TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleksei Tsõpov 185088IAIB

Nikita Ojamäe 185827IAIB

Ilja Nikolski 185789IAIB

TalTech Quiz veebirakenduse arendamine

Bakalaureusetöö

Juhendaja: Evelin Halling

PhD

Tallinn 2021

Autorideklaratsioon

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

Autorid: Aleksei Tsõpov, Nikita Ojamäe, Ilja Nikolski

Annotatsioon

Selle bakalaureusetöö eesmärgiks on kirjeldada TalTech Quiz veebirakenduse arenduse protsessi. Antud rakendus oli tellitud Ago Lubergi poolt.

TalTech Quiz on analoogiks olemasolevale tuntud veebirakendusele Kahoot. Peamine põhjus analoogi arendamiseks on see, et Kahoot ei paku tervet funktsionaalsust tasuta versioonis. Piiratud funktsionaalsus takistab õppimisprotsessi, kuna viktoriinide loojad peavad arvestama nende piirangutega nagu näiteks küsimuse tüübi valimine. TalTech Quiz pakub kogu funktsionaalsust tasuta kõigile, kellel on olemas personaalne TalTech-i konto.

TalTech Quiz on süsteem, mis koosneb kahest osast. Esimene osa on veebirakendus, mille eesmärgiks on viktoriinide haldus nagu näiteks loomine, muutmine ja ka mängitud mängude statistika. Teine rakendus annab võimaluse mängida eelnevalt loodud viktoriine.

Mõlemad veebirakendused koosnevad omakorda tagarakendusest ja esirakendusest. Mõlemad tagarakendused põhinevad Spring Boot raamistikul, mis võimaldab standardsete teekide ja arhitektuuride kasutust. Esirakendused on arendatud kasutades React'i ja TypeScript'i. Iga rakenduse edastamine serverile toimub läbi automaatse CI/CD torujuhtme, mis asub GitLab repositooriumis.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 9 peatükki, 10 joonist.

Abstract

Development of TalTech Quiz web application

The aim of this bachelor's thesis is to describe the process of development of TalTech Quiz web application. This web application was commissioned by Ago Luberg.

TalTech Quiz is an analogue of another well-known web application which is Kahoot. The main reason for the development of a similar web application is that Kahoot does not provide all its functionality for free. This fact highly restricts the process of education because quiz creators have to take into consideration all these restrictions such as possible question types while composing a quiz. At the same time, TalTech Quiz offers all the functionality for free for all users with a personal TalTech account.

TalTech Quiz is a system that consists of two logical parts. The first part is a web application that is responsible for the administration of quizzes, such as creation, modification and statistics. The second one is a web application that allows you to play a previously created quiz.

Both web applications in its turn consist of backend and frontend servers. Both backend servers are based on Spring Boot which allows the use of standard libraries and architectural solutions. Both frontend servers are developed using React and TypeScript technological stack. Deployment of each application is conducted via automatic CI/CD pipeline, which is located in GitLab as well as Git repository.

The thesis is in Estonian and contains 35 pages of text, 9 chapters, 10 figures

Lühendite ja mõistete sõnastik

Tagarakendus	rakendus, mida lõppkasutaja ei kasuta otseselt. Vastutab andmetega manipulatsiooni eest
Esirakendus	rakendus, mida kasutab lõppkasutaja
POC	Proof Of Concept; tõestus, et idee või meetod töötavad
UI/UX	user interface/user experience; kasutajaliides ja kasutuskogemus
mobile first	rakendus peab olema eelkõige kasutatav nutitelefonis
RESTful	representational state transfer; tarkvaraarhitektuuri laad
API	Application Programming Interface; rakendusliides
ORM	Object Relational Mapping; tehnoloogia, mis seob objektorienteeritud programmeerimiskeeli andmebaasidega
CRUD	Create, Read; Update, Delete; loo, loe, uuenda, kustuta
CSRF	Cross-site request forgery; rünnaku tüüp
XSS	Cross-site scripting; murdskriptimine, rünnaku tüüp
DTO	Data Transfer Object; objekt, mis edastab andmeid süsteemi osade vahel
XML	Extensive Markup Language; laiendatav märgistuskeel
JWT	JSON Web Token; JSON veebitunnus
MVC	Model, View, Controller; mudel, vaade, kontroller
duplex	kahekordistatud
STOMP	Simple Text Oriented Protocol; lihtne tekstile orienteeruv sõnumite edastusprotokoll
JSON	JavaScript Object Notation; andmevahetusvorming

reverse proxy	pöördproksi
CORS	Cross-origin resource sharing
IntelliJ IDEA	arenduskeskkond
HTTP	HyperText Transfer Protocol; hüperteksti edastusprotokoll
Docker image	juhised, mida kasutatakse konteineri loomiseks
Docker konteiner	käitatav versioon pildist

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tallinna Tehnikaülikoolis kasutatakse aktiivselt mitmesuguseid programme ja rakendusi, mis aitavad muuta õppimisprotsessi interaktiivsemaks. Üheks heaks näiteks sellistest rakendustest on Kahoot, kuid kahjuks tihtipeale taolised rakendused seavad funktsionaalseid piiranguid õppejõudule, pakkudes laiendatud võimalusi vaid lisatasu eest. Piirangute näitena võib välja tuua seda, et Kahoodis on võimalik luua ainult teatud tüüpi küsimusi ning ka seda, kui palju inimesi võib korraga mängus osaleda. Nendel põhjustel võivad kasutajad kokku puutuda olukorraga, kus neile vajaliku mälumängu loomine on ilma tasulist versiooni omamata võimatu.

Antud diplomitöö raames seadsime endale järgmise eesmärgi: luua süsteem, mis pakub alternatiivi sellele, mida võimaldab Kahoot ning mis lubaks funktsionaalsust, mida Kahoodi tasuta versioonis pole.

Antud süsteemi põhiliseks eesmärgiks on võimaldada kasutajale luua ja muuta mälumänge ning võimaldada neid läbi viia ja saada statistikat mängitud viktoriinidest. Antud ülesanne ei ole triviaalne tarkvara arenduse poolelt, kuna eeldab mitme erineva mooduli lähedast koostööd. Üheks näiteks võib tuua seda, et mängu alustamine toimub rakenduses, mis vastutab mälumängude loomise eest, kuid mäng ise toimub teises rakenduses, mis juba vastutab mängu loogika eest.

Eraldi keerukust lisab mängu loogika, sest iga õppejõu tegevus peab edastama viivitamatut tagasisidet igale mängijale. Küsimuste ja vastuste variantide näitamine peab toimuma igal kasutajal üheaegselt, et kõik mängijad oleksid võrdsetes tingimustes. Suurim osa antud tööst on pühendatud sellele, kuidas täpselt toimub meie mängu loogika. Rõhk on pandud sellele osale meelega, kuna see osa on tehniliselt kõige keerulisem ja kriitilisem kasutajakogemuses.

Antud töö raames oli meil palju ruumi õppimiseks ja olemasolevate teadmiste süvendamiseks. Väga palju küsimusi oli tehnoloogiate kohta, mida kasutada, arvestades projekti spetsiifikat ning edasiarenduse võimalust järgnevates iteratsioonides. Otsustasime, et kasutame neid tehnoloogiaid, millega oleksid järgnevad tiimid kokku puutunud ning neil oleks mugav ja lihtne projekti edasi arendada. Sellel põhjusel

otsustasime, et tagarakenduse kirjutame Javas ja esirakenduse React + TypeScript abil. Samal ajal oli meil mitmeid küsimusi, millele vastused puudusid, sest meil polnud piisavalt teadmisi ja oskusi saavutatud eelnevate õpingute jooksul. Seepärast oli vaja läbi viia probleemi põhjalik analüüs ning leida sobivaid lahendusi. Nii otsustasime võtta kasutusele Webflux + RSocket-i selleks, et lahendada eelnevalt kirjeldatud viivitamatu tagasiside probleem, kus iga õppejõu tegevus peab jõudma koheselt kõikide mängijateni. Antud lahenduse realiseerimine osutus meile keeruliseks, kuna uued tehnoloogiad vajavad palju aega süvenemiseks.

Selleks, et tulevastel arendajatel oleks lihtsam orienteeruda meie projektis, dokumenteerisime ja visualiseerisime tähtsamaid tarkvara osi. Suurt rõhku pöörasime diagrammide joonistamisele ja erinevate kasutajate *flow* kirjeldamisele. Lisaks antud dokumendis esitatud tarkvara arenduse protsessi kirjeldusele võib suureks abiks olla ka meie *Wiki* leht Gitlab keskkonnas.

2 Ülesande püstitus

Käesoleva bakalaureusetöö raames jätkasime Tarkvaraarenduse meeskonnaprojekti raames loodud projekti Taltech Quiz arendamist.

2.1 Kliendipoolne ülesanne

Antud tarkvara projekt oli tellitud Tallinna Tehnikaülikooli informaatika eriala programmijuhi, Ago Lubergi poolt, keda edaspidi nimetame kliendiks.
Kliendil oli soov teha kaasaegset küsimustiku rakendust, et lihtsustada ja soodustada teadmiste omandamist ning kontrollida ainete käigus saadud teadmisi. Kohe projekti alguses oli määratud ka kindlad analoogid, mis olid tellija poolt juba pikalt kasutusel olnud ning funktsionaalsus, mida klient soovis loodavas tarkvaras näha. Kuna klient oli juba pikalt analoogseid rakendusi kasutanud, siis teadis ta üsna täpselt, mida ta saada tahab. Kokkuvõtvalt oli kliendipoolseks ülesandeks luua veebirakendus, mis võimaldab luua küsimustikke, kus on erinevad vastusevariandid ja küsimuste tüübid. Küsimustikke

saab luua TalTechi õppejõud, üliõpilane või muu töötaja, kellel on olemas TalTech UNI-ID. Loodud küsimustikke saab mängida autentimata kasutajatega reaalajas.

2.2 Olemasolev rakendus

Tarkvaraarenduse meeskonnaprojekti aines sai rakendus valmis ainult „idee tõendamise tasemel (*poc*)“. Rakendus oli loodud kasutades Spring Boot, React ja WebFlux tehnoloogiaid. Bakalaureusetöö alustamisel rakenduse võimalused piirdusid sellega, et oli võimalik mängida läbi eelnevalt SQLi abil ettevalmistatud mäng ja näha selle tulemusi. Kuna rakendus oli tehtud POC tasemel, oli rakenduse kasutajakogemus ja stabiilsus minimaalne. Realiseeritud oli lihtsamal kujul küsimustikuga liitumine, küsimustiku küsimuste vastamine, konstantse punktide skoori jagamine ning võitjate kuvamine. Antud süsteemi töös esines ka tõrkeid, kuna see polnud täielikult läbi testitud.

2.3 Töö eesmärgid

Töö eesmärgiks oli saada töökindel ja funktsioneeriv tarkvara, mida saab kasutada ülikooli õppetegevuses ning mida oleks mugav kasutada kõigi ülikooli liikmete poolt. Põhiline funktsionaalsus on kasutajaloodena välja toodud, kus on kaks põhilist osatäitjat. Moderaator on kasutaja, kes on süsteemi sisse loginud ja saab küsimustiku mängu alustada ning seda hallata. Teine kasutaja on „külaline“ ehk mängija.

Moderaator saab:

Haldusliidesest:

- Sisse logida oma UNI-ID-ga
- Luua/muuta/kustutada küsimustikke
- Luua/muuta/kustutada küsimusi konkreetsele küsimustikule
- Valida küsimuste järjekorda
- Valida küsimuse tüüpi
- Valida vastusevariandid ja vastused vastavalt konkreetsele tüübile
- Valida punktide arvutamise strateegia
- Näha statistikat iga mängitud mängu kohta
- Näha statistikat iga mängija kohta

- Vaadata iga mängija detailseid tulemusi
- Kustutada statistikat
- Salvestada statistika ja tulemused PDF failina.

Mängu ajal:

- Küsida ja kuvada mänguga liitumise koodi
- Vaadata mänguga liitunud mängijate nimesid
- Visata mängija mängust välja
- Mängu alustada
- Lõpetada küsimustiku kuvamise
- Näidata vahestatistikat
- Panna järgmist küsimust
- Mängu lõpetada
- Vaadata mängutulemust

Mängija saab:

- Sisestada mängu liitumiskoodi ja mänguga liituda
- Näha vastuse variante
- Vastata küsimustele vastavalt küsimuse tüübile
- Näha tagasisidet

3 Projekti disain

Terviklik süsteem koosneb kahest osast: Public Application ning Back Office. Selles peatükis kirjeldatakse põhilisi tehnoloogiad, mida antud projektis kasutatakse.

3.1 Backoffice esirakendus

Esirakenduse tehnoloogiate valimisel oli eesmärgiks kasutada tänapäeva populaarsed tehnoloogiad, mida tunnevad enamus arendajad, ning mis on hästi dokumenteeritud ja laialdase kasutajaskonnaga. Lähtusime põhimõttest, et ka tulevastel arendajatel oleks võimalik ilma suurema sisseelamisperioodita projekti siseneda ja koodi kirjutada. Samuti oli meie jaoks äärmiselt oluline UI ja UX. Arvestasime, et rakendus tuleb luua *„mobile first“* põhimõttega.

3.1.1 ReactJS

On avatud lähtekoodiga JavaScripti teek kaasaegse ja paindliku kasutajaliidese loomiseks. React võimaldab luua keerukaid kasutajaliidese lahendusi, mis on parema hallatavuse eesmärgil jagatud väiksemateks koodiosadeks, mida nimetatakse komponentideks. React võimaldab kombineerida kasutajaliidest ka teiste teekide ja raamistikkudega. Samuti on Reacti jaoks arendatud kolmandate isikute poolt palju vabavaralisi komponente kasutajaliidese kiiremaks loomiseks [1], [2].

Kuna otsustasime järgida ühtset ja tänapäevase arenduse parimaid praktikaid, siis valisime esirakenduse kirjutamiseks funktsionaalse stiili, mis jagab terve koodi väiksemateks funktsioonideks. Antud koodi on palju lihtsam lugeda ja hallata.

3.1.2 Ant Design

Projekti kasutajaliidese ehitamisel otsustatasime võtta kasutusele Ant Design tehnoloogia. Tegemist on Reacti jaoks arendatud vabavaralise teegiga, mis pakub suure hulga valmis komponente, mida saab kiiresti stiliseerida ja kohandada vastavalt vajadustele. Samuti toetab Ant Design enamus kaasaegseid brausereid ja võimaldab märkimisväärselt kiirendada ning teha mugavamaks kasutajaliidese arendamist [3]. Komponentide stiilid on kaasaegsed ja sobivad ka väiksematele ekraanidele ja mobiilidele.

3.1.3 SPA printsiip

Antud projektis parema kasutajakogemuse loomiseks lähtusime SPA pritsiibist. Lühend SPA tuleb inglise keelsest *„single page application“*, mille põhimõtteks on teha kõik veebilehe muudatused tänu JavaScripti koodile, seejuures veebilehte mitte värskendades. Sellist printsiipi jälgides jääb kasutajale mulje, et tegemist on tervikliku ja sujuva veebirakendusega. React teegi kasutamine võimaldab SPA printsiibi rakendamist.

3.1.4 SASS

SASS on CSS preprotsessor, mida interpreteeritakse koodi kompileerimise käigus CSS'iks [4]. Antud projektis kasutasime SASS süntaksina SCSS, kuna see on uuem ja populaarsem tehnoloogia. SASS võimaldab teha komponentide stiliseerimise kiiremaks ja mugavamaks arendajatele, samuti on koodi kergem lugeda võrreldes klassikalise CSS'iga.

3.1.5 TypeScript

TypeScript on avatud lähtekoodiga keel, mis põhineb JavaScript'il ning võimaldab lisada deklaratiivselt tüüpe [5]. Tänu Typescript'i kasutusele tõuseb oluliselt projekti koodi loetavus ja veakindlus. Samuti elimineerib TypeScript väiksemaid vigu, mis võivad tekkida mitte rangelt tüübitud keeltega kirjutamisel. See tõstab oluliselt arendamise stabiilsust. Typescriptis kirjutatud kood transleeritakse Babeli abil brauserile arusaadavaks JavaScript'iks.

3.1.6 Redux

Redux on tehnoloogia, mis võimaldab hoida ja hallata JavaScript rakenduse globaalset olekut [6]. Tänu Redux'i kasutamisele võime palju lihtsamalt edastada andmeid lapskomponentidele, mis asuvad komponentide puus oma vanemaist mitme lapskomponendi võrra allpool. Tänu Redux'ile hoiame meie infot, kas kasutaja on rakendusse sisse loginud või mitte, ning iga komponent on vajadusel sellest teadlik.

3.1.7 React Query

React Query on React'i jaoks väljaarendatud teek, mis teeb kiireks ja mugavaks andmete pärimise, oleku muutmise ja serveri oleku vahemällu salvestamise (inglise keeles

caching) esirakenduses. Tänu sellele tehnoloogiale vähendame näiteks serveri koormust küsimustike pärimisel, kuna seda ei tehta enam igal pöördumisel vastava kasutajaliidese osa juurde. Samuti on see tehnoloogia hästi integreeritud React'iga ja võimaldab järgida arendamisel samat funktsionaalset stiili.

3.1.8 BEM

Block Element Modifier ehk BEM on HTML klasside nimetamiskonventsioon, mis võimaldab muuta esirakenduse koodi loetavamaks ja arusaadavamaks [7]. Tegemist on range nimetamise strateegiaga, mis aitab vältida stiilide kattuvusega seotud probleeme.

3.2 Backoffice tagarakendus

Backoffice on Spring Boot rakendus, mis kasutab Java 11. versiooni. See on rakendus, mis pakub RESTful teenuseid läbi API.

3.2.1 Spring Boot

Spring Boot on juba olemasoleva Spring raamistiku laiendus, millel on oma eripärad, mis annavad võimaluse luua eraldiseisvaid tootmisvalmis rakendusi, mida saab jooksutada ilma suurema konfiguratsioonita. Mõnedest eripäradest on näiteks: sisseehitatud veebiserver (nn. Tomcat), valikulised algsõltuvused ja XML konfiguratsiooni vajaduse puudumine. Spring Security, Spring Web MVC ja Spring Data JDBC on mõned näited nendest sõltuvustest, mis pakuvad võimsat infrastruktuuri rakenduse arendamiseks [8].

Sõltuvuste süstimine (*dependency injection*) on protsess, kus objekt määrab oma sõltuvusi ehk teisi objekte, millega ta töötab, kas läbi konstruktori argumentide, vabriku meetodi argumentide või välja määramisega peale objekti initsialiseerimist [9]. Sõltuvuste süstimiseks kasutatakse *Inversion of Control* konteinereid. Spring raamistikus on selleks konteineriks *ApplicationContext* liides, mis vastutab Java objektide initsialiseermise ja konfigureerimise eest, mida nimetatakse *bean*'iks. Konfigureerimine toimub läbi annotatsioonide, XML faili või Java koodis. Konfigureerime Java koodis on standartne viis, kuidas on sõltuvuste deklareerimine on realiseeritud antud projektis [10].

3.2.2 Spring Data JDBC

Antud rakenduses andmebaasiga suhtlemiseks kasutame Spring Data JDBC-d. Spring Data JDBC on lihtne, piiratud ORM süsteem, mis on osa Spring Data'st. See moodul pakub JDBC andmete kihile juurdepääsu tuge läbi CRUD (loo, loe, muuda, kustuta) operatsioonide. Erinevalt Spring Data JPA-st ei paku see „*lazy loading"-ut*, vahemällu salvestamist ega palju muud laiast Spring Data JPA funktsionaalsusest [8].

3.2.3 Azure OAuth2.0

Mitmesugused platvormid (Google, Facebook, Github, Microsoft) implementeerivad erinevalt OAuth 2.0 protokolli. Azure OAuth2.0 on teek, mis võimaldab seadistada kõike vajalikke läbi konfiguratsiooni faili, et kasutada Azure Active Directory autentimiseks [11].

3.2.4 Spring Security

Spring Security on raamistik, mis pakub valmis lahendusi autentimiseks ja autoriseerimiseks ning võimaldab seadistada kaitset mõnede ründevektorite eest (n.n. CSRF, XSS). Spring Security on *de-facto* standard Spring maailmas [12].

3.2.5 MapStruct

Mitmekihilistes rakendustes on tihtipeale vajalik kaardistada olemeid ja DTO. Antud koodi kirjutamine võib olla üksluine ja selle käigus on võimalik teha vigu. Selle vältimiseks kasutame oma rakenduses MapStruct teeki. Antud teek on koodigeneraator, mis lihtsustab olemite omavahelist kaardistamist, kasutades tavalist meetodite väljakutsumist, mille tõttu on kiire ja tüübi ohutu [13].

3.2.6 Flying-Saucer

Flying Saucer on XML/CSS renderer. Antud projektis kasutatakse seda teeki PDF dokumentide genereerimiseks. Arendaja peab ära määrama, milline on XML mall ning millised on CSS stiilid. Järgmiselt Flying Sauces loob dokumendi eelnevalt defineeritud formaadis [14].

3.2.7 Selenium

Selenium WebDriver on instrument brauseri automatiseerimiseks, mida kasutatakse brauseris kasutajate teostatavate tegevuste testimiseks. Meie rakenduses on kasutatud Seleniumit viktoriinide loomise, muutmise ja kustutamise testimiseks [15].

Testimise lihtsustamiseks on olemas Selenium IDE laiendus Google Chrome ja Mozilla Firefox brauseritele. Antud laiendus võimaldab salvestada automaatselt tegevusi brauseris ning on võimalus eksportida antud testid Java koodina.

3.2.8 JJWT

JJWT on teek, mis lihtsustab JWT genereerimist. Teek võimaldab kasutada erinevaid algoritme digitaalse allkirja genereerimiseks. Antud rakenduses on kasutusel RS512 asümmeetriline algoritm.

3.3 Public application esirakendus

Antud rakenduse osas on realiseeritud mängu kasutajaliides. Tehnoloogiate valik ja selle põhimõtted on analoogsed Backoffice esirakendusega. Ainuke erinevus on tingitud serveris (public application tagarakenduses) kasutatud tehnoloogia tõttu. RSocket tehnoloogia muudab klassikalise esirakenduse ja serveri suhtlemise viise.

3.3.1 RSocket

Tagamaks serveri ja kliendi püsiühendust, oli võetud kasutusele RSocket tehnoloogia. Kuna RSocket'i kasutamine erineb tavapärasest REST API põhimõttest, tuli kasutada esirankenduses RSocket'i spetsiifilisi teeke. Esialgu tuleb luua ühendus serveriga ning määrata ühenduses transpordikihi protokoll. Kui ühendus on määratud, oleme võimelised juba serverile erinevat tüüpi päringuid saatma ja ka serverist tulnud päringuid kuulama. Kõige rohkem erineb klassikalises mõistes klient-server rakendusest see funktsionaalsus, et klient masin kasutades RSocket'id saab olla kuulaja rollis kui on tellinud (*subscribe*) mõnele sündmusele. Selliste nö tellimuse loogika realiseerimiseks oli kasutusele võetud ka JavaScript reaktiivsete laienduste teek RxJS.

3.4 Public application tagarakendus

Tehnoloogilises mõistes public application tagarakendus pakub kõige suuremat väljakutset, kuna mõned tehnoloogiad, mida kasutatakse antud rakenduses, ei ole kaetud meie õppekavas ning nõuavad lisa teadmisi ja oskusi. Juba alguses tekkis meil küsimus, kuidas on võimalik luua süsteemi, mis saaks toetada tuhandeid kasutajaid korraga ilma kvaliteedi languseta. Probleem seisneb selles, et tavaline Spring Web MVC põhine rakendus loob iga päringu jaoks oma voo. Seda konkreetset voogu kasutatakse kõikide vajalike instruktsioonide täitmiseks ning ta täidab sealhulgas ka blokeeritavaid instruktsioone, nagu pöördumine andmebaasi poole. Kuna voogude arv on piiratud, siis suure koormuse all hakkab rakendus aeglasemalt päringuid töötlema, mis on kriitiline mängu kontekstis, seega hakkasime otsima alternatiivi.

Kirjeldatud probleem Spring maailmas ei ole uus ning suure koormuse haldamiseks kasutatakse tehnoloogiaid, mis kuuluvad *reactive-stack* koosseisu (*The reactive-stack web frameworks*). Spring Web MVC alternatiiviks on Spring WebFlux [16].

3.4.1 Spring WebFlux

See on veebi-raamistik, mis võimaldab asünkroonselt töödelda I/O operatsiooni, mida nimetatakse mitte-blokeerivaks (*non-blocking*). Põhiline erinevus seisneb selles, et igale päringule ei eraldata oma voogu, mis vastutab ainult selle konkreetse päringu eest. Asünkroonsete operatsioonide korral Spring WebFlux võimaldab suunata voo ressursse teiste vajalikke instruktsioonide täitmiseks, mis nõuavad tähelepanu just praegu. Kui asünkroonne operatsioon on täidetud ning on võimalik jätkata järgmiste instruktsioonide täitmist, siis eraldatakse vaba voog, mis hakkab vastutama edasiste instruktsioonide täitmise eest.

On oluline veel lisada, et Spring WebFlux võimaldab kirjutada mitte-blokeeritavat koodi tänu sellele, et kasutab teist raamistikku, mille nimi on Project Reactor [17]. See raamistik on aluseks mitte-blokeerivaks reaktiivseks programmeerimiseks Java virtuaalmasina jaoks. Samal ajal Project Reactor tugevalt muudab koodi kirjutamise stiili, kuna Project Reactor on mõeldud koodi kirjutamiseks funktsionaalses stiilis.

3.4.2 Spring Cloud Security

Antud projektis tavaline Spring Security on laiendatud teekidega, mis võimaldavad kirjutada koodi mitte-blokeeritavas stiilis. Selleks kasutatakse Spring Cloud Security, mis on teek erinevate autentimise ja autoriseerimise meetmetega, mida tavaline Spring Security ei paku.

3.4.3 RSocket

RSocket on rakenduse kihi protokoll, mis võimaldab *duplex* ühendust TCP, WebSocket või teise *byte stream* meetodi kaudu. Antud projektis me käsitleme RSocket protokolli WebSocket kontekstis.

Spring maailmas on võimalik kasutada WebSocket (transpordi kihi protokoll) ilma RSocket (rakenduse kihi protokoll) kasutamiseta. Probleem seisneb selles, et WebSocket protokoll ei kirjelda sõnumite formaati. Tavaliselt sellist probleemi lahendatakse nii, et kasutatakse sõnumi kirjeldatavat protokolli (n.n. STOMP) [18]. Meie otsustasime sellest lahendusest loobuda, kuna RSocket pakub sama funktsionaalsust ning kasutab JSON sõnumite formaadi, mis muudab lihtsamaks nii tagarakenduse kui ka esirakenduse arendamist.

Samal ajal RSocket protokolli implementatsioon Spring Boot baasil pakub mugavaid liideseid (*interface*), et seadistada erinevat tüüpi ühenduse teise osapoolega. Siin ei ole rohkem võimalik rääkida klient-server suhtlemisest, kuna mõlemad osapooled võivad olla nii kliendid kui ka serverid erinevatel ajahetkedel.

On oluline veel lisada, et RSocket kuulub mitte-blokeeritavat tehnoloogiate koosseisu ning on väga hästi integreeritud olemasoleva Spring Boot keskkonda.

3.4.4 Spring Data R2DBC

Antud rakenduses suhtlemine andemaasiga mängib olulist rolli ning peab töötama võimalikult efektiivselt. Spring Data Reactive Relational Database Connectivity (lühendatud R2DBC) võimaldab asünkroonselt pöörduda relatsioonilise andmebaasi poole.

Spring Data R2DBC on kontseptuaalselt lihtne ORM. Võrreldes Hibernate ORM-ga, Spring Data R2DBC ei paku vahemällu salvestamist, *lazy loading, write-behind*. Ühelt poolt võib selle funktsionaalsuse puudumine tunduda miinusena, kuid teisest küljest tehnoloogia lihtsus lubab kartmata ja ilma suure vaevata kasutada kõiki olemasolevaid võimalusi. Samal ajal Spring Data R2DBC on ainuke mitte-blokeeritav ORM, millest lähtub valiku puudumine [19].

3.4.5 Reactor Tests

Project Reactor pakub veel mugavat teeki, mis võimaldab testida mitte-blokeeritavat koodi nii ühik kui ka integratsiooni testide kaudu. Testide kirjutamine peab olema kooskõlas Project Reactor koodi stiiliga, mis on mõeldud funktsionaalseks stiiliks.

Reactor Tests töötab hästi ka teiste testimisteekitega, nagu JUnit 5 ning AssertJ.

3.5 CI/CD

Pidev integratsioon (CI) ja pidev edastamine (CD) on seotud tavade ja praktikatega, mis võimaldavad koodimuudatusi sagedamini ja usaldusväärsemalt edastada serverile. Tavaliselt CI/CD protsess koosneb mitmest etappidest: rakenduse loomine, pakkimine, testimine ning edastamine.

3.5.1 GitLab

CI/CD protsessis GitLab mängib väga olulist rolli, kuna antud projekti Git-repositoorium asub just GitLab-is. Samal ajal GitLab pakub mugavaid tööriistu, mis muudavad CI/CD seadistamist lihtsamaks. Üks olulisemaid tööriistasid on GitLab Runner, mis vastutab kõikide skriptide käivitamise eest CI/CD protsessi raames.

3.5.2 Docker

Docker on teenustoodete kogum, mis kasutab OS-taseme virtualiseerimist tarkvara pakkimiseks pakenditena kutsutavate pakettidena. Konteinereid kasutatakse nii lokaalselt (andmebaasi käivitamiseks), kui ka serveril. Serveril iga rakendus on eraldi pakitud oma konteinerisse. Konteineri loomine ning edastamine on automatiseeritud [20].

3.5.3 NGINX

NGINX on veebiserver. NGINX aitab seadistada Docker konteinereid, mis sisaldavad esirakendusi. Antud projektis NGINX kasutatakse ka *reverse proxy* rollis, et vältida probleeme, mis on seotud CORS mehhanismiga [21].

3.6 Andmebaas

Antud projektis on kasutusel avatud lähtekoodiga objekt-relatsiooniline andmebaasisüsteem PostgreSQL. Antud andmebaasisüsteem võimaldab andmete salvestamist, lugemist, muutmist ja kustutamist ehk kõiki CRUD operatsioone. Andmebaasi kirjeldamiseks kasutatakse tavalisi SQL lauseid PostgreSQL dialektis [22].

3.6.1 Liquibase

Andmemudelis muudatuste tegemiseks ning versioonide halduseks kasutatakse avatud lähtekoodiga teeki Liquibase. Muudatused toimuvad läbi *changeset*'i, kus igal on oma id ja autor. *Changeset* võib olla ühes neljast formaadist: SQL, JSON, XML ja YAML. Antud projektis kasutatakse SQL formaati [23].

4 Projekti kirjeldus

Projekti tegemisel lähtusime tarkvaraarenduse projektide headest tavadest, mis on kasutusel paljudes ettevõtetes. Kasutasime aktiivselt iteratiivseid arendusmetoodikaid ja Scrum'i.

4.1 Scrum

Selleks, et töö sujuks paremini, projekt saaks kiiremini valmis ja tulemus vastaks kliendi soovidele, kasutasime oma töös agiilseid arendusmetoodikaid ja Scrum metoodikat. Tegemist on ülesande lahendamise „raamistikuga", mis aitab lahendada keerulisi ülesandeid maksimaalse produktiivsusega ja luues seejuures suurima võimaliku väärtuse [24]. Antud metoodika oli valitud meie projekti mitmel põhjusel:

1. Scrum struktureerib arendusprotsessi ning on näha konkreetse panuse projekti peale iga sprindi lõpu.
2. Võimaldab kiiremini saavutada kliendi eesmärgi.
3. Võimaldab muuta arendusprotsessi kliendi jaoks läbipaistvamaks ja arusaadavamaks tänu iganädalasele demodele.
4. Võimaldab kõigile meeskonnaliikmetele olla töösse aktiivselt kaasatud ja omada teadlikkust kogu projekti ülesannetest ning kes konkreetse osa eest vastutab.

Meie projekti puhul oli kasutusel ühenädalased sprindid. Iga sprindi lõpus toimus väike retrospektiiv, nimelt räägiti ja demonstreeriti sprindi jooksul valminud projekti funktsionaalsust. Demonstreerimisel osales ka kliendi esindaja, kellel oli võimalus koheselt oma tagasiside anda. Kuna meie tiimi liikmete arvu tõttu pole võimalik määrata ainult ühele konkreetsele isikule scrum masteri rolli, oli järgmise sprindi planeerimine tehtud kollegiaalselt: arutamise käigus otsustasime, mis on järgmise sprindi skoobis ja kuidas neid ülesandeid jagada.

Kuigi Scrum'i kasutamine eeldab igapäevast tiimiliikmete kommunikeerimist väiksemate koosolekute näol või kirjalikus vormis, jäi see reegel meie projekti puhul täitmata, kuna puudus võimalus tegeleda projekti ülesandega igapäevaselt. Kuigi kompenseerisime seda omavahelise aktiivse suhtlemisega projekti teemadel kirjalikutes suhtluskanalites.

4.2 Tööjaotus

Kuna tegemist on väikese meeskonnaga, pole võimalik klassikaliselt rollide jaotust teha. Meie projekti puhul pidime jagama vastutust teemade kaupa mitte andma konkreetset rolli iga meeskonna liikmele. Vastutuse rollid olid jagatud järgmiselt:

1. Ilja Nikolski vastutas Backoffice esi- ja taga-rakenduse eest
2. Aleksei Tsõpov vastutas Public applicationi tagarakenduse, autentimise ja serveri eest.
3. Nikita Ojamäe vastutas nii Public kui ka Backoffice esirakenduste eest.

Antud rollid ei olnud rangelt määratud ja projekti käigus esines palju olukordi, kus tiimi liige tegi oma vastutusalast välist ülesanded.

Tööülesande jagamiseks, visualiseerimiseks ja jälgimiseks kasutasime Gitlabi tööriistu. Iga väiksem ülesanne on kirjeldatud Gitlabi piletis (inglise keeles *issue*). Iga pileti alla on kirjeldatud ülesande sisu ning selle ülesande planeeritud ja tegelikult kulutatud ajahulk. Ülesande piletite kirjeldus oli minimalistlik ja peegeldas võimalikult täpselt ja lühidalt, mis osa tuleb teha. Piletit, mille ülesandeks oli kasutajaliidese ülesehitamine oli juures ka pildid prototüübist. Samuti kasutasime Gitlabi keskkonnas silte (inglise keeles *label*) iga pileti juures. See võimaldas näha mis tüüpi ülesandega oli tegemist ja mis oli selle staatus ning anda arendajatele ja projekti jälgijatele ülevaade, mis tüüpi ja mis sisuga ülesanded on hetkel arenduses.

4.3 Koodi kvaliteet

4.3.1 Jagatud vastutus

Projekti koodi kirjutamisel oli eesmärgiks tagada koodi head kvaliteeti ja suhteliselt kerget loetavust. See aspekt on oluline nii meie meeskonna produktiivsemaks arendamiseks kui ka hilisema projekti edasiarendamise lihtsustamiseks. Antud projektis oli kasutusel jagatud vastutuse põhimõtte, mis oli realiseeritud läbi *code review*, mis on tarkvaraarenduses vastutuse jagamise võtmemehhanismiks. *Code review* on protsess, mis kujutab endast koodi analüüsimise ja uurimise teste arendajate poolt, mille eesmärgiks on tuvastada vigu või koodi nõrgemaid kohti, mis võivad hilisemates arendamise etappides probleeme tekitada. Selle põhimõte järgimiseks kasutasime Gitlabi keskkonnas *merge request-e*. Seadsime tiimisisese kokkulepe, kus iga muudatus koodis peab olema läbi vaadatud ja positiivselt hinnatud vähemalt ühe tiimiliikme poolt. Sellisel moel vastutab laivi mineva koodi eest mitte ainult selle autor, vaid ka kontrollija.

4.3.2 Testimine

Antud projektis koodi veakindluse ja kvaliteedi tagamiseks kasutasime ka tarkvara testimise. Peamisel on kasutusel kolm testimise võtted:

1. Ühik testimine on väiksemad testid, mis kontrollivad väikese koodiosa täitmise korrektsust [25].

2. Integratsiooni testid on testid, mis kontrollivad mingi kindla kasutusvoogu ja väiksemate koodiosade koostööd .
3. Automaattestid, mis imiteerivad päris kasutaja käitumist ning kontrollivad rakenduse kasutusvoogusid [26].

Lisaks oli testitud ka viidud läbi ka süsteemi koormus testid, et saada infot, mitu kasutajad suudavad maksimaalselt süsteemi kasutada. Testi tulemuseks oli 200 kasutajat. Kuid antud piirang tuleneb ainult serveri seadistustest ja võimsusest, kuhu on rakendus paigaldatud, kuna lokaalses testimiskeskkonnas oli see näitaja suurem.

4.4 Probleemide analüüs

Projekti arendamisel tuli teha palju jooksvat uurimistööd nii tehnoloogiate kui ka äriloogika osas. Probleemide mõistmiseks ja paremaks arusaamiseks kõigi meeskonnaliikmetele oleme kasutanud Gitlabi keskkonnas *Wiki* alljaotuse, kuhu oleme lisanud diagramme, kasutamisvoogude seletust ja andmemudelit. Lisaks sellele on GitLabis tehtud eraldi piletid, milleks on pandud silt *„research“*. Kus on peegeldatud jooksvate analüüside ja väiksemate uuringute tulemused.

Analüüsi tulemusena on valminud ka lahenduste prototüübid. Backoffice esirakenduse tegemisel lähtusime eelnevalt tehtud prototüübist, mis oli kliendile näidatud ja mitme iteratsiooni käigus valminud. Prototüüp on tehtud Figma veebikeskonnas.
https://gitlab.cs.ttu.ee/kahoot/back-office-front-end/-/wikis/Figma-prototype

Samuti on töö käigus valminud väiksemaid tarkvaraprojekte tehnoloogiate võimekuse katsetamiseks ja demonstreerimiseks. RSocketi tehnoloogia kasutava tagarakenduse ja esirakenduse ühenduse konfigureerimiseks on tehtud prototüübi formaadis näiteprojekt.
https://gitlab.cs.ttu.ee/kahoot/public-application-backend/-/wikis/85b80cfc-0738-4569-8ed4-95ddfad84876?random_title=true

5 TalTech Quiz

5.4 Süsteemi arhitektuur

Tarkvaraarenduse arhitektuuri üleseehitamisel on tänapäeval kasutusel kaks populaarset lahendust: monoliitne ning mikroteenuste arhitektuur.

Monoliitse arhitektuuri puhul töötavad kõik tarkvarakomponendid ühtse süsteemina. Selline süsteem on klassikalises mõistes suur protsess, mis suhtleb andmebaasiga [27]. Monoliitne süsteem on tavaliselt loodud mitme omavahel seotud ülesande täitmiseks. Sellised süsteemid on keerukad ning hõlmavad mitut omavahel seotud funktsiooni. Antud süsteemide koodibaas on suur ja muudatuste tegemine raske, kuna võib endaga kaasa tuua mitu soovimatut tagajärge.

Mikroteenuste arhitektuur vastukaaluks monoliitsele arhitektuurile koosneb eraldiseisvatest ja teineteisest sõltumatutest moodulitest, mis suhtlevad omavahel läbi võrguprotokolli [27]. Iga rakenduse osa täidab konkreetset ärieesmärki. Sellise lähenemise puhul on rakendust lihtsam skaleerida ja funktsionaalsust juurde kirjutada, seejuures ei pea muretsema kuidas see mõjutab ülejäänud süsteemi tööd. Mikroteenuste arhitektuuri kasutades on välistatud oht kogu rakenduse töövõimetusest, kui mingi konkreetne teenus on mingil põhjusel maas. Samuti on mikroteenuste süsteem tehnoloogiate suhtes agnostiline ehk tehnoloogiatest sõltumatu ja iga teenus võib kasutada erinevat tehnoloogiat äriloogika täitmisel.

Antud projektitöö arhitektuuri valimisel olid analüüsitud mõlemad lähenemised. Esialgse analüüsi tulemusena oli tarkvaka arhitektuur disainitud monoliitse süsteemina, kuid arendamisetappide alguses oli otsustatud jaga süsteemi äriloogika kihti kaheks sõltumatuteks osaks. Selline otsus oli tehtud parema laiendamise ja suurema töökindluse tagamiseks ning kergema funktsionaalsuse lisamise võimaldamiseks. Äriloogika järgi on projekt jagatud kaheks suureks osaks:

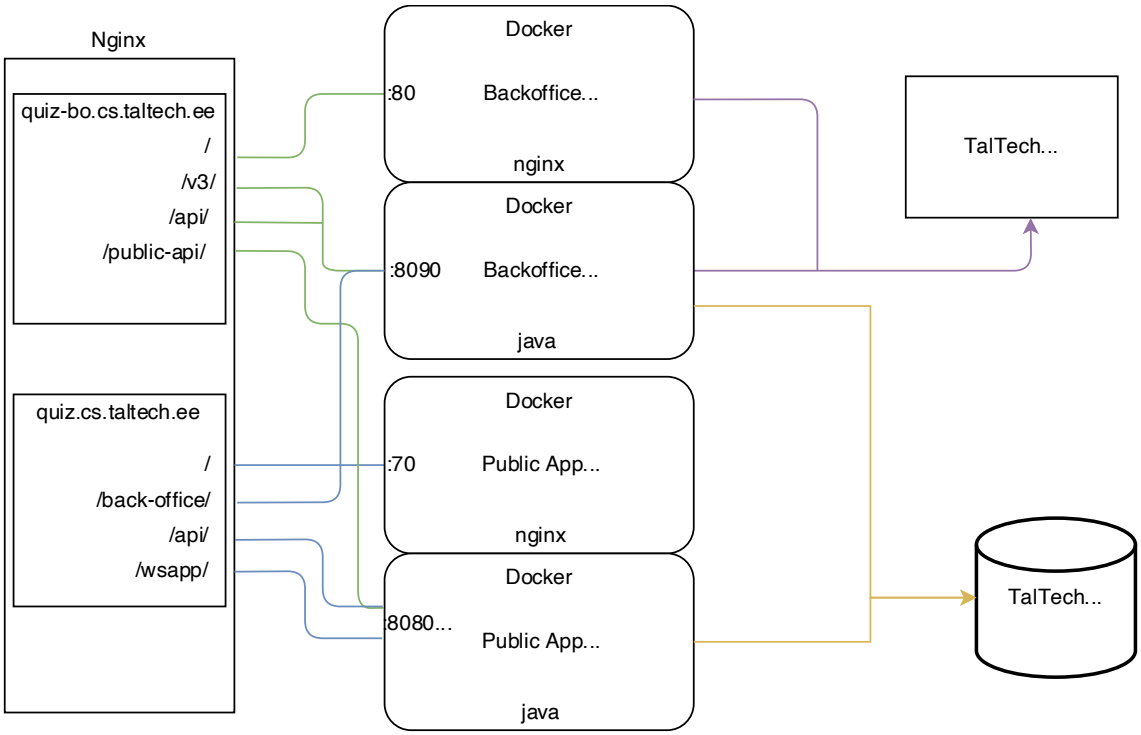
1. Public application – vastutab mänguloogika eest
2. Backoffice – vastutab küsimustike ja statistika haldamise eest

Tehniliselt koosneb tarkvaraprojekt neljast iseseisvast rakendusest:

1. Public application esirakendus
2. Public application tagarakendus
3. Backoffice esirakendus
4. Backoffice tagarakendus

Kuigi kõik neli rakendust on projekteeritud sellisel viisil, et kogu rakendus on ikkagi piiratud funktsionaalsusega töökõlblik, kui ühte teenust mingil põhjusel pole võimalik kasutada. Samas ei saa praegust projekti arhitektuuri nimetada täiesti mikroteenustel põhinevaks ega monoliitseks, kuna on olemas üks tsentraliseeriv instants. Kõik rakenduse osad kasutavad ühtset andmebaasi, mille töökõlbmatuks muutumisel muutub kasutuskõlbmatuks suurem osa projektist.

Projekti arhitektuur ei piirdu vaid meie loodud lahendustega, vaid kasutab ka välised teenuseid. Valitud disainimustrid võimaldasid implementeerida Azure Active Directory teenuseid, mis vastutab kasutajate autentimise eest (vt Joonis *1*. Süsteemi arhitektuur).



Joonis 1. Süsteemi arhitektuur

Nagu eelnevalt mainitud, pole võimalik vaadata iga rakenduse osa eraldiseisvalt.

Tervikliku kasutusvoo moodustamiseks peavad kõik rakenduse osad tegema omavahel sujuvat koostöö. Mängu alustamine on sellise koostöö kõige parem näide. Enne mängu alustamist peab kasutaja looma uut mängu haldusliidesest, mille eest vastutavad Backoffice tagarakendus ja Backoffice esirakendus. Mängu alustamiseks ja mängutoa loomiseks suhtleb Backoffice esirakendus juba Public application tagarakendusega ja kasutaja viiakse automaatselt Public application esirakendusele. Edaspidine tegevus toimub Public application esirakenduse ja Public application tagarakenduse vahel.

5.1.1 Arenduskeskkonna seadistamine ja arendamise lihtsus

Projekti arhitektuuri tegemisel oli olulisel kohal ka arenduskeskkonna võimalikult kiire ja lihtne ülesseadmine iga arendaja jaoks. Arendamisel oli kasutusel tarkvaramaailma ühte populaarsematest vahenditest IntelliJ IDEA, mis võimaldas rakendust kiiresti lokaalselt käivitada ja koodi siluda. Samuti võimaldab rakenduse arhitektuur käivitada igat rakendust omas Docker konteineris. Arenduse lihtsustamiseks on hoiame andmebaasi alati Docker konteineris ning tänu IntelliJ IDEA graafilisele kasutajaliidesele saab sellega kergesti interakteeruda.

5.1.2 Git

Tarkvara kood asub GitLab'i repositooriumides. Iga teenuse jaoks peaks olema eraldi repositoorium mugavamaks haldamiseks ja arendamiseks, kuid tagarakenduste eripärade tõttu oli otsustatud panna Public application tagarakendus ja Backoffice tagarakendus ühesse repositooriumisse, kuna nad jagavad ühist andmebaasi. Mõlema rakenduse esirakendused olid paigaldatud eri repositoorimitesse, et nende arendamine oleks teineteisest sõltumatu. Samuti on meil seadistatud CI/CD iga rakenduse jaoks eraldi.

5.5 Andmemudel

Andmemudel moodustab keskse osa igas projektis ning on väga oluline projekti mõistmiseks. See annab ülevaate sellest, mis olemitega tegutsetakse ja kuidas need olemid välja näevad. Tänapäevaste tarkvaraarenduse metodoloogiate järgi areneb iga osa iteratiivselt ning andmemudel ei ole erand. Aktuaalset andmemudelit (vt Lisa 2 – Andmemudel) on võimalik ka leida projekti GitLab Wiki-s. Mudeli visualiseerisime

Intellij IDEA abil, juhend, kuidas seda teha on samuti GitLab wiki-s olemas. Arengut saab vaadata projekti ajaloos.

Andmemudelit, nagu iga keerulist struktuuri, ei ole võimalik koheselt läbi mõelda, kuna laieneb rakenduse skoop ja selle tõttu tekivad uued detailid, mille peale alguses ei mõeldud. Antud projektis toimusid ka olulised muudatused andmemudelis. Parimaks näiteks sellest on tabelid, millel on prefiks *played_*. Tekkis vajadus nende järele, kuna arenduse käigus avastasime, et viktoriini kustutamisel või muutmisel, muutusid ka statistika vaates kuvatavad andmed. Vaatamata suurtele muudatustele proovisime teha terviklikku ja painduva süsteemi, mida oleks lihtne laiendada nii, et see ei rikuks üldist äriloogikat. Nii Backoffice kui ka Public application kasutavad samat andmemudelit, mis võimaldab lihtsat orienteerumist mõlemas rakenduses, kuna olemid on samasugused.

5.2.1 Viktoriin

Andmemudeli põhiliseks objektiks on viktoriin ehk *quiz*. Viktoriinil saab olla täpselt üks autor ning null või rohkem küsimust. Igal viktoriinil on:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- nimi - tekstiline väärtus;
- viide moderaatorile - välisvõti, mis viitab tabelis Author olevale väärtusele;
- viide esimesele küsimusele - välisvõti, mis viitab tabelis question olevale väärtusele. On vajalik selleks, et teada, mis küsimusest alustada mängu;
- viide pildile - välisvõti, mis viitab tabelis Image olevale väärtusele.

5.2.2 Küsimus

Küsimusi salvestatakse tabelisse question. Igal küsimusel on:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- pealkiri - tekstiline väärtus;
- küsimuse tekst - tekstiline väärtus;
- viide viktoriinile - välisvõti, mis viitab quiz tabelis olevale väärtusele;
- auhind - numbriline väärtus, mis määrab, palju selle küsimuse eest punkte saab;
- viide pildile - välisvõti, mis viitab Image tabelis olevale väärtusele;

- viide järgmisele küsimusele - välisvõti, mis viitab Question tabelis olevale väärtusele. Viidetest tekib linked list struktuur, kus viimasel küsimusel see väärtus puudub ehk on *null*;
- timer - numbriline väärtus, mis määrab ära selle, kui kaua antud küsimusele on aega vastata. Võimalikud väärtused on 15, 30, 45 ja 60 sekundit;
- Küsimuse tüüp - küsimused saavad olla kolme tüüpi:
    - SINGLE_MATCH - küsimusel on täpselt üks õige vastus. Punktide saamiseks peab valima õige vastuse;
    - MULTIPLE_MATCH - küsimusel on mitu õiget vastust. Punktide saamiseks peab valima kõik õiged vastused;
    - MULTIPLE_ANY - küsimusel on mitu õiget vastust. Mida rohkem õigeid vastuseid valitakse, seda rohkem saab punkte.
- Punktide andmise strateegia - strateegia on kahte tüüpi:
    - CONSTANT - punktisumma ei sõltu sellest, kui kiiresti küsimusele anti vastus;
    - FASTEST_ANSWER - punktisumma sõltub sellest, kui kiiresti vastati küsimusele.

Igal küsimusel saab olla kuni neli vastust, minimaalselt peab olema kaks vastust. Vastustel on:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- tekst - tekstiline väärtus;
- tõeväärtus - väärtus, mis näitab, kas antud vastus on õige või mitte;
- viide küsimusele - välisvõti, mis viitab question tabelis olevale väärtusele.

5.2.3 Kasutajad

Antud rakenduses autoriseeritud isikud ehk moderaatorid on salvestatud tabelisse nimega author. Antud kasutajad saavad luua viktoriine ning ka alustada mängu. Moderaatoril on:

- id - süsteemi poolt genereeritud identifikaator;
- OID identifikaator - tektstiline väärtus, mida saadakse Azure poolt;
- kasutajanimi - tekstiline väärtus, mis on kujul: uniid@ttu.ee.

Autoriseerimata kasutajad ehk mängijad on salvestatud tabelisse Player. Igal mängijal on:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- kasutajanimi - tekstiline väärtus, mille valib mängija ise;
- viide ruumile - välisvõti, mis viitab room tabelis olevale väärtusele.

5.2.4 Ruum

Viktoriini mängimiseks luuakse iga kord uus ruum, mida salvestatakse tabelisse Room. Ruumil on:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- pin - tekstiline väärtus, mille genereerib rakendus. Pin abil saavad mängijad liituda ja mängida;
- nimi - tekstiline väärtus, mille määrab moderaator
- hetkese küsimuse id - välisvõti, mis viitab Question tabelis olevale väärtusele. On vajalik selleks, et pidada järge, mis küsimuse juures hetkel ollakse;
- ajatempel - süsteemi poolt automaatselt loodud väärtus. Ajatempel näitab, millal ruum loodi;
- viide viktoriinile - välisvõti, mis viitab Quiz tabelis olevale väärtusele;
- viide salvestatud viktoriinile - välisvõti, mis viitab Played_quiz tabelis olevale väärtusele;
- status - igal ruumil võib olla erinev staatus sõltuvalt tema elutsüklist. Staatuse väärtused saavad olla:
    - REGISTERED - ruum on moderaatori poolt loodud, kuid pole veel valmis mängijate liitumiseks. Antud etapis toimuvad ettevalmistavad operatsioonid mängu alguseks;
    - OPEN - ruum on valmis mängijate liitumiseks;
    - READY - ruum on valmis mänguks. Mängijad ja moderaator saavad kogu vajaliku informatsiooni viktoriinist. Antud staatuses ruumiga enam liituda ei saa;
    - ANSWERING - on nähtav küsimus ja vastusevarianid. Mängijad saavad vastata.
    - REVIEWING - on nähtavad õiged vastused ja statistika. Mängijad vastata ei saa.
    - FINISHED - viktoriin on edukalt lõpetatud

- ABORTED - viktoriin ei lõppenud edukalt. Põhjusteks võib olla viga või moderaatori ühenduse katkestus.

5.2.5 Statistika

Mängu tulemuste hoidmiseks on viis tabelit. Kolmel tabelil on prefiks *played_* ning nendeks tabeliteks on Played_quiz, Played_question ja Played_answer. Antud tabelid on vajalikud selleks, et juhul kui moderaator soovib kustutada või muuta oma viktoriini, siis statikasse jääksid korrektsed muutumatud andmed esialgse viktoriini kohta. Teised kaks tabelit on Score ja Answer_frequency.

Tabelil Played_quiz on kolm välja:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- nimi - tekstiline väärtus, mis on sama nagu viktoriinil, mida mängitakse
- viide autorile - välisvõti, mis viitab tabelis author olevale väärtusele;

Tabelil Played_question on samad väljad nagu tabelil question, kuid on kolm erinevust:

- quiz_id - välisvõti, mis viitab tabelis Played_quiz olevale väärtusele;
- parent_question_id - välisvõti, mis viitab tabelis Question olevale väärtusele ehk küsimusele, mida kasutati antud viktoriinis;
- puudub väärtus next_question_id.

Tabelil Played_answer on samad väljad nagu tabelil Answer, kuid kahe erinevusega:

- question_id - välisvõti, mis viitab tabelis Played_question olevale väärtusele;
- parent_answer_id - välisvõti, mis viitab tabelis Answer olevale väärtusele ehk vastusele, mida kasutati antud küsimusel.

Tabel Score hoiab iga mängija punkte ning õigete ja valede vastuste arvu. Score tabelil on:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- correct_answers - numbriline väärtus, mis näitab, mitmele küsimusele andis mängija õige vastuse;
- wrong_answers - numbriline väärtus, mis näitab, mitmele küsimusele andis mängija vale vastuse;

- score - numbriline väärtus, mis näitab, kui palju sai mängija punkte kokku;
- viide mängijale - välisvõti, mis viitab tabelis Player olevale väärtusele;
- viide ruumile - välisvõti, mis viitab tabelis Room olevale väärtusele.

Tabel Answer_frequency annab ülevaate sellele, kui palju mängijaid valisid mingi vastuse. Answer_frequency tabelil on:

- id - unikaalne andmebaasi süsteemi poolt genereeritud identifikaator;
- viide ruumile - välisvõti, mis viitab tabelis room olevale väärtusele;
- viide vastusele - välisvõti, mis viitab tabelis Played_answer olevale väärtusele;
- viide küsimusele - välisvõti, mis viitab tabelis Played_question olevale väärtusele;
- frequency - numbriline väärtus, mis näitab, kui tihti antud vastusevarianti valiti.

5.6 Autoriseerimine

Tallinna Tehnikaülikoolis põhineb autoriseerimise süsteem Azure AD. Iga isik, kellel on personaalne kasutaja (uniid), saab autoriseerida läbi selle süsteemi. Suhtlus Azure AD-ga toimub läbi OAuth 2.0 protokolli. Azure AD kasutamine annab meile võimaluse luua paindliku autoriseerimise süsteemi, kus puudub vajadus hoida kasutajanimesid ja paroole meie andmebaasis, vaid selle eest vastutab Azure AD.

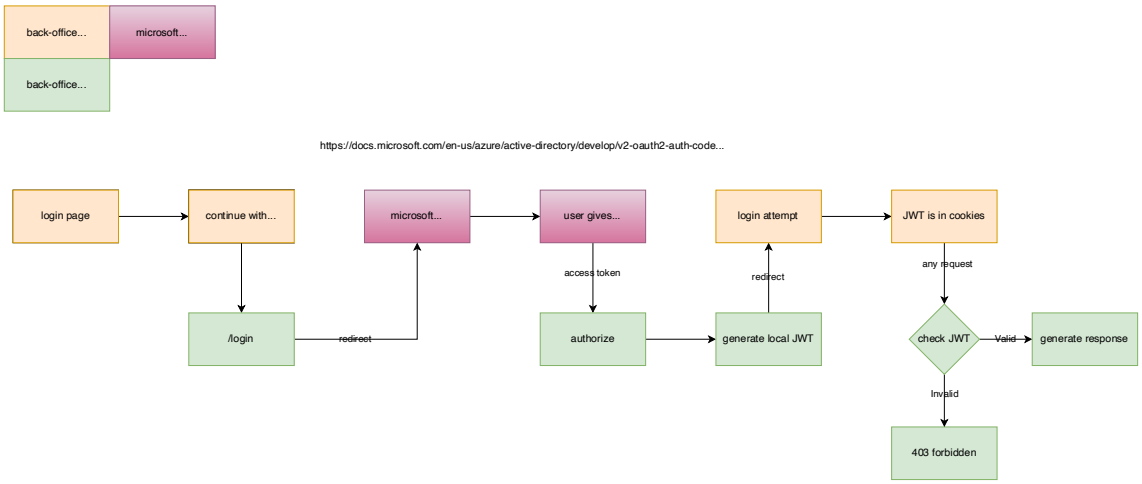
Backoffice tagarakenduse arenduse käigus oli meil vaja lisada meie rakendus Azure AD-sse, kuna see on paindlik platvorm, siis on lisamiseks mitu erinevat moodust, kuidas seda teha ning igaüks on mõeldud erinevat tüüpi rakenduste jaoks. Meie kasutasime OAuth 2.0 Authorization Code Flow, mille kohta võib leida informatsiooni ametlikus dokumentatsioonis

https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow.

Peale Azure AD seadistamist on võimalik Backoffice tagarakenduse ja Azure AD omavaheline suhtlus. Spring Security pakub valmislahendust, mis võtab enda peale kogu suhtluse Azure AD-ga. Selleks teegiks on spring-security-oauth2-client ning täpsemat informatsiooni võib leida ametlikus dokumentatsioonis

https://docs.spring.io/spring-security/site/docs/current/reference/html5/#oauth2

Eduka autoriseerimise järel genereerime enda JWT *token*-i JJW teegi abil selleks, et mitte kasutada iga kord Azure AD. Genereerimiseks kasutatakse avaliku - ja privaatvõtme paari, mida võib leida git repositooriumis *commons* moodulis. Antud võtmepaari kasutatakse ka JWT genereerimiseks Public application tagarakenduses. Genereerimine toimub OAuth 2.0 Authorization Code Flow lõpus. Kindluse mõttes luuakse kaks *token*-it, kus esimest kasutatakse autoriseerimiseks ja millel on lühike eluiga ning teist kasutatakse esimese värskendamiseks ja omad pikka eluiga. Mõlemat *token*-it asetame küpsistesse ja teeme ümbersuunamise Backoffice esirakendusele (vt Joonis 2. Autoriseerimise loogika diagramm).



Joonis 2. Autoriseerimise loogika diagramm

Selle tulemusena iga päring, mida teeb Backoffice esirakendus, sisaldab automaatselt küpsiseid. Tänu Spring Security filtrite süsteemile Backoffice tagarakenduses saame kontrollida *token*-ite valiidsust. Eduka autoriseemise korral on võimalik saada *token*-ist informatsiooni päringu tegijast. Näiteks on võimalik saada kasutaja id, mida hiljem on võimalik kasutada selleks, et leida kõiki viktoriine, mida on antud kasutaja teinud. Arvestades võimalikke laiendusi ja muudatusi tulevikus, lisasime *authorities*, mis võimaldab piirata kasutajale saadavat funktsionaalsust sõltuvast sellest, mis õigused on talle antud.

5.7 Backoffice

Backoffice on klassikalises mõistes veebirakendus, mis koosneb tagarakendusest ja esirakendusest, mille suhtlus käib läbi REST API. Backoffice kujutab endast küsimustike administreerimise rakendust, mille funktsionaalsus on jagatud kolmeks peamiseks osaks.

5.4.1 Viktoriinide haldus

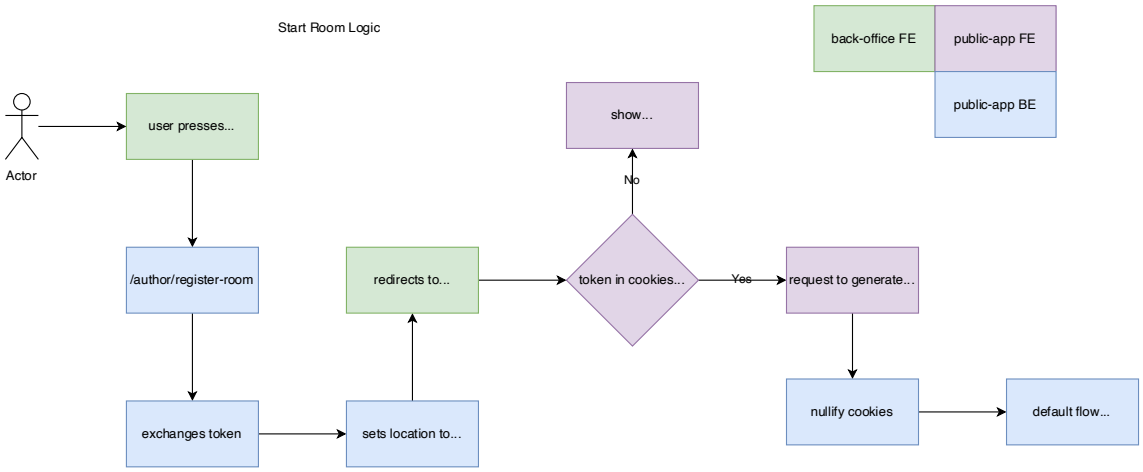
Üks Backoffice peamisest ülesandest on võimaldada mugavalt hallata kõik küsimustikega seotud protsesse. Selle funktsionaalsuse raames on kasutajal võimalik:

1. Luua/muuta/kustutada küsimustikke (vt Lisa 3 – Küsimustiku loomise kasutusvoo diagramm)
2. Luua/muuta/kustutada küsimusi konkreetsele küsimustikule
3. Valida küsimuste järjekorda
4. Valida küsimuse tüübi
5. Valida vastusevariandid ja vastused vastavalt konkreetsele tüübile
6. Lisada küsimustele ja viktoriinidele pilte.

(vt Lisa 4 – Tagarakenduse kasutajaliides)

5.4.2 Mängu alustamine

Backoffice võimaldab vaadata kasutaja poolt loodud küsimustikke ning sealt samast kasutajaliidesest mängu alustada. Mängu alustamisel sisestab kasutaja mängu toa nime ja teda suunatakse peamängu veebilehele (Public application), mis avaneb uues brauseri vaheaknas automaatselt. Mängu alustamisel algab mängu läbiviija (moderaatori) peamine kasutusvoog. Edaspidine tegevus toimub juba Public application raames, kuid kastutaja saab iga hetk mängu lõpetada ja pöörduda tagasi Backoffice vaheaknale (vt Joonis *3*. Mängu alustamise loogika diagramm).



Joonis 3. Mängu alustamise loogika diagramm

5.4.2 Statistika

Kui mäng oli edukalt lõppenud ning mängu läbiviija ei lõpetanud mängu varem kui süsteem kuvas ekraanile tulemuste tabeli, salvestatakse mängu statistika. Kasutajal on võimalik vaadata kõikide läbiviidud mängude loetelu tabelit, kus iga mängu juures on nupud, mis võimaldavad:

1. Vaadata mängu tulemuste statistika iga küsimuse lõikes eraldi, mis vastused olid õiged või valed ning palju inimesti on selle vastusevariandi valinud.
2. Vaadata statistikat iga mängija kohta eraldi. Mis olid selle mängija valikud ja tema tulemused
3. Laadida alla PDF faili, mis kuvab kogu statistikat

5.8 Public application

Public application koosneb kahest osast: esi -ja tagarakendus. Eristuvaks omapäraks antud rakenduses on RSocket protokoll, mille kaudu suhtlevad esirakendus ja tagarakendus. Muus mõttes on Public application klassikaline veebirakendus. Rakenduse paremaks kirjeldamiseks kirjeldame põhiliste osapoolte (mängu looja/moderator, mängija) kasutusvoogusid.

5.8.1 Moderaatori kasutusvoog

Moderaatori kasutusvoog saab alguse, kui toimub automaatne ümbersuunamine Backoffice rakendusest. Edaspidi saab moderaator:

1. Automaatselt genereerida mängu pin koodi, mida kasutajad peavad omakorda enda seadmetest vastavale tekstiväljale sisestada, et mänguga liituda.
2. Kuvada hetkel liitud mängijate nimed ekraanil ja vajadusel neid mängust välja visata klikkides vastava nimme peale.
3. Mängu alustada.
4. Iga hetk mängu lõpetada, kui selleks on vajadus. Kõik mängijad saavad vastava teade.
5. Kuvada küsimuse koos küsimuse pealkirjaga, pildiga, vastamiseks jäänud ajaga ning vastuste variantidega, mida saab olla maksimaalselt 4 ja minimaalselt Kusjuures küsimuste tüüp ei ole mängu ajal moderaatori ekraanil nähtav, seda näevad mängijad oma ekraanidel.
6. Lõpetada küsimusele vastamise vajutades nupule „*Skip*“.
7. Kui kõik pooled mängijad on küsimusele vastanud siis lühendab süsteem automaatselt järelejäänud aja hulka kümne sekundini. Seda kuvatakse vastaval taimeril.
8. Peale igat küsimust näha ning näidata õigeid ja valesid vastuseid
9. Peale igat küsimust näha ning näidata statistikat eelmise küsimuse kohta. Ekraanile on kuvatud maksimaalselt viis kasutajanime ja nende punkte, kellel on kõigest mängijatest kõige rohkem punkte.
10. Pärast kõikide küsimuste vastamist näidata terve mängu tulemuse. Süsteem kuvab kolme kõige rohkema punktide arvuga mängija nime punktide järgi kahanevalt.
11. Mängu lõpus salvestab süsteem mängu tulemusi, et hiljem kuvada neid Backoffice statistika lahtris.

Moderaatori kasutusvoo diagrammi saab näha lingilt.
https://gitlab.cs.ttu.ee/kahoot/public-application-backend/-/wikis/Author-Flow

5.8.2 Mängija kasutusvoog

1. Mängija sisestab moderaatori poolt genereeritud ja ekraanil ette näidatud mängutoa koodi, millele järgneb sellise toa olemasolu kontroll ja tekstivälja tekkimine kasutajanime sisestamiseks.
2. Mängija sisestab kasutajanime ja liitub mänguga. Mängija kasutajanime on näidatud moderaatori ekraanil.
3. Kui mäng on alanud, on mängija seadme ekraanile kuvatud vastusevariandid. Sõltuvalt küsimuse tüübist saab mängija kas valida ühe või mitu vastusevariandi. Kui küsimuse tüüp lubab mitut õiget vastust peab mängija valima kõik sobivad vastused ning hiljem oma valiku kinnitada vajutades vastavale nupule.
4. Kui moderaator on otsustanud küsimuse lõpetada või küsimusele mõeldud aeg on lõppenud, saab mängija näha oma vastusele tagasisidet ja vastuse eest saadud punktide arvu. Kui mängija jättis küsimusele vastamata saab mängija küsimuse eest null punkti ja vastust loetakse valeks.

5.8.3 RSocket

RSocket tagab esi -ja tagarakenduse vahelise suhtluse, kasutades transpordikihi protokollina mitte tavalist HTTP, vaid WebSocket Protocol. Selline lahendus annab võimaluse luua püsiühendus moderaatori ja mängija vahel läbi tagarakenduse serveri. Kasutades antud protokolle saame iga hetk edastada sõnumeid JSON formaadis. Seejuures sõnumeid saab edastada nii tavapärases mõistes kliendi pool (mängija/moderaator) kui ka serveri pool (tagarakendus). Kuna iga moderaatori tegevus terve mänguprotsessi jooksul peab olema koheselt peegeldatud mängija seadmes ja vastupidi, on selline funktsionaalsus meie rakenduse toimimiseks kriitilise tähtsusega ning eeldab serveril võimaluse saata mängijatele sõnumeid.

5.8.4 SSE

SSE on lühend inglise keelsest väljendist *Server Sent Event*. Tegemist on kontseptsiooniga, mis lubab serverile igal ajal kliendile sõnumeid saata. Tehniliselt on iga suundume tüübi jaoks olemas oma vastutav käitleja (ingilse keeles *handler*), millel on lihtne ja loogiline tööpõhimõte. Vaatleme näiteks toa staatuste muutmiste *handler*'i.

Antud käitleja luuakse serveris mängu alustamise hetkel. Mänguga liitumisel saadab liitunud mängija serveri päringu, millega tellib endale toa staatuse muudatusi. Kui moderaator muudab toa staatuse, saab *handler* sellest muudatusest teada ja koheselt edastab infot tellijatele. Meie rakenduses on lisaks veel mitu sellist tüüpi käitlejaid:

1. Vastutab tagasiside andmise eest
2. Vastutab kasutaja teavitamiseks kui mängija oli moderaatori poolt mängust eemaldatud.

(vt Lisa 5 - SSE diagramm)

5.9 CI/CD

Meie CI/CD süsteem on loodud sellisena, et arendaja saaks maksimaalselt mugavalt edastada muutusi serverile. Igal rakendusel neljast on torujuhe (ingl. *pipeline*), mis automaatselt kompileerib, testib ja edastab muutusi serverile.

5.9.1 **GitLab Runner**

See on rakendus GitLab platvormilt, mis võimaldab määrata CI/CD töid ning defineerida skripte, mida pannakse käima iga töö ajal.

Meie süsteemi raames käivitub GitLab Runner ainult siis, kui on muudatused *master* harus. Näitena vaatame CI/CD torustiku Backoffice tagarakenduse muutuste korral.

1. test - käivitatakse kõik testid, kui mõni neist ebaõnnestub, siis torustik lõpetad oma töö.
2. build - luuakse uut *Docker image* versiooni.
3. deploy - docker-compose abil käivitame konteineri serveril, mis on loodud uue docker image baasil.
4. post - kontrollime, kas on selliseid *Docker image*, mida enam ei kasutata, Kui sellised leiduvad, siis kustutame.

5.9.2 **Docker**

Meie süsteemis töötavad kõik rakendused Docker'is. See lihtsustab suhtlust rakendustega (kustutamine, loomine, peatamine, käivitamine). Docker image loomiseks kasutatakse Dockerfile. Igal rakendusel on see enda oma. Tagarakenduste käivitamiseks kasutatakse *docker-compose*, mis käivitab samal ajal Public App tagarakenduse, Backoffice tagarakenduse ja ka andmebaasi. Igat esirakendust käivitatakse eraldi.

Kogu süsteemi taaskäivitamise mugavuseks on torustikus kaks eraldi tööd: *restart_containers* ja *restart_and_reset_containers*. Esimene töö lihtsalt taaskäivitab kõiki rakendusi ja teine töö kustutab konteinereid ning käivitab uusi konteinereid. See võimaldab eemaldada kõiki kõrvalmõjusi, mis võisid tekkida nagu näiteks Liquibase Lock.

5.9.3 Nginx

Tähtsaks osaks rakenduse paigutamisest on NGINX seadistus. Meie rakenduses kasutatakse NGINX kui *reverse-proxy*'t, mis lahendab CORS probleemi erineva päritolu korral. NGINX seadistus on loogiliselt jagatud kaheks osaks: Backoffice ja Public application. Mõlemaid faile võib leida serveris /etc/nginx/sites-available kataloogis. Seadistus on standardne, kui välja arvata WebSocket ühenduse seadistust [28].

Samal ajal kasutatakse NGINX ka HTTPS ühenduse seadistamiseks. Kogu konfiguratsioon on ka eelpool mainitud kataloogis.

5.10 LTI

LTI - *Learning Tools Interoperability*. See on ulatuslik standardite kogum, mis kirjeldavad erinevate süsteemide integreerimise võimalusi haridusplatvormitega [29].

Meie skoobi raames puudus otsene ülesanne, kus pidime integreerima TalTech Quiz mingi haridusplatvormiga, kuid vaatamata sellele toimus esialgne analüüs selleks, et aidata tulevasi arendajaid sellise integratsiooniga näiteks moodle keskkonnaga.

5.10.1 Terminoloogia

Iga protokolli uurimine algab tutvusest domeeni mudeliga. LTI omab laialdast dokumentatsiooni, milles on lihtne ära eksida. Sellel põhjusel rühmitasime põhilisi mõisteid meie wiki lehel:

https://gitlab.cs.ttu.ee/kahoot/public-application-backend/-/wikis/LTI-terminology

5.10.2 ChemVantage registreerimine

Kõige esimeseks etapiks TalTech Quiz integreerimisel peab olema registreerimise protsess. Meie tiim uuris seda protsessi ChemVantage registreerimise näitel Schoolaby platformil, millega aitas meid Net Group.

Registreerimise detailset kirjeldust on võimalik leida meie wiki lehel:

https://gitlab.cs.ttu.ee/kahoot/public-application-backend/-/wikis/ChemVantage-registration

5.10.3 Launch Flow

Peale edukat registreerimist platvormil on võimalik käivitada rakendust läbi selle platvormi. Käivitamine toimub mitmest sammust. Diplomitöö raames analüüsisime esimest sammu ning koostasive kasutusvoo diagrammi, mida on võimalik leida wiki lehel:

https://gitlab.cs.ttu.ee/kahoot/public-application-backend/-/wikis/Launch-Flow

6 Valideerimine

Terve semestri jooksul saime iganädalaselt tagasisidet kliendi poolt. Arutasime kliendiga üldist pilti ja rakenduse arengut, kuid ei laskunud tehniliste detailideni. Vastavalt kliendi tagasisidele ja kommentaaridele tegime mitmeid muudatusi rakenduses. Loogiliselt võime jagada kaheks etapiks: semestri esimeseks ja teiseks pool.

6.1 Semestri esimene pool

Semestri esimesel poolel saime tagasisidet Backoffice esirakenduse prototüübi kohta Figma keskkonnas. Selle põhjal töötasime ümber disaini nii, et see vastaks Moodle keskkonna stiilile. Muutus vasakpoolne menüü paneel ning ka mobiilne vaade.

Samuti saime tagasisidet ka Public application'i kohta, mis koosnes sellest, kuidas jagatakse punkte mängijatele. Eelnevalt saadi alati sama arv punkte vaatamata sellele, kui kiiresti vastati küsimusele. Klient soovis, et oleks ka võimalus anda punkte sõltuvalt vastuse andmise kiirusest. Muutus ka punktide salvestamine nii, et see enam ei kustuks.

Lisaks sellele saime ka tagasisidet rakenduse stabiilsusele kohta. Seadistasime NGINX nii, et sessioon kestaks kauem ning muutsime marsruutimise *proxy*-ks. Toimus ka SSE loogika refaktoreerimine stabiilsuse saavutamise eesmärgina.

6.2 Semestri teine pool

Semestri teisel poolel oli kõige rohkem tagasisidet Backoffice kohta. Klient andis teada vigade kohta, mis tulid tal ette rakenduse kasutamise ajal. Oli ka palju tagasisidet kasutajakogemusest ja funktsionaalsusest, mida võiks kas lisada või muuta. Lisasime võimaluse muuta küsimuste järjekorda nende lohistamisega õigesse kohta. Tekkis võimalus panna küsimuse auhinnaks ka null punkti ning eemaldasime ühe küsimuste tüübi. Tagasiside tulemusena muutsime piltide salvestamist nii, et need skaleeruksid õigesse suurusesse, mitte ei kärbita neid.

7 Tulemused

Arendasime eelmise semestril alustatud rakendust minimaalselt elujõulise tooteni. Selle semestri jooksul tehtud muudatusi võib rühmitada kahte suurde kategooriasse: olemasoleva funktsionaalsuse paremaks muutmine ja uue funktsionaalsuse lisamine.

7.1 Muudatused

Muudatusi tuli teha palju, kuna eelmise semestri alguses ei suutnud ette näha kõiki võimalikke probleeme. Eesmärgiks oli muuta rakendus stabiilsemaks ning selle tõttu refaktoreerisime Public application tagarakenduse SSE ja parandasime leitud vigu. Public application esirakenduses toimusid mitmed muutused, mis olid seotud lehe kujundusega ja mängu loogika vigadega. Lisaks sellele vaatasime üle ka seda, kuidas mängu alustamise loogikat. Backoffice tagarakenduses toimusid ka mitmed muutused, mis olid seotud autoriseerimisega ja viktoriinide loomise loogikaga ja vigade parandusega. Muutusi tuli ette ka NGINX ja Swagger seadistustes.

7.2 Uus funktsionaalsus

Kõige rohkem uut funktsionaalsust sai Backoffice esirakendus, kuna eelmise semestri lõpuks oli loodud vaid tühi projekt. Oli vajalik luua kasutajaliides viktoriinide haldamiseks ja statistika vaatamiseks. Statistika jaoks lisasime funktsionaalsuse PDF raportite genereerimiseks Backoffice tagarakenduses. Lisaks sellele oli vajadus lisada tabeleid andmemudelisse, kuna eelnev lahendus ei saanud garanteerida korrektset statistikat, kui toimusid mingid muutused viktoriiniga nagu näiteks küsimuse kustutamine. Vastavaid muudatusi oli vaja läbi viia ka nii Backoffice tagarakenduses kui ka Public application tagarakenduses. Samuti toimus integratsioon Backoffice esirakenduse ja Public application tagarakendusega. Public application esirakendusse lisasime võimaluse näidata pilte ning näha, kui palju mängijaid vastasid jooksvale küsimusele. Public application tagarakendusse lisasime algrotimi, mis võimaldab anda punkte vastavalt sellele, kui kiiresti mängija vastab küsimusele.

7.3 Testimine

Nii Public application esirakendus kui ka Backoffice esirakendus olid testitud manuaalselt ning Backoffice esirakendus jaoks olid loodud Seleniumi automaattestid. Public application'i stabiilsuse kontrollimiseks tehti koormusteste. Backoffice funktsionaalsus on kaetud ühik - ja integratsiooni testidega. UI/UX muutus vastavalt kliendi kommentaaridele ja tagasisidele.

8 Järeldused ja edasised sammud

Antud projekti lõpuks sai implementeeritud, testitud ja valideeritud plaanitud funktsionaalsus ja kasutajaliides. Tulemusena on töötav rakendus. Kuna paljud tehnoloogilised lahendused ja kontseptsioonid olid väljastpoolt ülikooli programmi, tuli arendamise käigus teha suure hulga uurimistööd, mis on seotud jooksva arendamisega. Selle uurimistöö tulemusena saime palju uusi teadmisi, mida saime rakendada praktiliselt. Põhilisteks põhimõtteliselt uuteks teadmisteks olid:

1. Reaktiivne programmeerimine Javas (Reactive Spring)
2. NGINX serveri seadistamine
3. RSocket protokoll
4. Domeenidevaheline küpsiste käitlemine

8.1 Mis läks hästi?

Toome välja projekti arendamise ning projekti tulemuse iseloomustavad positiivsed punktid.

1. Scrum ideoloogiate kasutamine tõhustas projekti iteratiivset arengut.
2. Vastutusalad olid jagatud vastavalt meeskonnaliikmete varasemale kogemusele ja taustale ning igal liikmel oli enda vastutusala.
3. Kasutasime Gitlab *merge request'*e, mis võimaldas testele meeskonnaliikmetele jälgida ja kontrollida kõiki koodimuudatusi.
4. Kaasaegsete tehnoloogiate kasutamine (ReactJS, Spring Boot).
5. Seadistatud CI/CD, mis tegi mugavaks koodi tarnimise serverisse.

8.2 Mis võiks paremini minna?

1. Võiks olla mentor või suurema kogemusega arendaja, kes oskaks jagada arenduse paremaid praktikaid ning arendajate omavahelisi diskussioone koodi kvaliteedi osas lahendada.
2. Võiksid olla natukene selgemad kliendipoolsed ülesanded (Kliendipoolne detailne kirjalik analüüs oma soovidest).

3. Kuna projektis on kasutusel TalTech'i poolt pakutav veebiserver ja Microsoft Azure AD, oli ülikooli liikmeskonna vastutajate isikutega kontakteerumine raskendatud ning võttis väga kaua aega arenduse, tarne ja autentimise jaoks vajalike õiguste saamine.

8.3 Edaspidised sammud

Kõik edaspidised sammud on seotud olemasoleva funktsionaalsuse laiendamisega. Klient on juba avaldanud soovi näha järgmises iteratsioonides järgmist funktsionaalsust:

Public application:

1. Kasutajal on võimalik liituda mänguga jooksva mängu ajal.
2. Mängu moderaatoril on võimalus jooskvalt lisada küsimusele aega.

Backoffice:

1. Kuvada kasutajliideses, millal oli küsimustik viimati mängitud.
2. Kuvada kasutajaliideses, mitu korda oli küsimustik mängitud.
3. Küsimustiku autor saab oma küsimust teise moderaatoriga jagada.
4. Moderaator saatb genereerida csv faili mängude statistikaga.

9 Kokkuvõte

Diplomitöö raames on valminud terviklik lahendus kliendi probleemile. Projekt on funktsioneeriv, täidab püstitatud ülesandeid ning vajadusel on kergesti edasi arendatav. Bakalaureusetöö raames oleme kokku puutunud mitme erineva probleemiga: alustades

UI/UX ülesehitamisega lõpetades WebSocket ühenduse seadmisega läbi NGINX *reverse proxy*. Vahest oli probleemide lahendamine lihtne, kuid oli ka selliseid probleeme, millele lahendamiseks pidime mingi tehnoloogia dokumentatsiooni tervenisti läbi lugema.

Väga olulist rolli mängis meeskonnaliikmete vaheline suhtlus, mis võimaldas meil arutada ja leida lahendust probleemidele, valideerida lahendusi ja otsuseid, planeerida tegevust ja hoida motivatsiooni üleval.

Arendamise käigus juhtus tihti olukordi, kus oli väga raske näha tervet pilti, seepärast olime sunnitud juba töötavat lahendust pidevalt refaktoreerima. Refaktoreerimine on täiesti loomulik tarkvaraarenduse osa, eriti siis kui tegemist on kauakestva projektiga või suure koodi baasiga. Iga sellise tegevuse iteratsiooni tulemusena kood muutub paremaks ja funktsionaalsus stabiilsemaks. Scrum on väga hästi seda protsessi toetanud.

Ei ole olemas tarkvara, mis on 100% valmis. Selle diplomitöö raames oleme täitnud kõik püstitatud ülesanded ning oleme loonud väga hea aluse edaspidiseks arendamiseks. Antud projekt omab palju erinevaid arenguvõimalusi. Mõistame, et nullist projektiga alustada on lihtsam kui olemasolevat suurt koodibaasi edasi arendada, seega oleme visualiseerinud kasutusvoogusid diagrammide näol nii kasutajaliidese vaatepildid kui ka protsesside näol ning põhjalikult dokumenteerisime kõik peamised loogikaosad, et projekti edaspidise arendusega alustamine oleks minimaalse keerukusega.

Kasutatud kirjandus

[1] „ReactJS,“ [Võrgumaterjal]. Available: https://reactjs.org/. [Kasutatud 1 Mai 2021].

[2] „Tutorial: Intro to React,“ [Võrgumaterjal]. Available: https://reactjs.org/tutorial/tutorial.html. [Kasutatud 1 Mai 2021].

[3] „Ant Design of React,“ [Võrgumaterjal]. Available: https://ant.design/docs/react/introduce. [Kasutatud 1 Mai 2021].

[4] „Sass,“ [Võrgumaterjal]. Available: https://sass-lang.com/documentation. [Kasutatud 1 Mai 2021].

[5] „Typescript,“ [Võrgumaterjal]. Available: https://www.typescriptlang.org/. [Kasutatud 1 Mai 2021].

[6] „Getting Started With Redux,“ [Võrgumaterjal]. Available: https://redux.js.org/introduction/getting-started. [Kasutatud 4 Aprill 2021].

[7] „BEM,“ [Võrgumaterjal]. Available: http://getbem.com/. [Kasutatud 6 Aprill 2021].

[8] „Spring | Projects,“ [Võrgumaterjal]. Available: https://spring.io/projects/. [Kasutatud 29 Aprill 2021].

[9] L. Crusoveanu, „Intro to Inversion of Control and Dependency Injection with Spring,“ [Võrgumaterjal]. Available: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring. [Kasutatud 1 Mai 2021].

[10] „Core Technologies,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html. [Kasutatud 1 Mai 2021].

[11] „Microsoft identity platform and OAuth 2.0 authorization code flow,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow. [Kasutatud 14 February 2021].

[12] „Spring Security Reference,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-security/site/docs/current/reference/html5/. [Kasutatud 20 February 2021].

[13] „MapStruct,“ [Võrgumaterjal]. Available: https://mapstruct.org/. [Kasutatud 1 Mai 2021].

[14] „Flying Saucer,“ [Võrgumaterjal]. Available: https://flyingsaucerproject.github.io/flyingsaucer/r8/guide/users-guide-R8.html#xil_2. [Kasutatud 1 Mai 2021].

[15] „Selenium,“ [Võrgumaterjal]. Available: https://www.selenium.dev/. [Kasutatud 1 Mai 2021].

[16] „Web On Reactive Stack,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html. [Kasutatud 05 January 2021].

[17] „Reactor 3 Reference Guide,“ [Võrgumaterjal]. Available: https://projectreactor.io/docs/core/release/reference/. [Kasutatud 1 January 20 2021].

[18] „Stomp Over WebSocket,“ [Võrgumaterjal]. Available: http://jmesnil.net/stomp-websocket/doc/. [Kasutatud 08 January 2021].

[19] M. Smeets, „Spring: Blocking vs non-blocking: R2DBC vs JDBC and WebFlux vs Web MVC,“ [Võrgumaterjal]. Available: Spring: Blocking vs non-blocking: R2DBC vs JDBC and WebFlux vs Web MVC. [Kasutatud 18 February 2021].

[20] „Docker,“ [Võrgumaterjal]. Available: https://docs.docker.com/engine/reference/builder/. [Kasutatud 05 March 2021].

[21] „NGINX Reverse Proxy,“ [Võrgumaterjal]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/. [Kasutatud 08 March 2021].

[22] „PostgreSQL,“ [Võrgumaterjal]. Available: https://www.postgresql.org/. [Kasutatud 1 Mai 2021].

[23] „Liquibase,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Liquibase. [Kasutatud 2 Mai 2021].

[24] T. d. C. F. H. A. X. C. Igor Ribeiro Lima, „Adapting and Using Scrum in a Software,“ [Võrgumaterjal]. Available: http://www.fsma.edu.br/si/edicao9/FSMA_SI_2012_1_Principal_2_en.pdf. [Kasutatud 20 Aprill 2021].

[25] C. Tudose, *JUnit in Action, Third Edition*, Simon and Schuster, 2020.

[26] D. Molina, *Selenium Fundamentals*, Packt Publishing Ltd, 2018.

[27] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*, "O'Reilly Media, Inc, 2019.

[28] „NGINX as a WebSocket Proxy,“ [Võrgumaterjal]. Available: https://www.nginx.com/blog/websocket-nginx/. [Kasutatud 01 April 2021].

[29] „Learning Tools Interoperability,“ [Võrgumaterjal]. Available: https://www.imsglobal.org/activity/learning-tools-interoperability. [Kasutatud 1 Mai 2021].

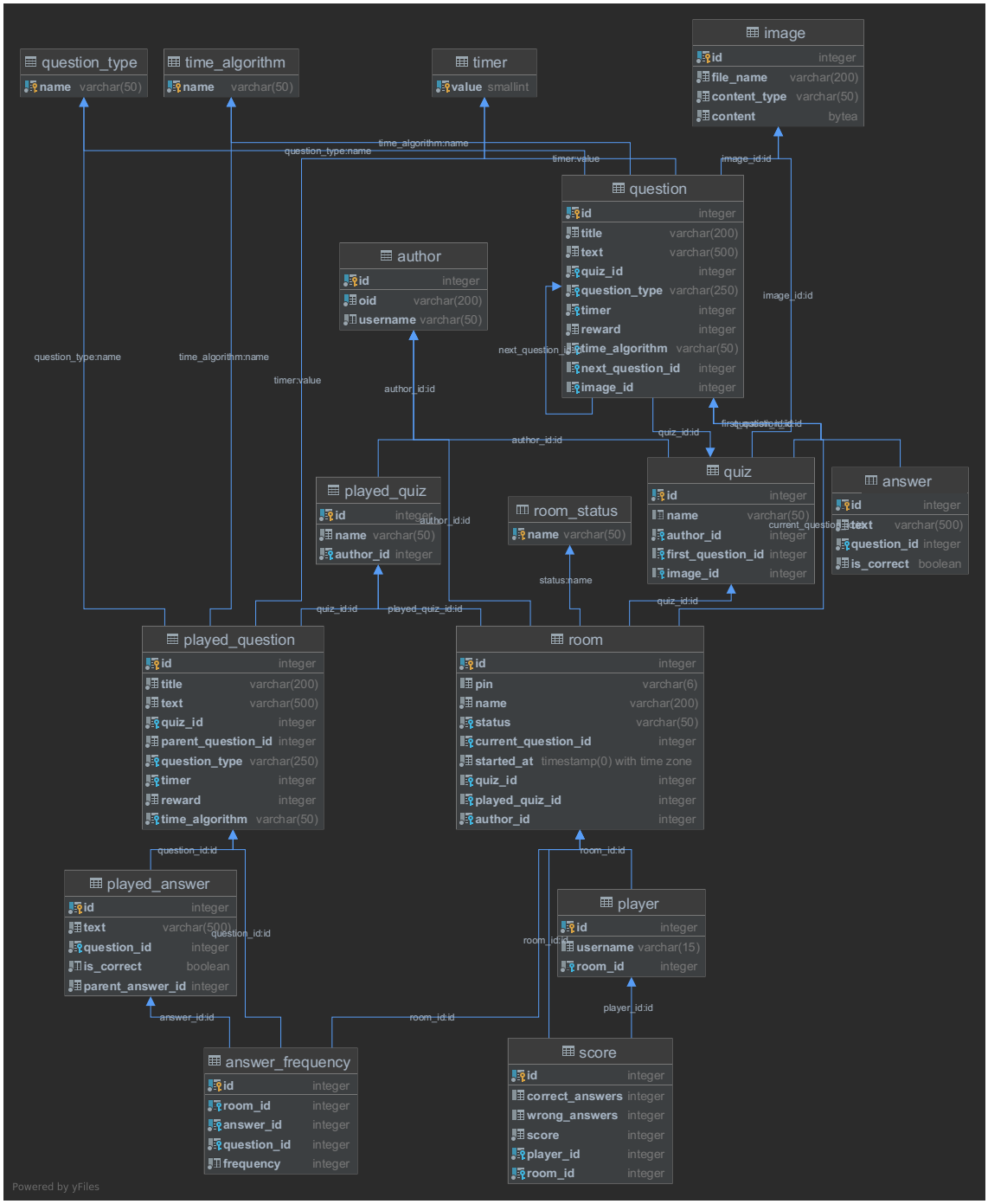
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Meie, Aleksei Tsõpov, Nikita Ojamäe ja Ilja Nikolski

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose , 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 autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

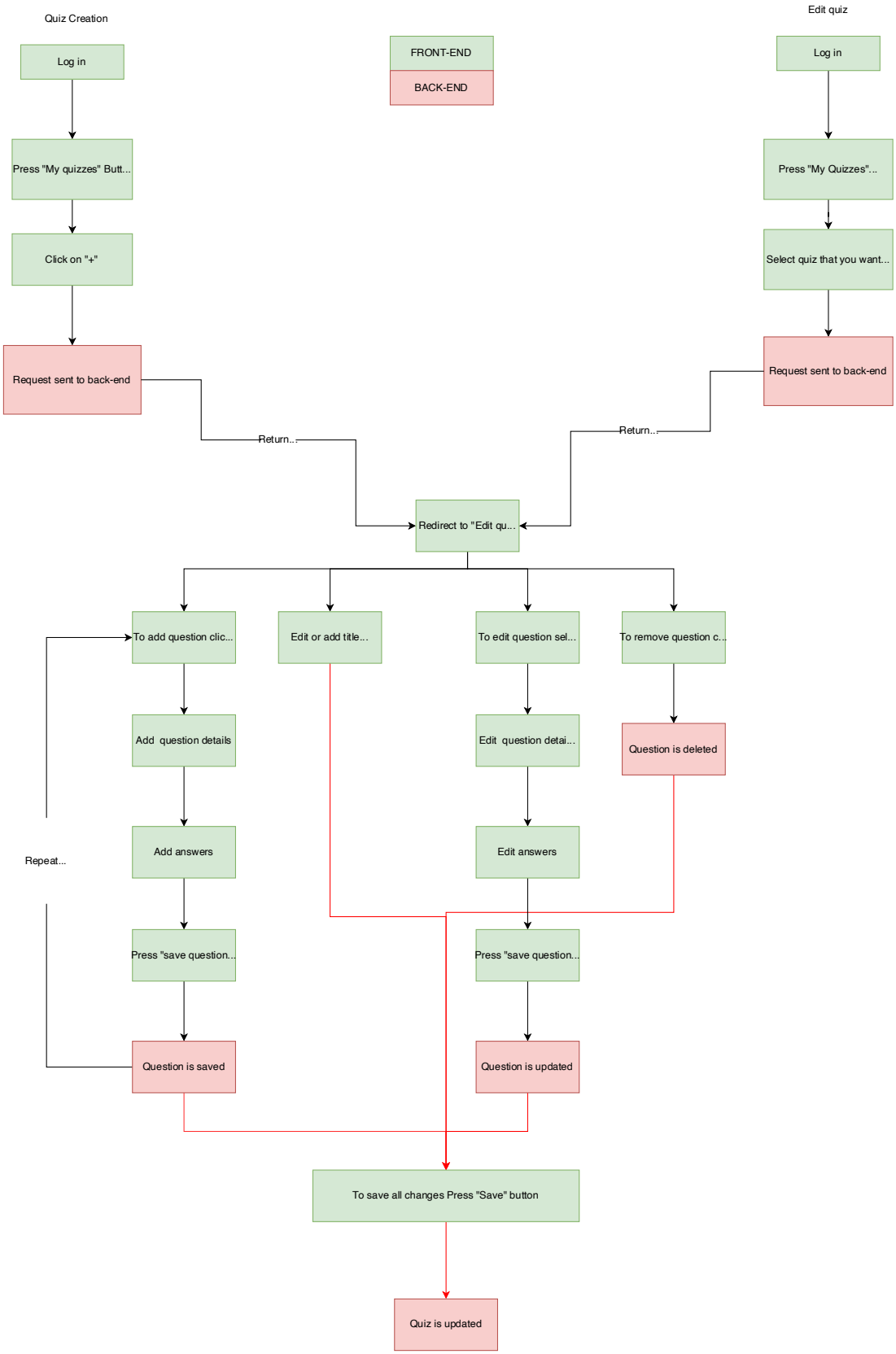
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või 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 – Andmemudel



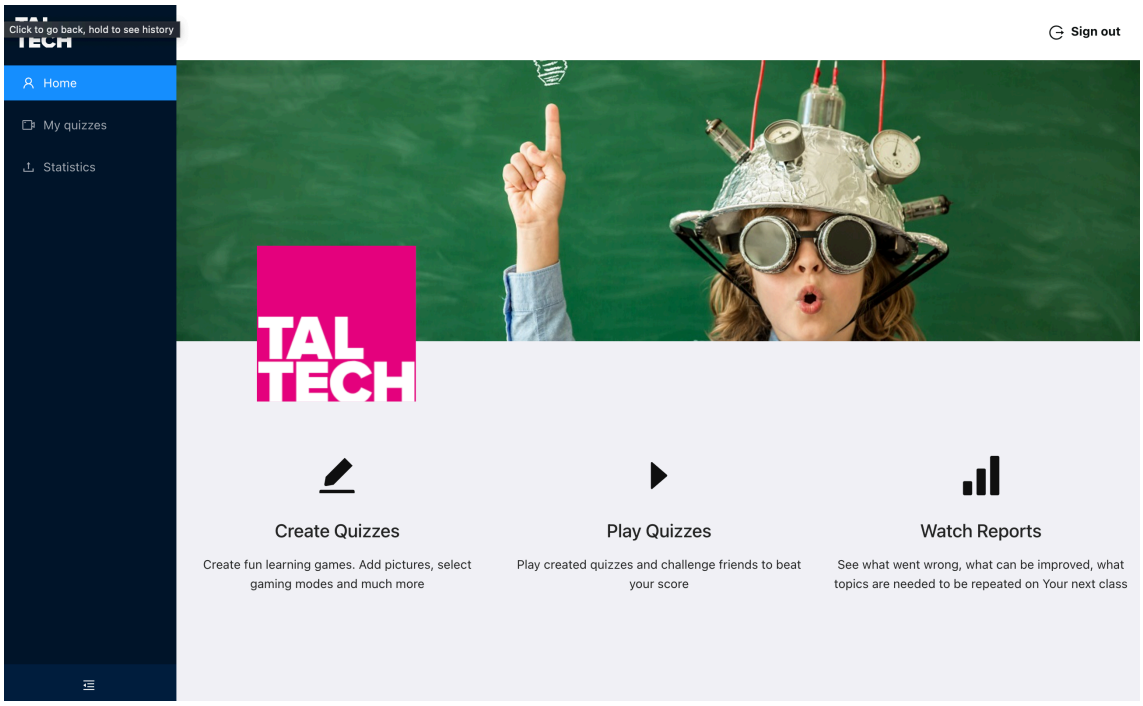
Joonis 4. Andmemudel

Lisa 3 – Küsimustiku loomise kasutusvoo diagramm

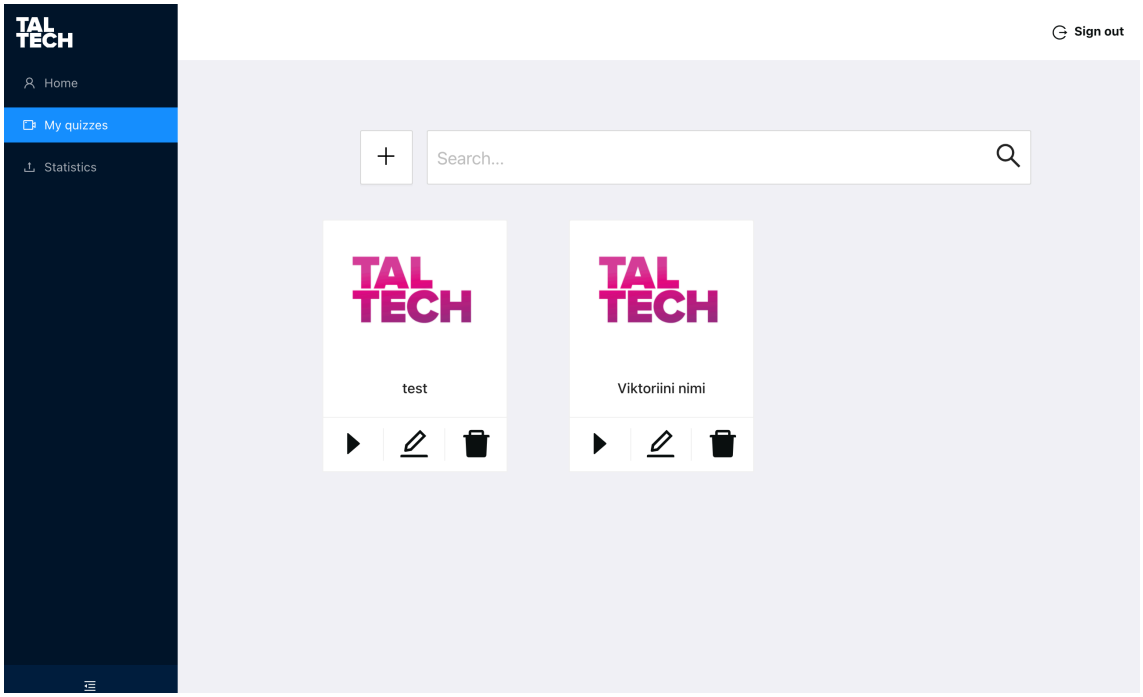


Joonis 5. Küsimustiku loomise kasutusvoo diagramm

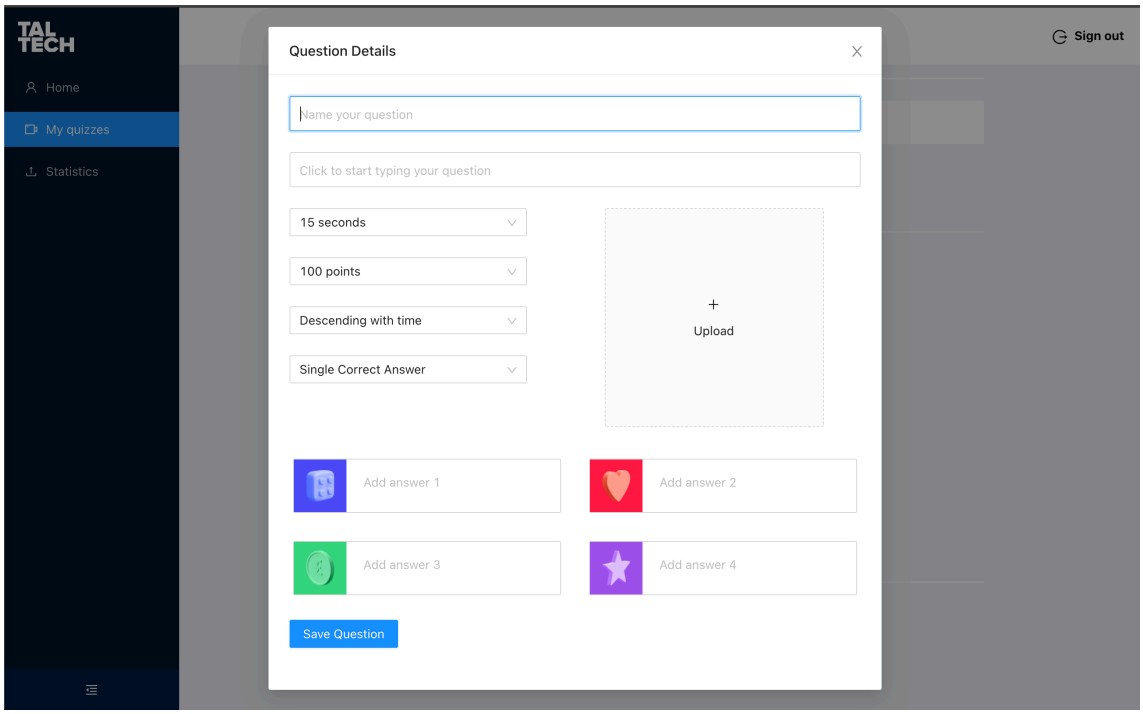
Lisa 4 – Tagarakenduse kasutajaliides



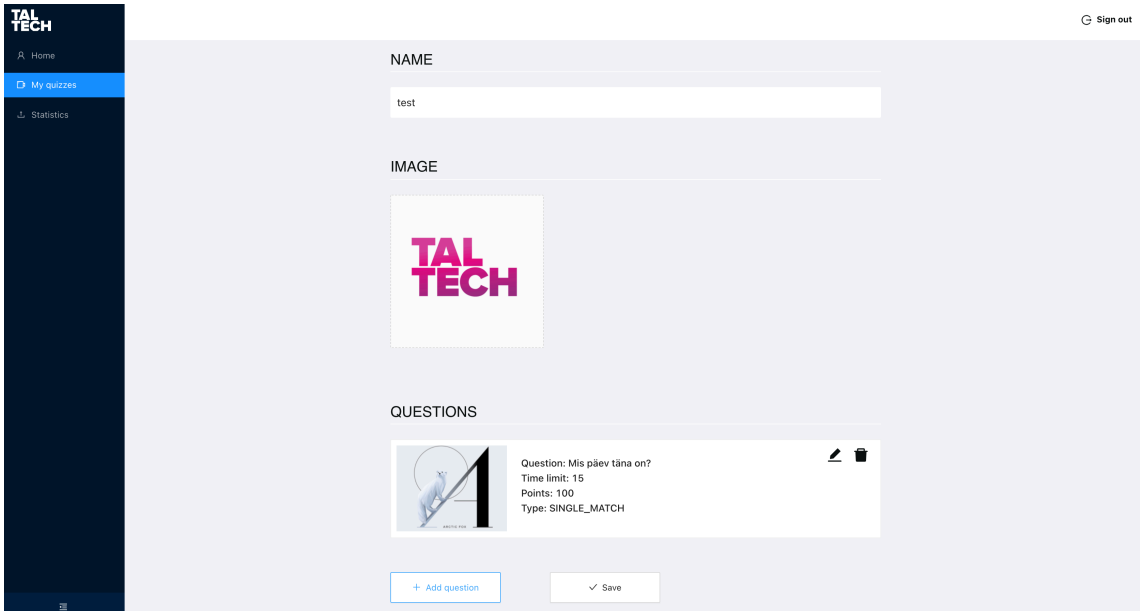
Joonis 6. Tagarakenduse avaleht



Joonis 7. Tagarakenduse kõikide kasutaja viktoriinide vaade

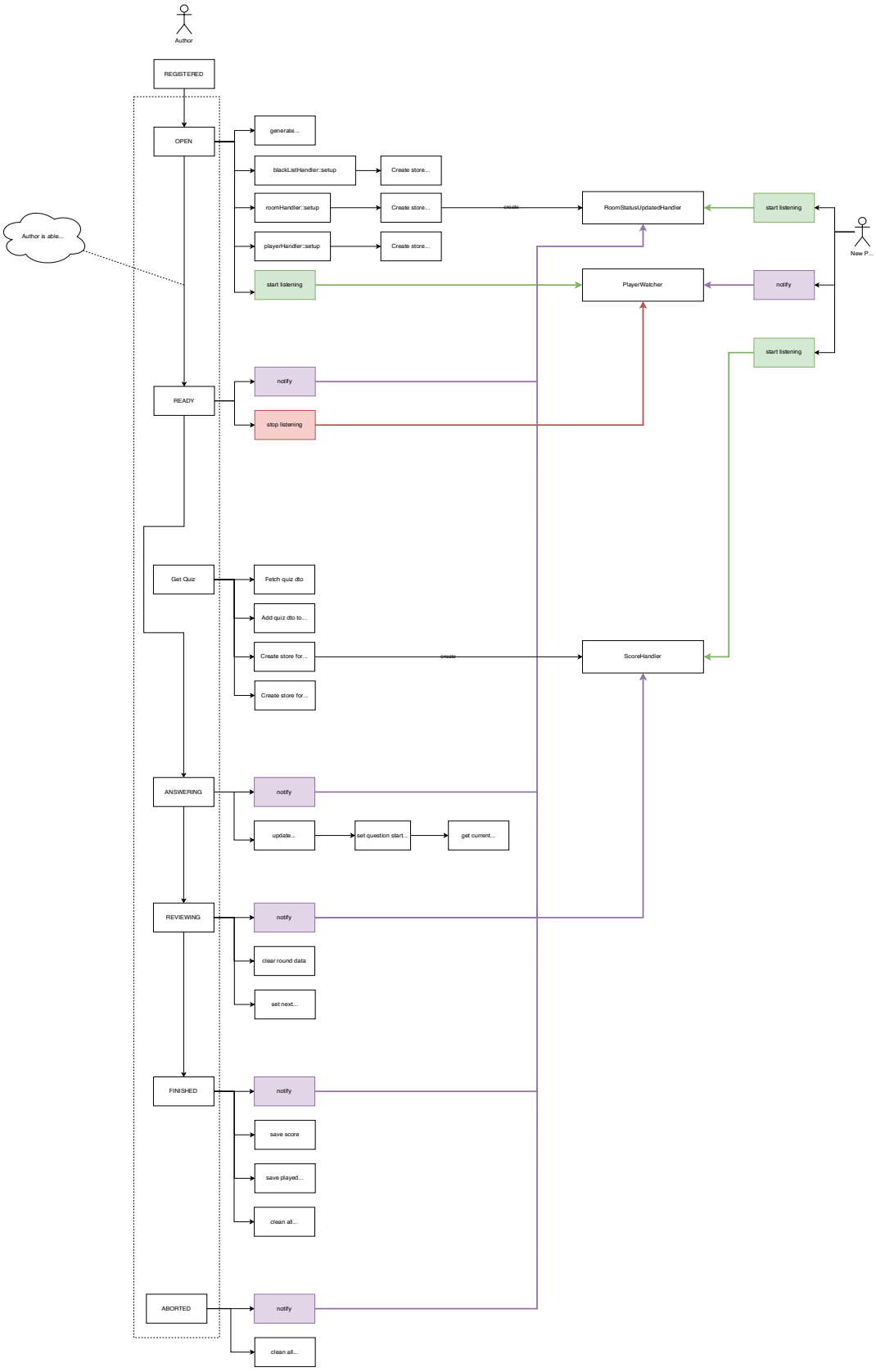


Joonis 8. Tagarakenduse küsimuse lisamise/muutmise vorm



Joonis 9. Tagarakenduse küsimustiku haldamise kasutajaliides

Lisa 5 - SSE diagramm



Joonis 10. SSE diagramm