutatakse ja/või salvestatakse.

Back-end on peamiselt üles-ehitatud Java, Spring Booti ning PostgreSQL-iga. Veel on erinevad lisad nagu Docker, Gradle, Flyway ning OAuth. Andmetöötluskihte on mitu tükki: andmebaas ise, teenuste kiht ning kontrollerite kiht. Andmebaasi ning teenuste vahelist suhtlust korraldavad JPA repositooriumi klassid, mis on Spring Booti sisse ehitatud. Kontrollerid vahendavad informatsiooni välismaailmaga REST API abil.

4.3.1 Docker

Docker oli algselt ühe PostgreSQL andmebaasiga nimega gamesdb aine *Tarkvara meeskonnaprojekti* jooksul juba üles seadistatud. Selle projekti jooksul pandi püsti teine andmebaas nimega test testide jooksutamise eesmärgil, et reaalselt kasutuses olevasse andmebaasi satuks võimalikult vähe mitte-soovituid andmeid, mis on testimise käigus loodud.

Selle andmebaasi loomine saavutati SQL faili tegemisega, mis teeb uue täiesti tühja andmebaasi. Joonisel 35. on test andmebaasi loomine.

```
DROP DATABASE IF EXISTS test;
CREATE DATABASE test;
```

Joonis 35. Testimise andmebaasi loomise SQL kood

See SQL fail pannakse *docker-compose up* käskluse käivitamisel Dockeri konteineri-sisesesse kausta docker-entrypoint-initdb.d ning käivitatakse. Tulemusena tekib uus andmebaas nimega test. Joonisel 36. on docker-compose fail.

```
version: "3.7"

services:
  users-service-db:
    image: postgres:12-alpine
    volumes:
    - ./docker/testing-db.sql:/docker-entrypoint-
initdb.d/testing-db.sql
    ports:
      - 5632:5432
    environment:
      - POSTGRES_DB=gamesdb
      - POSTGRES_USER=root
      - POSTGRES_PASSWORD=root
```

Joonis 36. Docker-compose.yml fail Dockeri käivitamiseks

4.3.2 Flyway

Flyway peamine kasutus selles projektis on andmebaasi migreerimine ning uuendamine võimalikult lihtsalt. Selleks kasutatakse peamiselt kolme käsklust (Clean, Migrate, Info), kuid kasutusel olevaid käsklusi on kokku 7 [8].

Teise andmebaasi loomine vajas Flyway lisa seadistamist. Algselt tehti üks flyway.conf fail, mis näitas Flywayle käskluste jooksutamiseks vajalikku informatsiooni. Seda faili kasutati .gitlab-ci.yml sisemiste migreerimise, puhastamise ning info näitamise käskluste korral. Failid jagati peale test andmebaasi lisamist kaheks uueks failiks prod-flyway.conf (Joonis 37.) ning test-flyway.conf, kus esimene näitab gamesdb andmebaasi peale ning teine test andmebaasi peale. Kuna andmebaasi sisemine struktuur ning algandmed on identsed, siis on neil identsed migratsiooni SQL failid.

```
flyway.url: jdbc:postgresql://localhost:5632/gamesdb
flyway.schemas: public
flyway.user: root
flyway.password: root
flyway.locations =
filesystem:src/main/resources/db/migration
```

Joonis 37. Prod-flyway.conf fail Flyway käskluste jooksutamiseks

4.3.3 GitLab CI/CD

Ühe andmebaasi lisamine tähendas seda, et GitLab *runneri* juhendit failis .gitlab-ci.yml oli vaja täiendada. Juhendis on kokku 4 sammu, mida *runner* peab läbima, et toimuks õnnestunud projekti evitamine. Nendeks sammudeks on *deploy-database, build, test* ning lõpuks *deploy*.

Esimene samm *deploy-database* püstitab Dockeri ning kasutades flyway.conf faili teeb ära andmebaasi puhastamise ning migreerimise. Selle sammuga algseadistatakse gamesdb andmebaas, mis on mõeldud päris kasutuse jaoks. Oli vaja muuta, et runner vaataks eelneva ühe faili asemel kindlat prod-flyway.conf faili, kus oli vajalik informatsioon gamesdb kohta.

Teine samm *build* kasutab gradle käsklust assemble ning paigutab ehitatud JAR faili kindlasse mainitud kohta.

Kolmas samm on *test*. Tuli kindlaks määrata, et andmebaasi puhastamine ning migreerimine toimuks test andmebaasiga, sest vastasel juhul võivad testimise käigus sisse jääda mitte soovitud andmed. Peale seda jooksutatakse Gradle käsklus check ning käivitatakse olemasolevad testid.

Neljas samm on *deploy*. Failid kopeeritakse serveris ümber õigesse kausta ning Linuxi teenus taaskäivitatakse.

Kõik sammud peale 3. sammu test tehakse ainult *master* ning *development* harus. Teistes harudes toimub lihtsalt testimine, et mitte-soovitud muudatused ei jõuaks reaalsesse keskkonda. Kui andmebaasi olemasolevates SQL-failides muudatusi ei tehta, siis peaks andmebaasi puhastamine olema välja lülitatud (v.a test andmebaasi puhul, mis tuleks enne igat testimist puhtaks teha.).

4.3.4 Gradle'i seadistamine

Teise andmebaasi tõttu ei töötanud Gradle'i Flyway käsklused enam korralikult. Build.gradle failis pidi läbi viima muudatused, mis täpsustavad käsu tüüpi ning valitud andmebaasi, ehk sisuliselt pidi dubleerima (Joonis 38.) Flyway Clean, Migrate ning Info käsklused mõlema andmebaasi kohta.

```
task flywayMigrateRealDb(type: FlywayMigrateTask) {
   url = 'jdbc:postgresql://localhost:5632/gamesdb'
   user = 'root'
   password = 'root'
   locations = ['filesystem:src/main/resources/db/migration']
}

task flywayInfoRealDb(type: FlywayInfoTask) {
   url = 'jdbc:postgresql://localhost:5632/gamesdb'
   user = 'root'
   password = 'root'
   locations = ['filesystem:src/main/resources/db/migration']
}

task flywayCleanTestDb(type: FlywayCleanTask) {
   url = 'jdbc:postgresql://localhost:5632/test'
   user = 'root'
   password = 'root'
   locations = ['filesystem:src/main/resources/db/migration']
}

task flywayMigrateTestDb(type: FlywayMigrateTask) {
   url = 'jdbc:postgresql://localhost:5632/test'
   user = 'root'
   password = 'root'
   locations = ['filesystem:src/main/resources/db/migration']
}

task flywayInfoTestDb(type: FlywayInfoTask) {
   url = 'jdbc:postgresql://localhost:5632/test'
   user = 'root'
   password = 'root'
   locations = ['filesystem:src/main/resources/db/migration']
}
```

Joonis 38. Gradle kood Flyway käskluste jooksutamiseks kahe erineva andmebaasiga

Samuti tuli teha 2 application.properties faili (Joonis 39.). Üks test kaustas ning teine main kaustas. Need failid näitavad endale määratud andmebaasi peale (*test* ning *gamesdb* vastavalt). Samuti aktiveeritakse testimise jooksul profiil *test* ning muul juhul profiil *production*.

```
spring.profiles.active=test

# DB Config
spring.datasource.url=jdbc:postgresql://localhost:5632/test
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=none

# Set /api suffix
server.servlet.context-path=/api

# Flyway
spring.flyway.baseline-On-Migrate = true
spring.flyway.enabled=true
spring.flyway.connect-retries=60
```

Joonis 39. Testimise jaoks tehtud application.properties fail

4.3.5 Spring Security ning OAuth2

Kuna sisselogimine toimub TalTechi kasutajatega, siis on vaja selle jaoks kasutada OAuth2. Sisseloginud kasutaja nimi ning unikaalne e-mail salvestatakse andmebaasi *Account* objekti. Projektis on kasutuses kaks Spring profiili, esimene ning selle turvalisuse seadistus käivitatakse testide tegemise jooksul ning teine muul juhul. Testide jaoks oli vaja eraldi turvalisuse seadistust, sest muidu Spring Security häiris testide läbiviimist.

OAuth2 kasutab autentimiseks application.properties olevaid Azure OAuth koode, mis on antud TalTechi poolt.

4.3.6 Andmebaasi kirjeldus

Kuna on vaja palju informatsiooni salvestada, siis on vaja korralikult üles-ehitatud andmebaasi. Andmebaasi keskosas on *Account* ehk kasutaja, millega lähedalt seotud on ruumid, arhiivid, küsimustikud ning kaardistikud.

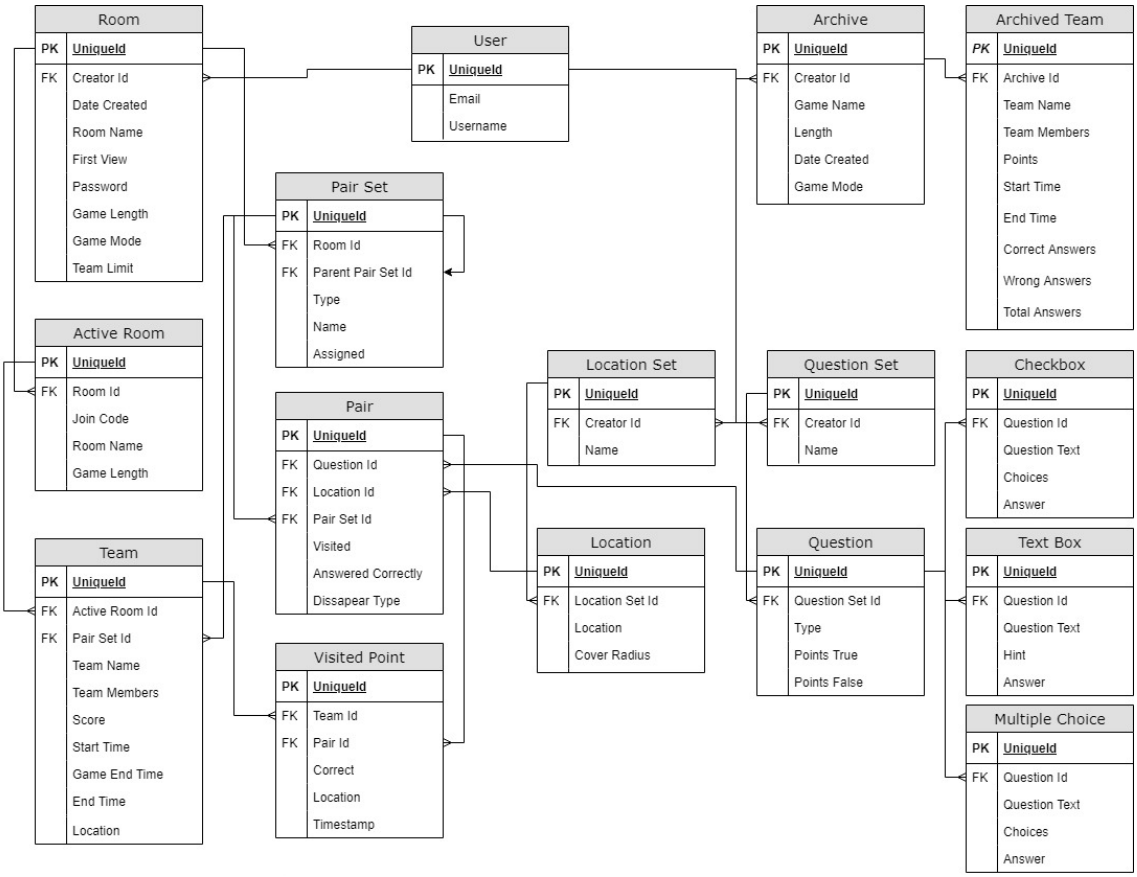
Arhiivid salvestavad omakorda arhiveeritud tiimide kohta informatsiooni. *Location* ning *Question Setid* salvestavad vastavalt *Locationite* ning *Questionite* kohta informatsiooni. *Locationid* ja *Questionid* tehtakse omakorda *Pairideks* (paarideks) ning *Pairid* salvestatakse *Pair Seti* külge. Küsimustega on ka seotud alamtüübid, sest erinevad küsimuste tüübid vajavad erinevaid välju.

Roomi tabeliga on seotud *Active Room* tabel, kuhu lisatakse sissekanne *Room* objekti aktiveerimisel. *Active Roomi* ning *Pair Seti* tabeliga on seotud *Teami* tabel, kus esimene

näitab mis aktiivsesse ruumi too kuulub ning teine viitab tiimile määratud paaridele(kaardil olevatele punktidele).

Teami ning *Pairi* tabeliga on omakorda seotud *Visited Point* tabel, kuhu lisatakse sissekanne kui tiim jõuab ühte enda punktidest kohale.

Andmebaas on realiseeritud PostgreSQL keeles (Joonis 40.) .



Joonis 40. Lõplikult realiseeritud andmebaas

Kuna tabelid on sõltuvuses üksteisest, siis nad tuleb lisada kindlast järjekorras. Igas SQL failis (nt. Joonis 41.) on tehtud üks tabel ning ka sisse pandud mõned testandmed.

```
CREATE TABLE IF NOT EXISTS account (
   id           SERIAL      PRIMARY KEY,
   email         VARCHAR(20) UNIQUE NOT NULL,
    account_name   VARCHAR(20) UNIQUE NOT NULL
);
```

Joonis 41. SQL kood *account* tabeli loomisest

Tabelite lisamise järjekord, samal real olevad tabeleid võib lisada suvalises järjekorras:

1. *account* tabel
2. *room, question set, location set, archive* tabelid
3. *active room, pair set, question, location, archived team* tabelid
4. *pair, multiple choice question, checkbox question, textbox question* tabelid
5. *pair set* tabel
6. *team* tabel
7. *visited point* tabel

SQL failidele tuleb anda Flywayle sobilikus formaadis (Joonis 42.) nimi, et Flyway käsklused töötaksid korrektselt ning andmed liiguksid käskluste abil andmebaasi. Failid peavad olema järgnevas formaadis:

1. Eesliide: V, U või R
2. Versiooni number: Eraldatud punktidega või alakriipsudega.
3. Eraldaja: 2 alakriipsu
4. Kirjeldus: eraldatud ala kriipsudega või tühikutega
5. Järelliide: .sql

```
V1.3.1__Create_Account_table.sql
```

Joonis 42. Flyway näidis nime formaat

4.3.7 Suhtlemine andmebaasiga

Igas andmebaasis olev rida realiseeritakse *entity* [32] (Joonis 43.) ehk üksusena. Üksus on tavaline Java objekt, mis kirjeldab andmebaasis olevaid andmeid. Üksust peab märgistama @Entity annotatsiooniga klassi peal. Igal üksusel on olemas enda väljad vastavalt temale määratud tabeli kirjeldusele.

Id ehk primaarvõtme väljad on märgistatud @Id ning @GeneratedValue annotatsiooniga [33]. Viimasel peab olema lisa parameeter, mis määrab id genereerimise strateegia.

```
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Account {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 20, unique = true)
    private String accountName;

    @Column(length = 20, unique = true)
    private String email;

    @OneToMany(mappedBy = "account")
    @JsonIgnore
    private List<Room> rooms;

    @OneToMany(mappedBy = "account")
    @JsonIgnore
    private List<QuestionSet> questionSets;

    @OneToMany(mappedBy = "account")
    @JsonIgnore
    private List<LocationSet> locationSets;

    public Account(String accountName, String email) {
        this.accountName = accountName;
        this.email = email;
    }

}
```

Joonis 43. *Account* entity kood

Kõik üksused on seotud enda repositooriumitega [34]. Spring Datal on sisseehitatud klass andmebaasidega suhtlemiseks nimega JPA Repository (Joonis 44.). Eelnimetatud klass võtab sisse *Entity* ja selle Id välja tüübi ning selle abil nad automatiseerivad suhtlust andmebaasiga.

```
public interface AccountRepository extends
JpaRepository<Account, Long> {

    Optional<Account> findAccountByAccountName(String name);

    Optional<Account> findAccountByEmail(String email);

}
```

Joonis 44. *Account* Repository kood

4.3.8 Teenuste kiht

Järgmine vahekiht on *Service Layer* ehk teenuste kiht. Teenustes asub peamine äriloogika. Teenused suhtlevad eelnevalt peatükis mainitud JPA Repository klassidega ning nende kaudu salvestatakse andmebaasi uusi andmeid, tehakse päringuid, muudetakse ning kustutatakse andmeid.

Teenused on üles ehitatud nii, et teenused võivad suhelda ka omavahel kui ka JPA Repository klassidega. Üldiselt on iga tabeli kohta tehtud eraldi teenus (Joonis 45.). Teenustes toimib peamine errori-töötlus ning andmete manipuleerimine ja salvestamine.

Peale sisse tulnud andmete töötlemist saadetakse tagasi vastus *Optional<Object>* või *Boolean* kujul. Kui andmete töötlemine õnnestus saadetakse tagasi *Optional*, kus on *true* või manipuleeritud andmed sees. Vastasel juhul saadetakse tühi *Optional* või *false* väärtus.

```
public interface AccountService {

    Optional<AccountDTO> saveAccount(AccountDTO accountDTO);

    Optional<AccountDTO> getAccountById(Long id);

    Optional<AccountDTO> getAccountByEmail(String email);

    Optional<AccountDTO> getAccountByAccountName(String
accountName);

    Optional<Account> getAccountObjectById(Long id);

    Optional<List<AccountDTO>> getAllAccounts();

    Boolean deleteAccount(Long id);

    Boolean hasAnyActiveGames(Account account);

}
```

Joonis 45. *Account* teenus

4.3.9 Kontrollerite kiht

Teenuste ning API vahelist suhtlemist juhendab *controller layer* ehk kontrollerite kiht. Kontrollerid võtavad sisse välist informatsiooni (Joonis 46.), annavad selle informatsiooni vastavale teenusele edasi, kus seda informatsiooni töödeldakse ning saadetakse tagasi vastus.

Sõltuvalt vastusest tagastatakse *ResponseEntity* objekt, mis kujutab endast tervet *HTTP* vastet: staatust, päist ning keha. Kui päring teenusesse õnnestus, tagastatakse *HTTP* staatus *OK* või *CREATED* ning tagastatav objekt. Kui päring ebaõnnestus, siis tagastatakse *HTTP* staatus *NO_CONTENT*.

```
@RestController
@RequestMapping("/accounts")
@RequiredArgsConstructor
public class AccountController {

    private final AccountServiceImpl accountService;

    @GetMapping("{id}")
    @ResponseBody
    public ResponseEntity<AccountDTO>
getAccountById(@PathVariable Long id) {
        Optional<AccountDTO> account =
accountService.getAccountById(id);
        if (account.isEmpty()) return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
        return ResponseEntity.ok(account.get());
    }

    @DeleteMapping("{id}")
    @ResponseBody
    public ResponseEntity<Boolean> deleteAccount(@PathVariable
long id) {
        boolean deleted = accountService.deleteAccount(id);
        if (!deleted) return new ResponseEntity<>(false,
HttpStatus.NO_CONTENT);
        return ResponseEntity.ok(true);
    }

}
```

Joonis 46. *Account* kontroller koos kahe meetodiga

4.3.10 REST ning Data Transfer Object

Kontrolleritel on vaja mingisugust objekti, millega suhelda välismaailmaga. Sageli ei piisa lihtsalt paarist parameetrist, mis saab kaasa anda. *Entityd* ehk üksused ei sobi selleks, sest nad sageli sisaldavad informatsiooni, mis ei ole sobiv näidata tavakasutajale. Selleks vahelüliks on *data transfer object* ehk DTO [35] (Joonis 47.), mille ainuke eesmärk on soovitud andmete edastus.

```
@Getter
@Setter
public class AccountDTO {

    private Long id;
    private String userName;
    private String email;

}
```

Joonis 47. *Account* DTO

DTO on vaheobjekt, millega saadetakse informatsiooni välismaailmast kontrollerisse ning kontroller annab selle edasi teenusele. Üksuse *data transfer objectiks* muutmise loogika asub teenuste kihis (Joonis 48.). Teenus võtab sealt andmeid, teeb vajadusel uusi üksuseid ning salvestab või muudab andmeid ning tagastab vajadusel kontrollerile uuendatud DTO-d.

```
private AccountDTO convertAccountToDto(Account account) {
    AccountDTO accountDTO = new AccountDTO();
    accountDTO.setEmail(account.getEmail());
    accountDTO.setId(account.getId());
    accountDTO.setUserName(account.getAccountName());

    return accountDTO;
}
```

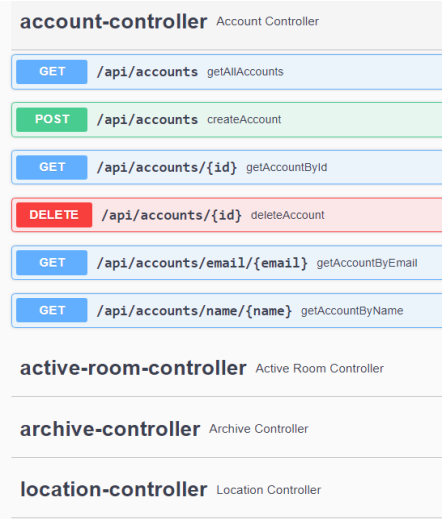
Joonis 48. *Account entity* muutmine DTO objektiks

Kontroller võtab selle DTO uuesti vastu ning annab selle API-le edasi vastavalt meetodisse kirjutatud *HTTP* staatusega [36].

4.3.11 API

Projekti *back-end* on seadistatud käivituma pordil 8080. API-le ligipääsemiseks on vajalik URLile lisada /api järelliide. Igal kontrolleri meetodil on määratud eraldi *endpoint*, kuhu vastavalt saab saata kas POST, GET, DELETE või PUT päringut. Kuna *endpointe* on palju, siis on projektile lisatud Swagger-UI, mis aitab visualiseerida API võimalusi. Swagger-UI kuvamiseks tuleb /api järelliite taha lisada /swagger-ui.

Täpseid aadresse mingisuguse päringu tegemise jaoks vajalikku informatsiooni on võimalik lähemalt näha Swagger-UI vaadates (Joonis 49.) või spetsiifilist kontrolleri klassi vaadates.



Joonis 49. Swagger-UI visualiseeritud API

4.3.12 Loogika

Mängitava ruumi tegemine ning ruumi lõpetamine toimub teatud järjekorras kindla tsüklina. Kõik sammud ühe astme peal peavad olema tehtud enne järgmisele astmele minemist, sest järgmisel tasemel olevad objektid võivad sõltuda eelmisel tasemel olevatest objektidest. Samal astmel asuvaid samme võib teha suvalises järjekorras. Osasid samme ei pea samas ruumis uuesti tegema kui uut mängu hiljem aktiveerida. Igas tsüklis peab aga iga kord uuesti tegema kõik sammud alates allolevas nimekirjas 5. sammust.

1. Luua kasutajale *Account* kui pole varem loodud
2. Luua *Room*, *Question Set*, *Location Set* ning *Pair Set*
3. Luua *Questionid* ning *Locationid*
4. Luua *Pairid*
5. Luua *Active Room*
6. Luua *Team*
7. Luua *Visited Point*
8. Luua *Archive* ning sellega seotud *Archived Team*

4.3.13 Account

Esimest korda OAuth kaudu Tallinna Tehnikaülikooli kasutajaga sisse logides tuleb teha API kaudu *Account* objekt, mis seotakse *String* e-maili ning *String* nime kaudu andmebaasi.

Seotud API meetodite funktsionaalsus:

1) Loo uus
2) Otsi kõike
3) Otsi id järgi
4) Otsi emaili järgi
5) Otsi nime järgi
6) Kustuta

4.3.14 Room

Room objekti loomiseks on vaja olemasolevat *Account* objekti. Loomise jaoks peab ruumi DTO-le kaasa andma parameetrid: *String* roomName, *String* firstView, *Integer* gameLength, *String* gameMode ning *Long* creatorId. FirstView sees kokkulepitud formaadis laius ning pikkuskraadi koordinaadid, kust mängu vaatamist saaks alustada. GameLength on mängu pikkus minutites ning gameMode üks järgnevast: *normal*, *rush* või *preassigned*. GameMode parameetrit ei ole võimalik hiljem muuta ning peab uue ruumi objekti tegema teise mängu tüübi valimiseks.

Kui mängu tüüp pole normal, siis peab kaasa andma ka *Integer* teamLimit parameetri arvuna, mis piirab kui palju tiime saab maksimaalselt aktiivse ruumiga hiljem liituda. On olemas veel *String* password, mille mõte seisneb selles, et ruumi käsitlemiseks oleks vaja parool sisestada. See on ainult andmebaasis realiseeritud hetkeseisuga.

Eelnimetatud ruumi parameetreid välja arvatud gameMode on võimalik hiljem muuta ja kustutada, kui ruumil ei ole aktiivset mängu.

Seotud API meetodite funktsionaalsus:

1) Loo uus
2) Uuenda
3) Otsi kõike kasutaja id järgi
4) Otsi kõike
5) Otsi id järgi
6) Kustuta

4.3.15 Question Set

Uue *Question Set* objekti loomiseks on vaja olemasolevat *Account* objekti. Selle loomiseks tuleb anda DTO-le kaasa *String* name ning *Long* creatorId parameetrid. Esimene ei ole kohustuslik, kuid soovitatav kergema visualiseerimise eesmärgil.

Hiljem on võimalik taaskasutada kasutaja loodud question sete sama või uute ruumide puhul. *Question Seti* on võimalik hiljem muuta ja kustutada kui kasutajal ei ole hetkel aktiivset mängu.

Seotud API meetodite funktsionaalsus:

1) Loo uus
2) Otsi kõiki
3) Otsi kõiki *Account* id järgi
4) Uuenda
5) Kustuta

4.3.16 Question

Question objekti loomiseks on vaja olemasolevat *Question Set* objekti. DTO-le tuleb kaasa anda järgmised parameetrid: *Long* questionSetId, *Integer* pointsTrue ning *Integer* pointsFalse. Viimased 2 võivad olla mõlemad nii negatiivsed kui ka positiivsed. *Questioni* tüüp määratakse automaatselt sõltuvalt sellelt, mis meetodis ta luuakse. See on üks järgnevast:

A. “TEXT”
B. “CHECKBOX”
C. “MULTIPLE_CHOICE ”

Kuna erinevad *Questioni* tüübid soovivad erinevaid parameetreid, siis vastavalt nende loomiseks on päring vaja saata erinevale URL-ile, sest kõikidele on tehtud eraldi vastav DTO.

Multiple Choice tüüpi *Questioni* loomiseks on vaja kaasa anda *String* questionText, *List<String>* choices ning *String* answer. Eelnimetatud väljad on kohustuslikud ning neid saab hiljem muuta vastava päringu tegemisega.

Text tüüpi questioni tegemiseks on vaja kaasa anda *String* questionText, *String* hint ning *String* answer. Eelnimetatud väljad on kohustuslikud ning neid saab hiljem muuta vastava päringu tegemisega.

Checkbox tüüpi questioni loomiseks on vaja kaasa anda *String* questionText, *List<String>* choices ning *List<String>* answer. Eelnimetatud väljad on kohustuslikud ning neid saab hiljem muuta vastava päringu tegemisega.

Algselt luuakse ühiste parameetride järgi baas *Question* objekt, peale seda tehakse alamküsimus vastavalt tüübile ning seotakse see baas küsimusega. Kustutamine toimub baas *Questioni* objekti kaudu, mille kustutamine automaatselt kustutab ka sellega seotud alamküsimuse.

Questioni parameetreid on võimalik hiljem muuta ja kustutada kui kasutajal ei ole hetkel aktiivset mängu.

Seotud API meetodite funktsionaalsus:

- Loo uus *Checkbox* tüüpi küsimus
- Loo uus *Multiple Choice* tüüpi küsimus
- Loo uus *Text Box* tüüpi küsimus
- Uuenda *Checkbox* küsimust
- Uuenda *Multiple Choice* küsimust
- Uuenda *Text box* küsimust
- Otsi kõiki
- Otsi kõiki *Question Set* id järgi
- Otsi id järgi
- Kustuta

4.3.17 Location Set

Uue *Location Seti* loomiseks on vaja olemasolevat *Account* objekti. DTO-le tuleb kaasa anda kaasa *String* name ning *Long* creatorId parameetrid. Esimene ei ole kohustuslik, kuid soovitatav kergema visualiseerimise eesmärgil.

Hiljem on võimalik taaskasutada kasutaja loodud *Location Sete* sama või uute ruumide puhul. *Location Sete* on võimalik muuta ning kustutada kui kasutajal ei ole hetkel aktiivset.

Seotud API meetodite funktsionaalsus:

1) Loo uus
2) Uuenda
3) Otsi kõiki
4) Otsi kõiki *Account* id järgi
5) Otsi id järgi
6) Kustuta

4.3.18 Location

Locationi loomiseks on vaja olemasolevat *Location Set* objekti. Loomisel tuleb DTO-le kaasa anda *Long* locationSetId, *Integer* coverRadius ning *String* location parameetrid. Esimene seob *Locationi* kindla *Location Setiga*. Teine on arv meetrites, kui kaugel punkt peaks aktiveerima ning viimane on kokkulepitud formaadis sõne.

Hetkel on kasutuses formaat *"{"lat": koordinaat, "lng": koordinaat}"*, kus 1. koordinaat ja 2. koordinaat on vastavad koordinaadid ning *lat* määrab laiuskraadi ja *lng* pikkuskraadi.

Hiljem on võimalik taaskasutada kasutaja loodud *Location* objekti sama või uute ruumide puhul. *Locationit* on võimalik muuta ning kustutada kui kasutajal ei ole hetkel aktiivset mängu.

Seotud API meetodite funktsionaalsus:

1) Loo uus
2) Uuenda
3) Otsi kõiki *Location Set* id järgi
4) Otsi kõiki
5) Otsi id järgi

6) Kustuta

4.3.19 Pair Set

Uue *Pair Set* objekti loomiseks on vaja olemasolevat *Room* objekti. DTO-le tuleb kaasa anda *String* name ning *Long* roomId ning *String* type parameetrid. Esimene ei ole kohustuslik, kuid soovitatav kergema visualiseerimise eesmärgil.

Samuti on *Pair Set* objektil olemas *Boolean* assigned, mis näitab, kas ta on mingisuguse *Teamiga* seotud. Seda ei ole vaja kaasa anda. Type võib olla: „original“ või „copy“.

Pair Setidest on võimalik teha koopiaid, mis teeb ka koopiad *Pair Setis* olemasolevatest *Pair* objektidest. Koopia on täiesti uus objekt, mis viitab uutele *Pair* objektidele. Pair Setidest on võimalik teha mõlemat tüüpi koopiaid. *Pair Setid* tüübiga „copy“. kustutatakse, kui aktiivne mäng lõpetatakse.

Hiljem on võimalik taaskasutada kasutaja loodud pair sete sama või uute ruumide puhul. *Pair Sete* on võimalik muuta ning kustutada kui kasutajal ei ole hetkel aktiivset mängu.

Seotud API meetodite funktsionaalsus:

1) Loo uus
2) Loo koopia
3) Otsi kõiki
4) Otsi kõiki *Room* id järgi
5) Otsi kõiki originaal tüübiga *Pair Sete Room* id järgi
6) Otsi kõiki koopia tüübiga *Pair Sete* originaalse *Pair Set id* järgi
7) Otsi id järgi
8) Kustuta
9) Kustuta kõik *Room* id järgi

4.3.20 Pair

Paaride loomiseks on vaja olemasolevaid *Pair Set*, *Question* ning *Location* objekte. *DTO*le tuleb kaasa anda *Long* pairSetId, *Long* questionId, *Long* locationId ning *String* disappearType parameetrid. Viimane võib olla üks kahest väärtusest: „*correct“* või

„answered“ . See määrab ära, et kas punkt peaks ära kaduma õigesti vastates või lihtsalt vastates.

Pairil on ka lisa *Boolean* visited ning answeredCorrectly parameetrid, mida teenused uuendavad, kui luuakse käesoleva *Pair* objektiga seotud *Visited Point* objekt .

Kui tehakse *Pair Setist* koopia, siis tehakse ka automaatselt seal olevatest *Pair* objektidest täiesti uued koopiad, mis seotakse selle *Pair Setiga*.

Samuti on olemas võimalus seadistada kõik paarid algsetele väärtusele, mida tehakse automaatselt peale iga aktiivse ruumi lõpetamist.

Hiljem on võimalik taaskasutada kasutaja loodud pair sete sama või uute ruumide puhul. *Pair Sete* on võimalik muuta ning kustutada kui kasutajal ei ole hetkel aktiivset mängu.

Seotud API meetodite funktsionaalsus:

- Loo uus
- Uuenda
- Otsi kõiki *Pair Set* id järgi
- Otsi kõiki
- Otsi id järgi
- Kustuta

4.3.21 Active Room

Active Room objekti loomiseks on vaja olemasolevat *Room* objekti. Kaasa tuleb anda *Long* roomId ning *String* roomName parameetrid. Viimast kasutatakse ära arhiveerimise eesmärgil.

Active Room objekti loomisel genereeritakse 6-täheline kood, mis võib koosneda nii suurtest tähtedest, väikestest tähtedest kui ka numbritest. Koodi genereeritakse niikaua kuni leitakse kood, mis ei ole kasutuses. Igal aktiivsel ruumil on enda unikaalne kood, mida ei ole võimalik hiljem muuta.

Active Room objekti kustutamisel kustutatakse ka sellega seotud *Team* ning *Visited Point* objektid. Enne kustutamist arhiveeritakse aktiivne ruum ning tiimidega seotud *Pair Set* objektid tüübiga „copy“ kustutatakse ning „original“ seadistatakse algseadetele.

Seotud API meetodite funktsionaalsus:

- Loo uus
- Otsi kõiki
- Otsi kõiki *Room* id järgi
- Otsi id järgi
- Otsi liitumiskoodi järgi
- Otsi kõiki *Pair Sete Active Room* id järgi
- Kustuta ning arhiveeri
- Kustuta kõik

4.3.22 Team

Team objekti loomiseks on vaja olemasolevat *Active Room* ning *Pair Set* objekti. Selle loomiseks tuleb DTO-le kaasa anda *Long* activeId, *Long* pairSetId, *String* teamName ning *String* teamMembers.

Kui tiim läheb mängu, siis on vajalik kutsuda välja meetodit, mis seadistab temale alustamisaja ning mängu lõpu aja, mis arvutatakse välja *Room* objekti *Integer* gameLength kaudu.

Mängu jooksul on vaja pidevalt uuendada tiimi *String* locationit, mis näitab ta hetkelist asukohta. Samuti on vaja uuendada tiimi pidevalt vaja uuendada punktide arvu, mis on määratletud parameetrina *Integer* score.

Kui tiim kogub kaardi pealt kõik punktid ära ning mäng on tema jaoks lõpetatud, siis on vaja kutsuda välja meetodit, mis seadistab lõpetamisaja.

Team objekti ei ole võimalik muuta ega otseselt kustutada. See kustutatakse koos *Active Room* objektiga.

Seotud API meetodite funktsionaalsus:

- Loo uus
- Uuenda
- Uuenda asukohta
- Määra mängu algus- ning lõppaeg
- Määra lõpetamisaeg
- Otsi kõiki *Active Room* id järgi
- Otsi id järgi
- Otsi nime järgi
- Otsi kõiki paare *Team* id järgi
- Otsi kõiki mitte külastatud paare *Team* id järgi

4.3.23 Visited Point

Visited Point objekti loomiseks on vaja olemasolevat *Pair* ning *Team* objekti. Selle DTO-le tuleb kaasa anda *Long* pairId ning *Long teamId ning Boolean correct*. Loomise jooksul tekitatakse ajatempel ning võetakse tiimi viimane asukoht ja salvestatakse see. Neid väärtuseid kasutatakse hiljem arhiveerimise käigus.

Objekti loomise käigus määratakse viidatud paarile, et seda on külastatud.

Visited Point objekti ei ole võimalik muuta ega otseselt kustutada vaid selle kustutamine toimub *Active Room* objekti kustutamise tagajärjel.

Seotud API meetodite funktsionaalsus:

- Loo uus
- Otsi id järgi
- Otsi kõiki *Team* id järgi
- Otsi kõiki *Active Room* id järgi

4.3.24 Archive ning Archived Team

Archive objekti loomiseks on vaja olemasolevat *Active Room* objekti ning sellega seotud *Room*, *Team* ning *Visited Point* objekte. Arhiveerimine toimub aktiivset ruumi lõpetades,

millega kaasneb ka *Pair* ning *Pair Setide* seadistamine algväärtusele ning lõpuks *Active Room* objekti kustutamine.

Arhiveerimise käigus tehakse uus *Archive* objekt, kuhu salvestatakse aktiivse ruumi nimi, loomise kuupäev, mängu pikkus ning looja kasutaja.

Peale objekti salvestamist käiakse üle kõik aktiivse ruumiga seotud tiimid, ning iga tiimi kohta luuakse *Archived Team* objekt. Sellele objektile salvestatakse alustamise ning lõpetamise aeg. Kui tiim ei lõpetanud, siis võetakse tiimi lõpetamise ajaks mängu lõpu aeg. Samuti salvestatakse tiimi ning tiimiliikmed.

Iga tiimi puhul vaadatakse seotud *Visited Point* objektid ning salvestatakse õigesti vastatud, valesti vastatud ning kogu vastatud küsimuste numbrid.

Peale arhiveerimise protsessi kõik aktiivse ruumiga seotud paarid seadistatakse algväärtusele, et neid saaks uuesti kasutada ning peale seda kustutatakse *Active Room* objekt ning sellega seotud *Team* ja *Visited Point* objektid.

Seotud API meetodite funktsionaalsus:

- Otsi kõiki
- Otsi kõiki *Account* id järgi
- Otsi id järgi
- Otsi kõiki *Archived Team* objekte *Archive* id järgi

5 Valideerimine

Valideerimine on vajalik, et kinnitada, et projektis realiseeritud funktsionaalsused töötaks nii nagu nad on ette nähtud. Samuti välisel vaatlejal võib olla selline perspektiiv, et ta märkab midagi, mis arendajal jäi kahe silma vahele ning mis vajaks kiiresti parandamist.

Projektis on valideerimiseks kasutatud mitut erinevat protsessi. Nendeks protsessides on andmete valideerimine, integratsioonitestimine ning testimised päriselus, mis olid nii tiimisisesed kui ka väliste inimestega.

5.1 Testimine

Projekti on mitu korda iseseisvalt katsetatud. On läbi viidud tervet mängu tegemise protsessi, mis hõlmab kõiki olemasolevaid funktsionaalsuseid. On katsetatud leheküljele sisselogimist ning seda, et kasutajat luuakse korrektselt. Samuti on katsetatud ruumi ning kõikide küsimuste, asukohtade ning nende kogumike loomist. Samuti on mängu mängimist ning punktide korjamist katsetatud päris elus. Veel on katsetatud ruumi lõpetamist ning mängu arhiveerimist.

Kliendi ning juhendaja tagasiside alusel on märkimisväärselt muudetud lehekülje disaini ning kasutaja voogu. Tagasiside on projekti lisatud ka mängu alustamise juhend, mis aitab uutel kasutajatel mängu loomise protsessi läbi viia.

Samuti on mängu tegijate tuttavad katsetanud mängu tegemist ning selle mängimist. Projekti autorid tegid Google Formsi lihtsa ülesandega ning paari küsimusega. Kokku on katsetanud seda 4 inimest (välja arvatud klient ning juhendaja). Kõik inimesed arvasid, et mäng ise on funktsionaalne ning mängitav. Segaseks jäi ühel inimesel küsimustiku loomine ning mainiti, et see oleks võinud olla paremini selgitatud. Tiim oli sellega nõus, nii, et *tutorialitele* lisati igale lehele rohkem selgitusi ning täiendusi. Samuti oli probleeme erinevate elementide paigutamisega, paigutamine võiks ühtlasem olla igal pool. Üks element, mis erines igal pool oli *Tagasi* nupp. See element on nüüd igal pool samas kohas.

Aktiivse mänguga liitumisena olid kerged resolutsiooniprobleemid ning tagumine taustapilt segas arusaamist. Tiim parandas resolutsiooniprobleemid ära ning muutsid taustapildi valgeks.

Küljemenüüs toodi välja, et see kaob poole teksti pealt ära. See viga sai likvideeritud ning küljemenüü ei kao enam liiga vara ära.

Samuti toodi välja, et osad elemendid on erineva pikkusega. Näiteks oli ’Loo uus ruum’ kaart väiksem kui olemasolevad ruumide kaardid. Need said samuti ühtlustatud ning ühe pikkuseks tehtud.

5.2 Andmete valideerimine

Andmed valideeritakse *back-endis* kui kontrollerid saadavad andmed DTO kaudu teenusesse. Projektis on kasutatud Optionale, mis tagastavad kontrollerile kas tühja või täidetud objekti. Tühjad Optionial objektid annavad API-le vastuseks HTTP staatuse NO_CONTENT ning täidetud Optional objektid annavad staatuse CREATED või OK (Joonis 50.).

```
public ResponseEntity<AccountDTO> createAccount(@RequestBody
AccountDTO accountDTO){
    Optional<AccountDTO> account =
accountService.saveAccount(accountDTO);
    if (account.isEmpty()) return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
    return new ResponseEntity<>(account.get(), HttpStatus.CREATED);
}
```

Joonis 50. Account kontrolleris loomise andmete valideerimine

POST päringuga kontrollitakse, kas vajalikud objektid on andmebaasid olemas uue valitud objekti loomiseks. Seejärel vaadatakse kas DTO objektile on kohustuslikud väljad kaasa antud. Tehakse uus valitud objekt, kuhu kantakse andmed sisse ja mis salvestatakse andmebaasi. Kui loomine õnnestus, DTO kujule viidud eelnevalt loodud objekt, mis on Optional objekti sisse pandud. Vastasel juhul tagastatakse tühi Optional (Joonis 51.).

```
@Override
public Optional<AccountDTO> saveAccount(AccountDTO accountDTO) {
    try {
        Optional<Account> accountFoundByEmail =
accountRepository.findAccountByEmail(accountDTO.getEmail());
        Optional<Account> accountFoundByAccountName =
accountRepository.findAccountByAccountName(accountDTO.getUserNam
e());
        if (accountFoundByAccountName.isPresent() ||
accountFoundByEmail.isPresent()) return Optional.empty();

        Account account = new Account();
        account.setEmail(accountDTO.getEmail());
        account.setAccountName(accountDTO.getUserName());

        Account savedAccount = accountRepository.save(account);
        return Optional.of(convertAccountToDto(savedAccount));

    } catch (Exception e) {
        return Optional.empty();
    }
}
```

Joonis 51. Post päringu jaoks tehtud meetod *Account* teenuses

GET päringuga kontrollitakse antud id numbriga objekti olemasolu. Kui objekti ei ole, siis tagastatakse tühi *Optional* objekt. Kui otsitav objekt on olemas, siis tagastatakse täidetud *Optional*, mille sees on DTO kujule tehtud otsitav objekt (Joonis 52.).

```
@Override
public Optional<AccountDTO> getAccountById(Long id) {
    Optional<Account> account = accountRepository.findById(id);
    if (account.isEmpty()) return Optional.empty();
    return Optional.of(convertAccountToDto(account.get()));
}
```

Joonis 52. GET päringu jaoks tehtud meetod *Account* teenuses

PUT päringuga kontrollitakse, et kas otsitav objekt on andmebaasis olemas. Kui leitakse see objekt, siis vaadatakse DTO välju. Kui väli ei ole null, siis seda uuendatakse. Lõpuks salvestatakse ära uuendatud objekt, viiakse DTO kujule ning tagatatakse *Optionali* objekti sees. Kui tekib viga, siis tagastatakse tühi *Optional* objekt (Joonis 53.).

```
@Override
public Optional<TeamDTO> setTeamLocation(TeamDTO teamDTO) {
    try {
        Optional<Team> teamOptional =
teamRepository.findById(teamDTO.getId());
        if (teamOptional.isEmpty()) return Optional.empty();
        Team team = teamOptional.get();

        String location = teamDTO.getLocation();
        if (location != null && !location.isEmpty())
team.setLocation(location);
        Team savedTeam = teamRepository.save(team);
        return Optional.of(convertToTeamDTO(savedTeam));
    } catch (Exception e) {
        e.printStackTrace();
        return Optional.empty();
    }
}
```

Joonis 53. PUT päringu jaoks tehtud funktsioon *Team* teenuses

Delete päringuga kontrollitakse, kas valitud objekt on andmebaasis olemas. Kui objekt on olemas, siis kustutatakse objekt ära ning tagastatakse tõeväärtus. Kui ei ole, siis tagastatakse vale väärtus. Tavaliselt on teenuses kustutamise funktsioonides ka lisafunktsioon, mis kontrollib, kas tohib valitud objekti kustutada sellel ajal (Joonis 54.).

```
@Override
public Boolean deleteAccount(Long id) {
    Optional<Account> account = accountRepository.findById(id);
    if (account.isEmpty()) return false;
    accountRepository.deleteById(id);
    return true;
}
```

Joonis 54. DELETE päringu jaoks tehtud meetod *Account* teenuses

5.3 Testimine

Kaks tuntumat testimise meetodit on ühiktestimine ning integratsioonitestimine [37]. Ühiktestid on tavaliselt isoleeritud ning nad ei sõltu teistest klassidest ega nende töökäigust. Nendega kontrollitakse sageli, et kas mingisugune funktsioon töötab korrektselt. Integratsioonitestid kontrollivad erinevate komponentide koostööd ning nende ühildumist üksteisega. Nad kontrollivad suuremat tervikut ning selle terviku töö lõpptulemust.

Selle projektiga keskenduti integratsioonitestimisele, sest kõik komponendid on mingil määral sõltuvusest üksteisest ning isoleeritud ühiktestimise eesmärgid ja funktsioonide

töötamine saab kontrollitud integratsioonitestimise jooksul. Kui kõik komponendid iseseisvalt töötavad korrektselt, siis saadakse õige vastus.

Testimise jooksul kasutati SQL failidesse lisatud juba eelnevalt lisatud test andmeid ning loodi ka uusi andmeid. Igale kontrollerile on välja arvatud OAuth kontrollerile on tehtud integratsioonitestid.

Testide jaoks on kasutatud JUnit 4 raamistikku ning Spring Booti sisseehitatud klassi *TestRestTemplate'i*, mis võimaldab suhtlust API-ga. Testide jooksul kontrolliti kõike GET, POST, PUT ning DELETE päringuid. Samuti kontrolliti HTTP staatust ning vastuse keha ja selles olevaid andmeid (Joonis 55.).

```
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
public class AccountControllerTests {

    public static final
ParameterizedTypeReference<List<AccountDTO>> LIST_OF_ACCOUNTS =
new ParameterizedTypeReference<>() {
    };

    @Autowired
    private TestRestTemplate testRestTemplate;

    @Test
    void canGetAllAccounts() {
        ResponseEntity<List<AccountDTO>> exchange =
testRestTemplate.exchange("/accounts",
                HttpMethod.GET, null, LIST_OF_ACCOUNTS);
        List<AccountDTO> savedAccounts = assertStatus(exchange,
HttpStatus.OK);
    }



    private <T> T assertStatus(ResponseEntity<T> exchange,
HttpStatus status) {
        assertNotNull(exchange.getBody());
        assertEquals(status, exchange.getStatusCode());
        return exchange.getBody();
    }

}
```

Joonis 55. *Account* kontrolleri testide klass koos ühe testiga

Algseks eesmärgiks oli lõpptulemusena saavutada kõrge testide kattumus. Kokku on tehtud 90 integratsioonitesti, ning saavutati 93% meetodite kattumus ning 87% ridade kattumus.

Selle tulemusena kindlustatakse *back-endi* töökindlus ning funktsionaalsust. Kui midagi olemasolevast funktsionaalsusest läheb tulevikus katki, siis testid enam ei tööta. Samuti see minimeerib *bugide* olemasolu ning teket.

Teste jooksutatakse peale igat muudatust igas harus ning õnnestunud *deploy* jaoks on vaja, et kõik testid läheksid läbi. GitLab *runner* tühjendab Flyway käskluste abil andmebaasi ning laeb sinna peale SQL failides olevad algandmed. Peale seda jooksutatakse kõik testid ning tulemus läheb kirja GitLabi torustikku.

6 Tulemused

Semestri jooksul on valmis tehtud mängitav seiklusmäng, kus on 3 erinevat mängu tüüpi. Nendeks on *normal*, *rush* ning *preassigned*. Esimesel mängu tüübil on igal tiimil iseseisev kaart ning teised tiimid ei mõjuta nende punkte. Teisel mängu tüübil on kõikidel tiimidel sama kaart ning punkt kaob ära kui antakse nii õige kui ka vale vastus. Viimasel mängutüübil on piiratud tiimide arv ning igale tiimile on eraldi sätitud paarid, mida nad peavad läbima.

Mängu leheküljele on võimalik sisse logida ning uusi ruume luua. Peale ruumi loomist on võimalik teha küsimusi, asukoha punkte ning küsimustikke ja kaardistikke. Peale seda peab moodustama paarid varem tehtud küsimustest ning asukohtadest ning seadistama nad kogumikele. *Preassigned* mängu tüübil peab tegema tiimidele eraldi *Pairide* kogumid. Vastavalt valitud mängu tüübile seotakse paarid liituvatele tiimidele.

Mängu saab alustada seda aktiveerides, peale mida genereeritakse liitumiskood ning mängijad saavad liituda. Sõltuvalt mängutüübist on tiimide arvuline piirang.

Mängija saab liituda suvaliselt genereeritud 6-tähelise liitumise koodi abil. Peale seda on vajalik sisestada tiimi nimi ning tiimi liikmed. Mängija saab oma initsiatiivil mängima minna. Mäng lõpeb kui kõik punktid on ära korjatud või kui mängu aeg saab läbi. Kui tiim lõpetab, siis talle seatakse lõpetamise aeg. Peale mängu lõppemist automaatselt arhiveeritakse mäng ning sellega seotud tiimide informatsioon ning seatakse kasutatud paarid uueks kasutuseks valmis.

Mängu loojal on mängimise ajal kõikide tiimide tegevusest võimalik vaadata. On näha tiimide edetabelit, nende asukohti värvi abil ning hetkel olevat punktide arvu.

Peale mängu lõppemist lõpetatakse aktiivne ruum ning arhiveeritakse info. Loodud paare on võimalik pärast uuesti kasutada.

7 Kommentaarid

Käesoleva projekti arendamise jooksul on kasutatud erinevaid tehnoloogiaid nagu Spring Boot, Angular, Java, Docker, Flyway, Gradle jms.

Projekti arendamise jooksul saime endale seotud eesmärgid täidetud ning tunneme, et oleme valinud oma tehnoloogilise *stacki* hästi. Oleme saanud hulga kogemust ning üldisi tehnoloogisi valikuid me ei muudaks. Valisime tehnoloogiad peamiselt sellepärast, et me olime nendega varem tegelenud ning teadsime, et nad on töökindlad.

Front-endi jaoks kasutatud tehnoloogiad Angular TypeScriptiga, Bootstrap ning OpenLayers on meil võimaldanud kokku panna funktsionaalse ning moodsa kasutajaliidese. Angulariga saime valmis teha erinevaid komponente ning neid kuvada, kus vajalik. Bootstrap tegi meil mobiilivaate tegemise võimalikult mugavaks. Seda oli ka lihtne kasutada ning paljud komponendid, kui siis mitte enamus, kasutavad Bootstrapi klasse. OpenLayers oli meile uus tehnoloogia, kuid saime sellega hästi toime. See võimaldas meil kõiki funktsionaalsusi implementeerida ning ei jäänud väheseks. Kuigi esines seal mõni tõrge, siis lahendasime seda teistmoodi.

Back-endi valitud tehnoloogiad Spring Boot koos Java, Dockeri, Flyway ning PostgreSQL-iga on võimaldanud meil välja töötada lahendus, mis suudab salvestada erinevate objektide kohta andmeid. PostgreSQL-iga ning Dockeriga on tehtud andmebaasi lahendus, mis suudab hallata kõike mänguks vajalikke objekte. Andmebaasis olevaid andmeid ning tabeleid on võimalik hallata Flywayga. Neid andmeid saab otsida, muuta ning kustutada vastavalt vajadustele tänu Spring Bootile ning Javale. See on võimaldanud meil välja töötada funktsionaalne ning mitmekesine REST API, mis täidab *front-end* kihi soove ning vajadusi.

Alternatiivina oleks võinud REST API asemel implementeerida WebSocket API, mis lubaks luua otsese ühenduse kliendi ja serveri abil. Sellega polnud aga ühelgi meist kogemust ning kollektiivselt otsustasime REST API tegemis jaoks.

8 Järeldused ja edasised sammud

Projektiga on jõutud staadiumi, kus see on funktsionaalne ning mängitav. Eesmärgiks seatud funktsionaalsus on implementeeritud.

Tiim on saavutanud oma eesmärgid, mis nad on semestri jooksul seadnud. Projekti arengu ning arendamisprotsess on dokumenteeritud GitLabis erinevate võimaluste nagu *issue'de, milestone'ide* ning harude abil.

Tiim on valinud oma tehnoloogilise *stacki* hoolikalt ning on üldiselt sellega rahul, sest see on võimaldanud meil implementeerida kõik vajalikud funktsionaalsused.

Käesolevas projektis on seletatud lahti, kuidas on üles ehitatud *front-end* ning selle graafiline liides. Projektis lahti seletatud erinevad vaated ning kuidas nende funktsionaalsused. On kirjutatud detailne üles ehitus graafilise liidese taga taustal olevast *back-endis* nii seletuste andmebaasi kohta kui ka erinevate objektide ning nende kasutusloogika kohta.

Samuti on tehtud ära valideerimine erinevate meetoditega. Tiimiliikmed on ise käinud katsetamas, samuti on katsetanud juhendaja ning klient. Lisaks sellele toimub taga taustal andmete valideerimine nii *front-endis* kui ka *back-endis.* Samuti on tehtud mahukad integratsioonitestid, mis vähendavad *bugide* tekkimise võimalust märgatavalt.

Sellegipoolest ei ole projekt veel valmis ning leidub võimalusi edasi arendamiseks. Implementeerimist vajavad järgnevad asjad:

1. Chat room - koht, kus mängus viibivad inimesed saaksid üksteisega rääkida.
2. Mängijate sünkroniseerimine - Hetkel mängijad lähevad enda initsiatiivil mängima. Ooteruumi võiks sünkroniseerida kuidagi, et mängu looja saab alustada mängu ning see algab kõikidel mängijatel samaaegselt.
3. Web hookid - Kui mingisugune tegevus lõpetatakse, siis saadetakse päring koos informatsiooniga mingisugusele URL aadressile, mis kasutaja seadistab.

4. Andmetöötlus - Oleks võimalik teha statistikat ning visualiseerida näiteks tiimide teekondasid, mis tüüpi küsimustele vastati rohkem õigesti, kui kaua võttis keskmine mäng aega, kui palju punkte keskmiselt tiimid teenisid jne.
5. Erinevad mängutüübid - Praeguse seisuga on projektis implementeeritud kolm erinevat mängu tüüpi *normal*, *rush* ning *preassigned*. Oleks võimalik veel erinevaid mängu tüüpe välja mõelda, et mängimise varieeruvust ning keerukust suurendada.
    1. Näiteks ajastatud paaridega mängutüüp. Kui õigeks ajaks ei jõua punktini, siis punkt kaob ära.
    2. Paarid, mis avavad teisi paare. Kui ühe paari õigesti või valesti vastad, siis ta näitab sulle järgmist punkti, kuhu liikuma pead.
6. Kasutajaliides - Alati saab graafilist liidest mugavamaks ning ilusamaks teha kasutajale.
7. Rollide põhine autentimine - Hetkel toimub autentimine sisselogimise põhjal. OAuth2 saadud informatsioon Tallinna Tehnikaülikooli kasutajaga sisse logides ei tagasta informatsiooni, kas sisse loginud kasutaja on õppejõud või tudeng. See tuleks andmebaasi põhiselt implementeerida ning teistel õppejõududel võiks olla õigus määrata kasutajale õppejõu (mängu looja) õigused.

9 Kokkuvõte

Semestri jooksul on saanud valmis mängitav projekt, kus on töötavad andmebaas, testid, mängu loogika ning ilus ja funktsionaalne graafiline liides. Kõik seatud eesmärgid said saavutatud. Samuti on dokumenteeritud terve progress GitLabis.

Projekti liikmed on kõik hoogsalt panustanud projekti arendamise käigus. Kaks inimest tegeles *front-endiga* ning üks *back-endiga.* Kõik inimesed täitsid nädalate kaupa erinevaid ülesandeid ning valmis on saanud töötav ning funktsionaalne projekt, mis on oma eesmärke täitnud ning mida on võimalik tulevikus edasi arendada.

Karl-Erik Hein tegeles peamiselt *back-endiga.* Tema ülesannete hulka kuulusid näiteks andmebaasi ning koodi haldamine, uuendamine ning refaktoreerimine. Lisaks tegi ta integratsiooniteste, haldas GitLabi *runnereid* ning Flywayd.

Robin-Kevin Koppa tegeles peamiselt *front-endiga*. Tema ülesannete hulka kuulus veebilehe üleüldine vaade ehk UI disain, õpetuse tegemine, sisselogimine veebilehel. Lisaks *back-endi* koodi ülevaatamine ja dokumenteeris iga nädalaseid koosolekuid.

Artur Kerb tegeles peamiselt *front-endiga*. Tema ülesannete hulka kuulus veebilehe vaadete disainimine, kaardi impleminteerimine koos markeritega, paaride loomise loogika tegemine ning ka mängude loomine. Lisaks aitas kaasa UI disanimis otsuseid teha ning *front-end* koodi ülevaatamine.

Valminud projektis on võimalik Tallinna Tehnikaülikooli kasutajaga sisse logida, luua, kustutada ning muuta ruume, kus saab teha kogumikke vastavate küsimuste ning asukohtade punktidega.

Ruumidel on kolm erinevat mängitavat mängu tüüpi. *Normal* mängutüübil saavad mängijad eraldi identsed paaristikud ning paarid kaovad ära nii õigesti kui ka valesti vastates. *Rush* mängutüübil jagavad mängijaid punkte ning paar kaob korrektselt vastates. *Preassigned* mängutüübil on võimalik tiimidele eraldi teha sättida tiimidele paaristikke vastavalt enda soovidele.

Mängu aktiivseks tegemisel genereeritakse kood, mille abil mängijad saavad mänguga liituda. Mängijad peavad koguma punkte ning erinevate küsimuste tüüpidele õigesti vastama. Peale mängimist on võimalik mängu tegijal vaadata varasemalt tehtud arhiveeritud mänge

Mängu tegijal on mängu ajal ülevaade värviga eristatud mängus olevates tiimidest, nende punktidest ning nendele määratud paaridest. Samuti on võimalik hiljem vaadata arhiveeritud mänge ning nendega seotud informatsiooni.

Kasutatud kirjandus

[1] „Angular,“ 2021. [Võrgumaterjal]. Available: https://angular.io/guide/what-is-angular. [Kasutatud 26 04 2021].

[2] „TypeScript,“ 2021. [Võrgumaterjal]. Available: https://www.typescriptlang.org/. [Kasutatud 25 04 2021].

[3] J. Gosling, B. Joy, G. Steele ja G. Bracha, „The Java Language Specification, Third Edition,“ [Võrgumaterjal]. Available: https://www.researchgate.net/publication/200040359_The_Java_Language_Specification_Third_Edition. [Kasutatud 25 04 2021].

[4] „OpenLayers 2,“ 2008. [Võrgumaterjal]. Available: http://docs.openlayers.org/. [Kasutatud 25 04 2021].

[5] „Gradle,“ 2021. [Võrgumaterjal]. Available: https://docs.gradle.org/current/userguide/what_is_gradle.html. [Kasutatud 25 04 2021].

[6] „Spring,“ 2021. [Võrgumaterjal]. Available: https://spring.io/projects/spring-boot. [Kasutatud 25 04 2021].

[7] „PostgrSQL,“ 2021. [Võrgumaterjal]. Available: https://www.postgresql.org/. [Kasutatud 25 04 2021].

[8] „Flyway Documentation,“ 2021. [Võrgumaterjal]. Available: https://flywaydb.org/documentation/. [Kasutatud 27 04 2021].

[9] „Flyway,“ 2021. [Võrgumaterjal]. Available: https://flywaydb.org/blog/organising-your-migrations. [Kasutatud 27 04 2021].

[10] „Docker docs,“ Docker, Inc., 2021. [Võrgumaterjal]. Available: https://docs.docker.com/get-started/overview/. [Kasutatud 27 04 2021].

[11] „Docker,“ Docker, Inc., 2021. [Võrgumaterjal]. Available: https://www.docker.com/. [Kasutatud 27 04 2021].

[12] J. Hamano, „Git,“ 2021. [Võrgumaterjal]. Available: https://git-scm.com/. [Kasutatud 27 04 2021].

[13] D. Zaporozhets ja V. Sizov, „GitLab,“ 2021. [Võrgumaterjal]. Available: https://about.gitlab.com/why/. [Kasutatud 25 04 2021].

[14] „GitLab,“ 2021. [Võrgumaterjal]. Available: https://about.gitlab.com/. [Kasutatud 27 04 2021].

[15] „GitLab docs,“ 2021. [Võrgumaterjal]. Available: https://docs.gitlab.com/ee/ci/introduction/index.html. [Kasutatud 27 04 2021].

[16] „GitLab CI/CD,“ 2021. [Võrgumaterjal]. Available: https://docs.gitlab.com/ee/ci/. [Kasutatud 27 04 2021].

[17] „Bootstrap (front-end framework),“ Wikipedia, 2021. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework). [Kasutatud 30 04 2021].

[18] „What is Bootstrap,“ w3schools, 2021. [Võrgumaterjal]. Available: https://www.w3schools.com/whatis/whatis_bootstrap.asp. [Kasutatud 30 04 2021].

[19] „What is Ubuntu?,“ Canonical Ltd., [Võrgumaterjal]. Available: https://help.ubuntu.com/lts/installation-guide/s390x/ch01s01.html. [Kasutatud 30 04 2021].

[20] „What is NGINX,“ F5, Inc., 2021. [Võrgumaterjal]. Available: https://www.nginx.com/resources/glossary/nginx/. [Kasutatud 30 04 2021].

[21] „Spring Web Services,“ VMware, Inc, 2021. [Võrgumaterjal]. Available: https://spring.io/projects/spring-ws. [Kasutatud 18 05 2021].

[22] „Spring Security,“ VMware, Inc., 2021. [Võrgumaterjal]. Available: https://spring.io/projects/spring-security. [Kasutatud 30 04 2021].

[23] „Spring Data JPA,“ VMware, Inc., 2021. [Võrgumaterjal]. Available: https://spring.io/projects/spring-data-jpa. [Kasutatud 30 04 2021].

[24] A. Parecki, „OAuth 2.0,“ 2021. [Võrgumaterjal]. Available: https://oauth.net/2/. [Kasutatud 30 04 2021].

[25] „JUnit,“ tutorialspoint, 2021. [Võrgumaterjal]. Available: https://www.tutorialspoint.com/junit/junit_overview.htm. [Kasutatud 30 04 2021].

[26] „Swagger,“ SmartBear Software, 2021. [Võrgumaterjal]. Available: https://swagger.io/. [Kasutatud 30 04 2021].

[27] „Swagger UI,“ SmartBear Software, 2021. [Võrgumaterjal]. Available: https://swagger.io/tools/swagger-ui/. [Kasutatud 30 04 2021].

[28] P. Kathiresan, „React,“ Medium Corporation, 05 09 2020. [Võrgumaterjal]. Available: https://pradeepakathiresan.medium.com/react-64e0c6859e41. [Kasutatud 02 05 2021].

[29] „Vue.js,“ 2021. [Võrgumaterjal]. Available: https://vuejs.org/v2/guide/. [Kasutatud 02 05 2021].

[30] J. Duffy ja J. Brown, „Aurelia: An Introduction,“ Code Magazine, 19 02 2019. [Võrgumaterjal]. Available: https://www.codemag.com/article/1607091/Aurelia-An-Introduction. [Kasutatud 02 05 2021].

[31] Meefirms, „ELI5: What in the heck is node.js?,“ Reddit, 2016. [Võrgumaterjal]. Available: https://www.reddit.com/r/learnjavascript/comments/3d4hs5/eli5_what_in_the_heck_is_nodejs/. [Kasutatud 02 05 2021].

[32] Vivek Balasubramaniam, „Baeldung,“ Baeldung, 7 12 2019. [Võrgumaterjal]. Available: https://www.baeldung.com/jpa-entities. [Kasutatud 17 5 2021].

[33] baeldung, „Baeldung,“ Baeldung, 6 9 2020. [Võrgumaterjal]. Available: https://www.baeldung.com/hibernate-identifiers. [Kasutatud 17 5 2021].

[34] O. G. T. R. J. B. M. H. Mark Pollack, „Chapter 4. JPA Repositories,“ %1 *Spring Data: Modern Data Access for Enterprise Java*, O'Reilly Media, Inc., 2012, p. 150.

[35] Microsoft, „Data Transfer Object,“ 17 3 2014. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649585(v=pandp.10)?redirectedfrom=MSDN.

[36] V. Seniuk, „Data Transfer Object Pattern in Java - Implementation and Mapping,“ Stackabuse, [Võrgumaterjal]. Available: https://stackabuse.com/data-transfer-object-pattern-in-java-implementation-and-mapping/. [Kasutatud 17 5 2021].

[37] „Tarkvara testimine,“ Wikipedia, [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Tarkvara_testimine#Integratsioonitestimine. [Kasutatud 5 17 2021].

[38] Flyway, „Documentation,“ Flywaydb, [Võrgumaterjal]. Available: https://flywaydb.org/documentation/. [Kasutatud 17 5 2021].

[39] Spring.io, „Spring.io,“ Spring.io, 2021. [Võrgumaterjal]. Available: https://spring.io/projects/spring-ws. [Kasutatud 17 05 2021].

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

Meie, Karl-Erik Hein, Artur Kerb ja Robin-Kevin Koppa

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose TalTechi seiklusmäng, mille juhendaja on Evelin Halling.
   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. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2021

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

Lisa 2 – Google Forms küsitluse vorm

Skoop

Ülesanne:
https://skoop.cs.taltech.ee/avaleht
1) Mine 'loo uus ruum' ning tee läbi tutorial
2) Tee uus mäng, vali mängutüübiks normal
2.1) Loo kaardistik 3 markeriga
2.2) Loo küsimustik kõigi 3 erinevat tüüpi küsimusega
2.3) Mine paaride leheküljele ning genereeri manuaalselt paarid eelnevalt tehtud küsimustiku ning kaardistikuga
2.4) Aktiveeri ruum
2.5) Proovi game master vaatesse minna
2.6) Proovi aktiveeritud mänguga liituda ning endale tiim koos liikmetega teha
2.7) Vastake mängus olevatele küsimustele

*Required

Kas tutorial aitas Teid? *

Your answer

Mis jäi arusaamatuks? *

Your answer

Kuidas saaks kasutajaliidest paremaks teha? *

Your answer

Kas mäng ise on funktsionaalne ning mängitav? *

Your answer

Mis üks asi Teile meeldis ning mis asi Teile ei meeldinud? *

Your answer

Submit