TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Robert Tõnisson 179611

Mälumängude organiseerimise rakendus

Bakalaureusetöö

Juhendaja: Meelis Antoi
MSc

Tallinn 2021

Autorideklaratsioon

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

Autor: Robert Tõnisson

16.05.2021

Annotatsioon

Bakalaureusetöö eesmärgiks on luua rakendus mälumänguürituste korraldamiseks. See rakendus peab võimaldama luua mitmekülgseid mälumänge ning sinna inimesi/tiime osalema kutsuda. Lisaks peab mälumängu toimumise ajal olema kerge vaevata sisestada vastuseid ja pidada arvet tiimide punktiskooride üle ning pärast mälumängu toimumist peaks kõik osalejad saama tagantjärele uuesti küsimusi ja tulemusi vaadata. Siiani puudus lahendus, mis kõike seda funktsionaalsust mugavalt pakuks.

Lahenduse elluviimiseks selgitatakse läbi analüüsi loodava rakenduse skoop. Selle jaoks analüüsitakse erinevaid olemasolevaid lahendusi, tuues välja nende plussid ja miinused. Sellele järgnevalt analüüsitakse erinevaid tehnoloogiaid, mida süsteemi loomisel kasutada, ja langetatakse nende osas kaalutletud valik, et tagada jätkusuutlik ja edasiarendatav rakendus. Arendusprotsessi käigus kirjeldatakse erinevate tehnoloogiate, metoodikate ja protsesside kasutamist infosüsteemi loomiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 58 leheküljel, 8 peatükki ja 10 joonist.

Abstract

Application for Organizing Quizzes

The aim of this bachelor's thesis is to create an application that helps to organize quiz events. That application must provide opportunity to create diverse quizzes, where you are able to invite team/users to play at scheduled time. Organizer should have easier time inserting answers and checking if they are correct, as well as keeping accurate track of teams' points during any point of the game. There should also be an opportunity to revisit past quizzes. So far, the organizers have had to rely on multiple existing solutions like having quiz in one place and keeping track of points in another, also, sharing questions after the event is over is quite troublesome.

Through the analysis of existing solutions, we will be able to define the scope of the application by bringing out the positive and negative features of the existing solutions. After that, there will be an analysis of the technologies, that can be used to develop the information system. Based on the analysis, there will be augmented selection between analyzed technologies, to make sure that the created application will be sustainable and scalable.

The development is separated into two applications: server-side application and client application. Each of them using their specific technologies. During the development process, the use of different technologies, methodologies and processes for creating the information system will be described, giving the reader a simple understanding of the created applications.

The result of this thesis is a new application for organizing quiz events and a fundamental analysis that made its creation happen. The application is a solid foundation for further development.

The thesis is in English and contains 58 pages of text, 8 chapters and 10 figures.

Lühendite ja mõistete sõnastik

.NET	Microsofti tarkvaraplatform.
AJAX	*Asynchronous JavaScript And XML* – on kogum omavahel seotud veebiarenduse tehnikaid, mida kasutatakse rakenduse kliendi poolel interaktiivsete veebirakenduste loomisel.
API	*Application Programming Interface* – rakendustarkvara liides
Back-end	Serveripoolne rakendus.
BLL	*Business-Logic-Layer* – rakenduse kiht, kus on kõik rakenduse äriloogika ja andmetöötlus.
CoC	*Convention over configuration* – raamistike paradigm, mille abil üritatakse vähendada otsuseid, mida arendaja peab raamistiku konfigureerimiseks ise tegema.
Create-and-run	Idee, mille kohaselt on uue projekti loomisel juba töötava rakenduse põhi valmis.
CRUD	*Create-Read-Update-Delete* – neli peamist funktsooni, mida on vaja ühe andmesalvestuse rakenduse jaoks vaja implementeerida.
CSRF	*Cross-Site Request Forgery* – veebisaidi pahatahtlik ärakasutamine, kus volitamata käsud esitatakse kasutajalt, keda veebirakendus usaldab.
CSS	*Cascading Style Sheets* – keel mida, kasutatakse HTML dokumentide kujundamiseks.
DAL	*Data-Access-Layer* – rakenduse kiht, mis tegeleb vahendajana rakenduse ja andmebaasi vahel.
DOM	*Document Object Model* - Dokumendi objektmudel - Mudel mis luuakse veebilehe laadimisel.
DRY	*Do not Repeat Yourself* – arenduse printsiip, mille kohaselt ei tohiks luua korduvaid arenduse mustreid.
ERD	*Entity Relationship Diagram* – diagramm, mis näitab ära andmebaasimudelid ja nende suhted.
Front-end	Kasutajapoolne rakendus.
Garbage collection	Automaatne mäluhaladus.
GraphQL	Andmetöötluskeel API jaoks
JOIN	SQL lause tehe, millega saab tabelite andmeid kokku liita.
JSON	*JavaScript Object Notation* – lihtsustatud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal.

JWT	*JSON Web Token* – avatud veebimärgid kahe osapoole turvaliseks andmevahetuseks.
KJ	Kasutusjuht.
MIT litsents	Tarkvaralitsents, mille alusel võib seda kasutada igaks otstarbeks, kui sa lisad originaalse autoriõiguse ja litsentsi oma koopiasse.
MVC	*Model-View-Controller* – mudel-vaade-kotroller – arhitektuur, mida kasutatakse rakendustes.
MVVM	*Model-View-ViewModel* – mudel-vaade-vaatemudel – Tarkvaraahritektuuriline muster graafilise kasutajaliidese hõlbustamiseks.
NPM	*Node package manager* – Paketihaldussüsteem Node.js jaoks.
NuGet	.NET platvormi paketihaldur.
Props in react	Sisendid, mida kasutatakse react raamistikus.
REST	*Representational State Transfer* – tarkvara arhitektuur laad, mis paneb paika kindlad piiri veebirakenduse jaoks.
SOAP	*Simble Object Access Protocol* – suhtlusprotokoll.
Solution	.NET rakenduse projektikaust oma seadistustega.
URI	Ühtne ressursiidentifikaator on sõne, mida kasutatakse infoallika üheseks määramiseks veebis.
WAR	*Web Application Resource* – Java faililaiend veebirakenduste pakkimiseks.

Sisukord

Jooniste loetelu

1 Sissejuhatus

Viimasel ajal on inimeste elu liikunud üha rohkem digitaalmaailma. Tööd tehakse palju arvutites, omavaheline suhtlus käib peamiselt läbi interneti, isegi toitu tellitakse läbi rakenduste. Inimesed on sellest teadlikud ning tunnevad tihti, et vajavad mingit tegevust või üritust, kus nad saaksid oma sõpradega reaalselt koos olla. Üheks selliseks populaarsust kogunud ürituseks on seltskonna mälumängud, kus inimesed saavad kokku, moodustavad tiimid ja hakkavad kollektiivselt vastama rasketele küsimustele.

Käesolev lõputöö analüüsib probleemi ja uuritakse erinevaid olemasolevaid rakendusi, mis hetkel kasutajatele saadaval. Probleemi lahenduseks on välja pakutud veebirakendus, mis võimaldab inimestel lihtsa vaevaga korraldada kollektiivset mälumängu. Rakenduse kaudu saab kasutaja efektiivselt liituda mälumänguga, lisada oma tiime ja hiljem üle vaadata juba toimunud mälumänge. Korraldaja saab selge ülevaate vastustest ja punktiseisust, et mälumäng saaks toimuda sujuvalt.

1.1 Taust ja probleem

Töö autor on juba aastaid tegelenud hobikorras mälumängudega. Mälumängude korraldamine on aga sageli kujunenud vaevarikkamaks kui ta tegelikult olla võiks. Mängu ajal on vaja pidada punktiarvestust, kuid praeguste lahenduste läbi on see kohmakas, tekib palju vigu andmete sisestamisel ning arvestus läheb sassi ja selle läbi tekivad vaidlused. Lisaks on senimaani igaüks saanud paberkandjal küsimused, mida sageli ei jääta alles, ja pärast on ebamugav igaühele meili peale eraldi koopia mälumängust saata. Probleemi lahendamiseks tuleks luua veebirakendus, mis aitaks korraldajatel mängu organiseerida ning kasutajad saaksid sellega vaevata liituda.

1.2 Metoodika

Käesoleva lõputöö käigus analüüsitakse esmalt erinevate olemasolevate lahenduste funktsionaalsust, tuues välja nende puudujäägid ja plussid. Selle analüüsi abil määratakse

loodava rakenduse skoop, kus määratakse selle funktsionaalsed ja mittefunktsionaalsed nõuded.

Pärast skoobi määramist analüüsitakse selle rakenduse loomiseks erinevaid võimalike tehnilisi lahendusi, sealhulgas nii kliendipoolsed tehnoloogiad kui ka serveripoolsed tehnoloogiad.

Seejärel kirjeldatakse veebirakenduse loomist ja analüüsitakse valitud tööriistu, meetodeid, protsesse ja tehnoloogiaid lähemalt. Viimaseks analüüsitakse loodud infosüsteemi ning tuuakse välja võimalused edasiarenduseks.

2 Olemasolevate lahenduste analüüs

Käesolevas peatükis kirjeldatakse olemasolevaid lahendusi, tuues välja nende positiivsed ja negatiivsed küljed funktsionaalsuses

2.1 MäluMäng

MäluMäng on Eestimaine mälumängu veebileht. Seal saab ise mänge luua ja neid teistega jagada. Lehel on aga mitmeid probleeme. Esiteks leht on tasuline. Et teiste inimestega end proovile panna, tuleb tasuda kuumakse. Tasuta versioonis teiste tulemusi näha ei ole ning ise mänge koostada ei saa [1].

Tasulises versioonis on küll võimalik luua mälumänge, kuid küsimused saavad olla ainult valikvastustega, mis piirab lahendaja loovust ning mõtlemist [2]. Küsimusi saab ühes mängus kokku olla vaid 10 ning need peavad vastama ainult ühele teemale.

MäluMäng on mõeldud vaid ühele kasutajale korraga mängimiseks. Mängud kehtivad piiramatult ning igaüks saab neid mängida ükskõik millisel ajal. Pärast on võimalik oma tulemust võrrelda teistega.

Lehekülg näeb visuaalselt hea välja ning seal on lihtne navigeerida. Lisaks on kasutatud reageerivat veebidisaini, mis võimaldab vajadusel rakendust mugavalt ka nutiseadmes kasutada.

2.2 Tabelitöötlus

Tabelitöötluse(nt. Excel, Google Sheets) tarkvara on suurepärane tabelite loomiseks. Nende mugavaks kasutamiseks on vaja seda õppida. Seda kasutatakse mälumängude juures enamasti võistkondade punktide sisestamiseks ja punktiarvestuse pidamiseks. Seal on võimalik oma valemeid kirjutada, mis võimaldaks automaatselt tiimide punktid plokkide või kogu mängu arvestuses kokku liita. Korraldaja peaks lihtsalt iga küsimuse punktid õigesse lahtrisse sisse kandma ja kogupunktid ilmuvad ekraanile.

Tabelitöötluse programmid ei ole aga mälumängude korraldamiseks visuaalselt väga sõbralik. Küsimusi ennast pole näha, sageli ainult küsimuse number. Andmete kiire sisestamisega tulevad vead ning kõikide tiimide erinevate küsimuste vastusepunktide vahelt on raske vigu üles leida.

Lisaks eelmainitule tuleb arvestada, et tabelitöötluse tarkvara on andmete töötlemiseks ja statistikaks, mitte sotsiaalsete ürituste korraldamiseks. Puuduvad võimalused kuupäevade kokkuleppimiseks, inimeste lisamiseks/kutsumiseks ja mängude jagamiseks.

Tabelitöötluse kõrvale tuleb mälumängu enda loomiseks kasutada muud rakendust ning sellest tulenevalt on pärast võistkondadele küsimuste ja punktitabelite jagamine tülikas, kuna mängu läbiviimiseks on kasutatud erinevaid lahendusi.

2.3 QuizMaker

Quiz-Maker on väga populaarne inglise keelne mälumängude loomise lehekülg. Lehel on võimalik luua mitmekülgseid küsimusi (valikvastustega, piltidega, vahemikvastustega, vabatekstis vastustega jpm.). Küsimustele on võimalik määrata ise saadavate punktide arv, mis muudab punktiarvestuse huvitamaks. Samuti on võimalus kasutada ajalimiiti küsimuste lahendamiseks [3].

Veebirakendus on aga jällegi mõeldud individuaalseks lahenduseks, mistahes ajahetkel ning mälumängu üritusena seal korraldada on raske. Lisaks pole korraldajal võimalik ise vastuseid üle vaadata, mis on oluline just vabas tekstis kirjutatavate vastuste kontrollimiseks ning kõik vastused kontrollitakse automaatselt. Lisapunkte või karistuspunkte vastuste eest anda ei saa.

Teenuse suurim miinus on see, et tasuta versiooniga saab luua 1 mängu 10 küsimusega ning seda saavad lahendada ainult 25 inimest. Tasulised versioonid jäävad 39-99 dollarilise kuutasu juurde, mis võib mälumängu grupi jaoks kujuneda liiga kalliks. Sealhulgas tuleks ära mainida ka, et kasutaja jagamine pole lubatud, kui just ei valita kalleimat paketti, mis lubab kasutada rakendust kuni kümnel inimesel.

3 Loodava rakenduse skoop

3.1 Loodava rakenduse eesmärk.

Eesmärgiks on lõputöö autori enda elust tuleneva probleemi lahendamine. Selle jaoks tuleks rakenduse kasutajale tagada maksimaalne väärtus, tehes rakenduse, mida on intuitiivne kasutada ning muutes huviliste/korraldajate elu lihtsamaks. Rakenduse abil peab kasutaja saama kasu, tagades seda, et rakendust tullakse ka edaspidi kasutama ning soovitatakse ka teistele edasi.

3.2 Rakenduse nõuete määramine

Rakenduse nõuded on esitatud loeteluna, et parandada loetavust, ning jagatud kahte gruppi: funktsionaalsed- ja mittefunktsionaalsed nõudmised.

3.2.1 Funktsionaalsed nõuded

- Rakendus tuleb luua veebirakendusena ning kasutama reageerivat veebidisaini, et vajadusel oleks võimalik raekndust kasutada ka väiksemal seadmel.
- Rakendus peab kasutama andmebaasi info salvestamiseks.
- Süsteemi peab olema võimalik luua kasutajakonto, sinna sisse logida ja oma kasutaja andmeid redigeerida.
- Süsteemist peab olema võimalik üles leida teisi inimesi/kasutajaid ning neid peab saama sõbralisti lisada.
- Rakenduses peab olema võimalik luua mälumängu.
- Korraldaja peab saama määrata mälumängu toimumise kuupäeva/aega.
- Mälumängule peab saama lisada osalejaid ja ka nendega liituda.
- Mälumängule saab registreerida nii individuaalselt kui ka tiimina
- Mälumängule peab saama luua teemablokke.

- Mälumängu peab saama luua mitmekülgseid küsimusi ning neile peab saama panna erinevat võimalikku saadavat punktiarvu.
- Küsimusi ja teemaplokke peab saama vajadusel ka eemaldada.
- Mängijaid peab saama mälumängult eemaldada.
- Küsimustele või teemaplokkidele peab olema võimalik määrata ajalimiiti.
- Korraldaja peab saama mälumängu interaktiivselt läbi viia.
- Korraldaja peab saama mälumängu toimumisel sisestada mängijate vastuseid ja neid kas automaatselt või käsitsi kontrollida.
- Korraldajal peab olema võimalus lisapunkte anda.
- Korraldajal peab saama mängu toimumise jooksul vaadata igakell hetke punktiseisu tabelit ja varasemalt vastatud küsimusi.
- Pärast mälumängu lõppu peab korraldaja saama lõppseisu muuta.
- Kasutajad peavad saama pärast mängu lõppu tulemusi vaadata.
- Kasutajad saavad anda lõppenud mälumängule tagasisidet/hinnangut.
- Lõppenud mälumänge peavad osalenud kasutajad saama õppimise eesmärgil üle vaadata ning selle küsimusi endale salvestada.

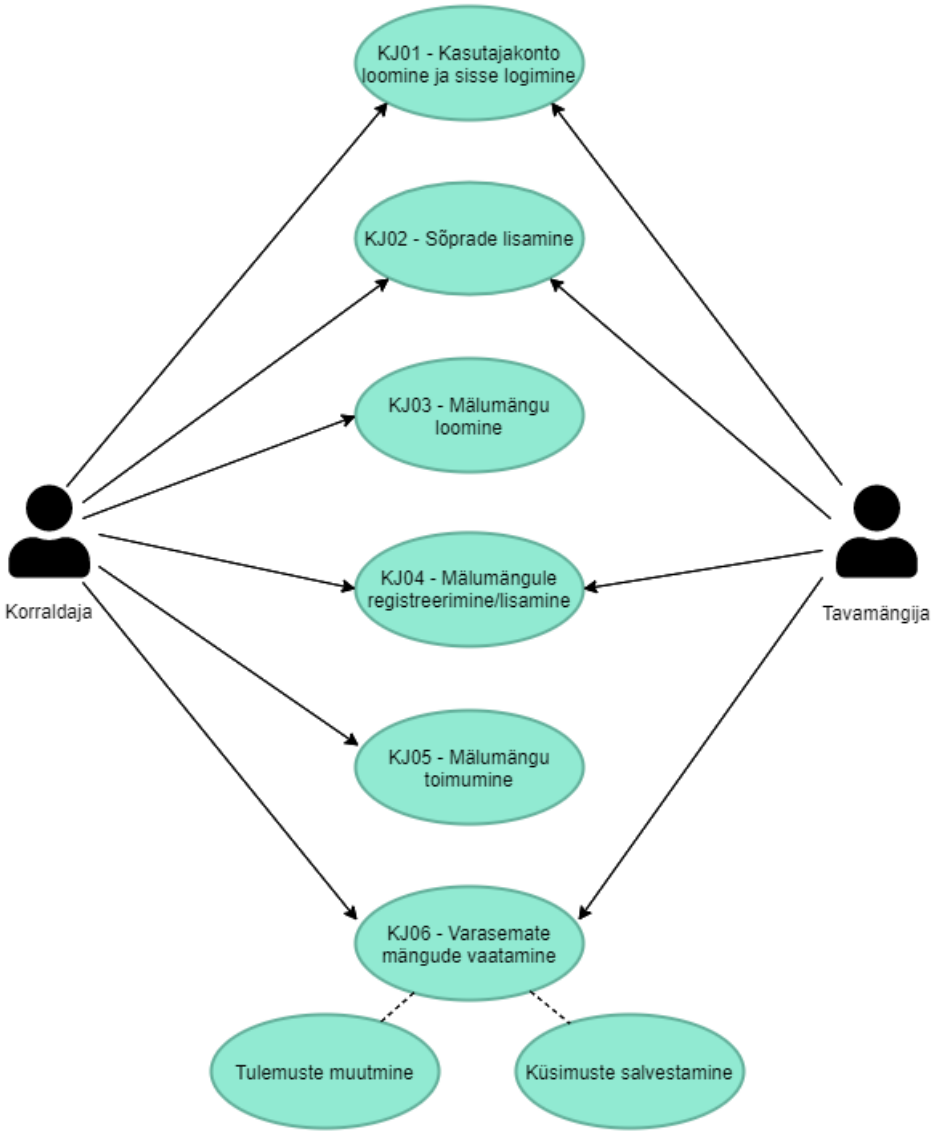
3.2.2 Mittefunktsionaalsed nõuded

- Rakenduse lähtekood peab olema avalik.
- Rakenduse lähtekood peab sisaldama taaskasutatavaid osasid.
- Kood peab olema kirjutatud järgides puhta koodi kirjutamise tavasid.
- Tarkvara peab olema dokumenteeritud.
- Tarkvara peab olema lihtsasti hooldatav ja edasi arendatav.
- Rakendus pean kasutama Git veebihaldust.

- Rakendus peab võimeline stabiilselt töös olema vähemalt 97% ajast.
- Isiklikud andmed ja paroolid ei tohi olla teistele kätte saadavad ega nähtavad.
- Rakendus peab olema esialgselt suuteleine toimetama kuni 75 inimesega paralleelselt.
- Loodav teenus peab olema tasuta kättesaadav.
- Loodav rakendus peab olema suuteline jätkama ka vea tekkimisel.
- Kasutajaliides peab järgima kasutajasõbraliku disaini printsiipe.
- API ja kasutajaliidese viivitus ei tohi mõjutada sujuvat rakenduse kasutamist.
- Rakendus peab toetama kõiki suuremaid veebibrausereid.
- Inimesed peavad olema võimelised kasutama loodud rakendust ilma eelneva koolituseta.

3.3 Kasutusjuhtude mudel

Kavandatava rakenduse kasutusjuhtude mudel on välja toodud joonisel 1. Jooniste loomiseks on kasutatud Online Visual Paradigm teenust [4]. Mudel on loodud kahe kasutajatüübi vahel – korraldaja ja tavamängija. Korraldaja on see kes ise korraldab mälumängu. Tavamängija on see, kes mängib korraldatavat mälumängu.



Joonis 1. Rakenduse kasutusjuhud.

3.3.1 KJ01 – Kasutajakonto loomine ja sisse logimine

Ülevaade: Algab rakenduse avamisega ja lõppeb kasutaja eduka sisselogimisega.

Tegutsejad: Korraldaja, tavamängija.

Eeltingimused: Kasutaja on avanud rakenduse.

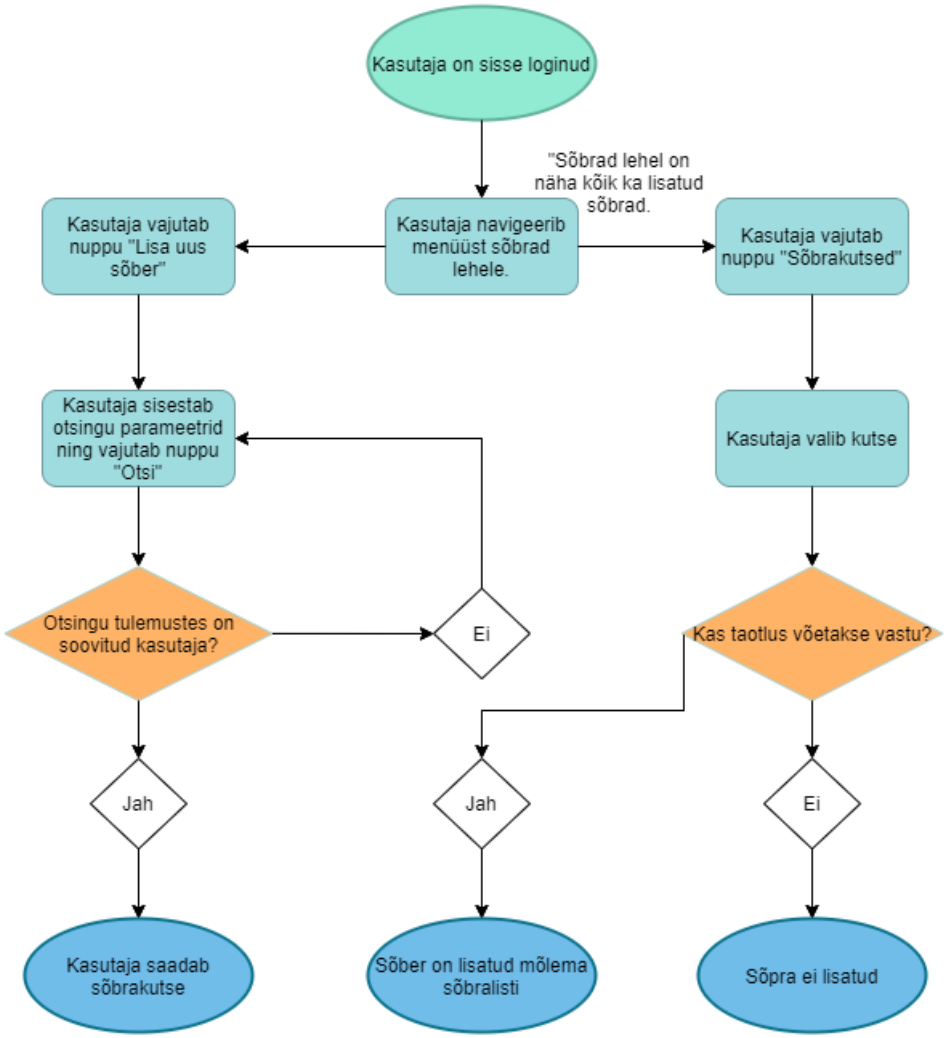
Joonis 2. KJ01 - Kasutajakonto loomine ja sisselogimine.

3.3.2 KJ02 – Sõprade lisamine

Ülevaade: Algab sõbralisti avamisega ja lõppeb sõbrakutse saatmisega või sõbrakutsele vastamisega.

Tegutsejad: Korraldaja, tavamängija.

Eeltingimused: Kasutaja on rakendusse sisse loginud.



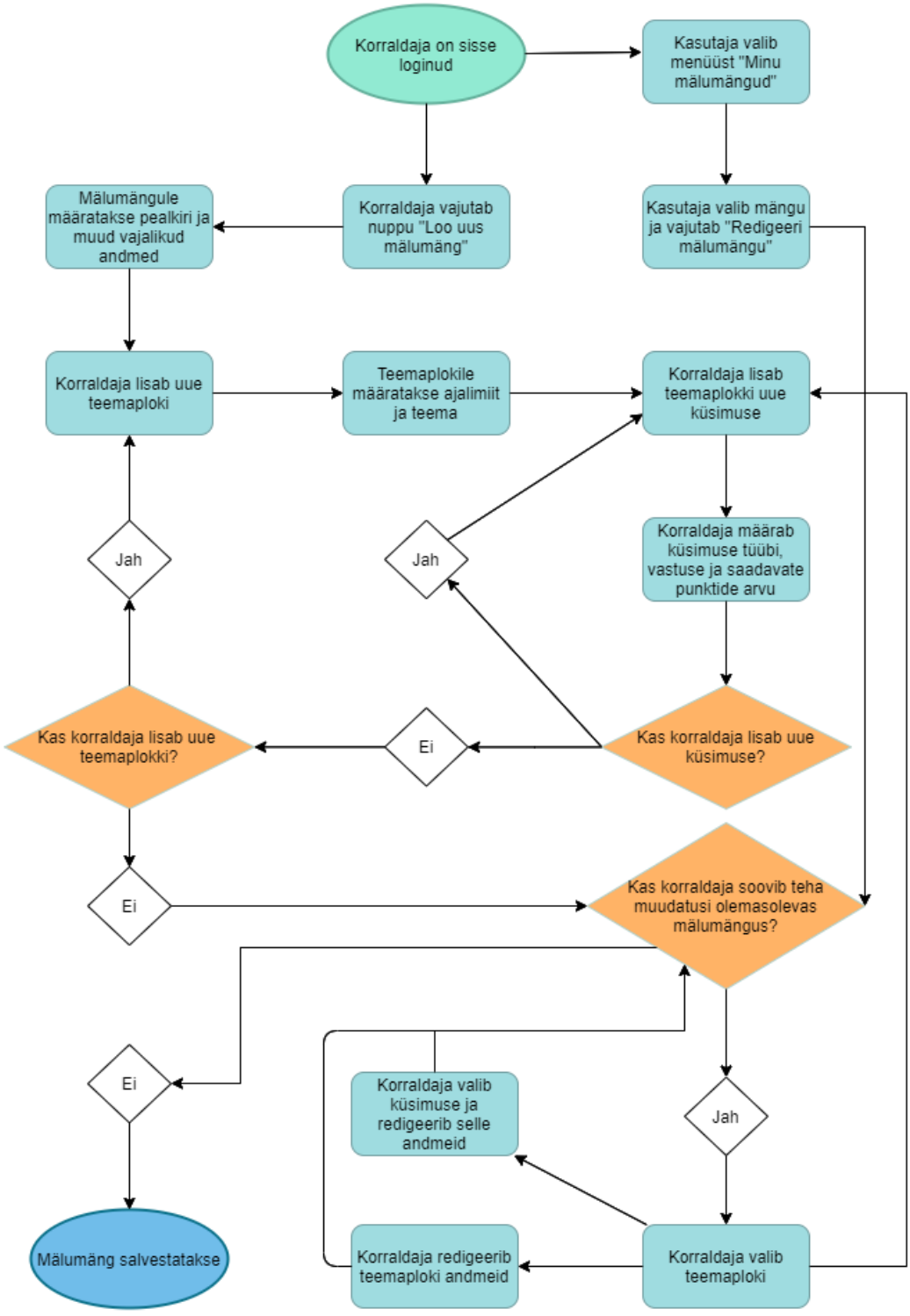
Joonis 3. KJ02 - Sõprade lisamine.

3.3.3 KJ03 – Mälumängu loomine

Ülevaade: Algab vajutusega nupule „Loo uus mälumäng“ või olemasoleva mängu avamisega ning lõppeb mälumängu salvestamisega.

Tegutsejad: Korraldaja.

Eeltingimused: Korraldaja on rakendusse sisse loginud.



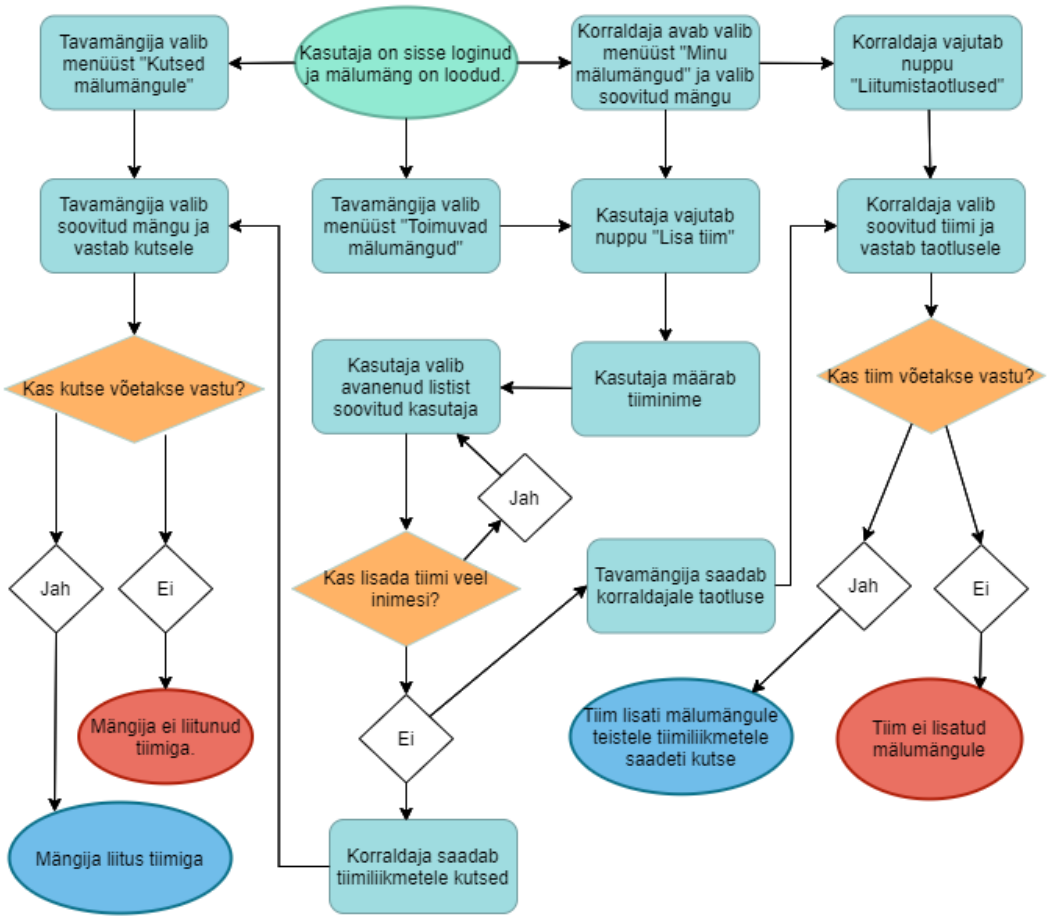
Joonis 4. KJ03 - Mälumängu loomine.

3.3.4 KJ04 – Mälumängule registreerimine/lisamine

Ülevaade: Algab kas oma korraldatava mälumängu või teiste poolt korral mälumängu avamisega ning lõppeb mälumängule registreerimise/lisamisega.

Tegutsejad: Korraldaja, tavamängija.

Eeltingimused: Korraldaja on rakendusse sisse loginud ja mälumäng on loodud.



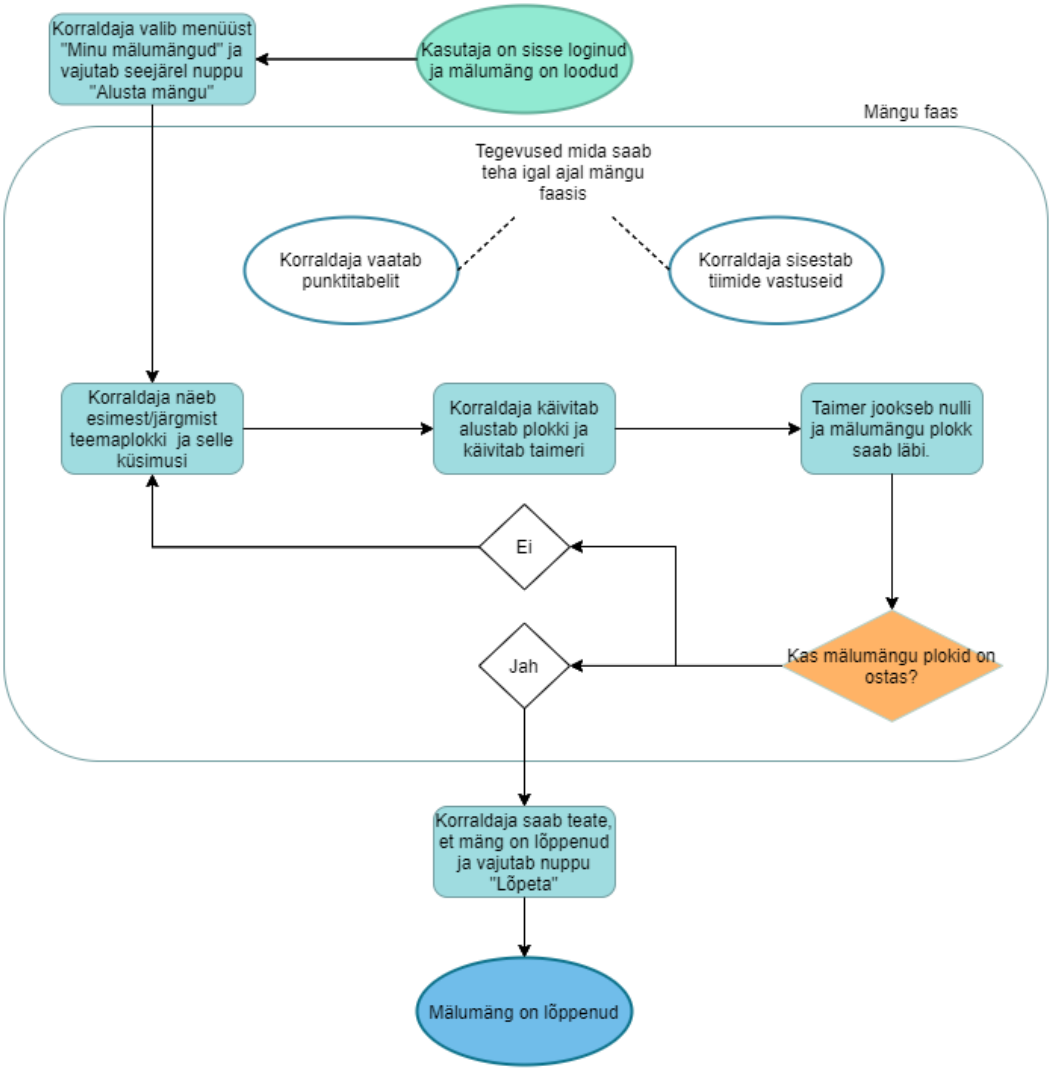
Joonis 5. KJ04 - Mälumängule registreerimine/lisamine.

3.3.5 KJ05 – Mälumängu toimumine

Ülevaade: Algab vajutusega nupule „Alusta mängu“ ja lõppeb mälumängu lõpetamisega.

Tegutsejad: Korraldaja.

Eeltingimused: Korraldaja on rakendusse sisse loginud ja mälumäng on loodud.



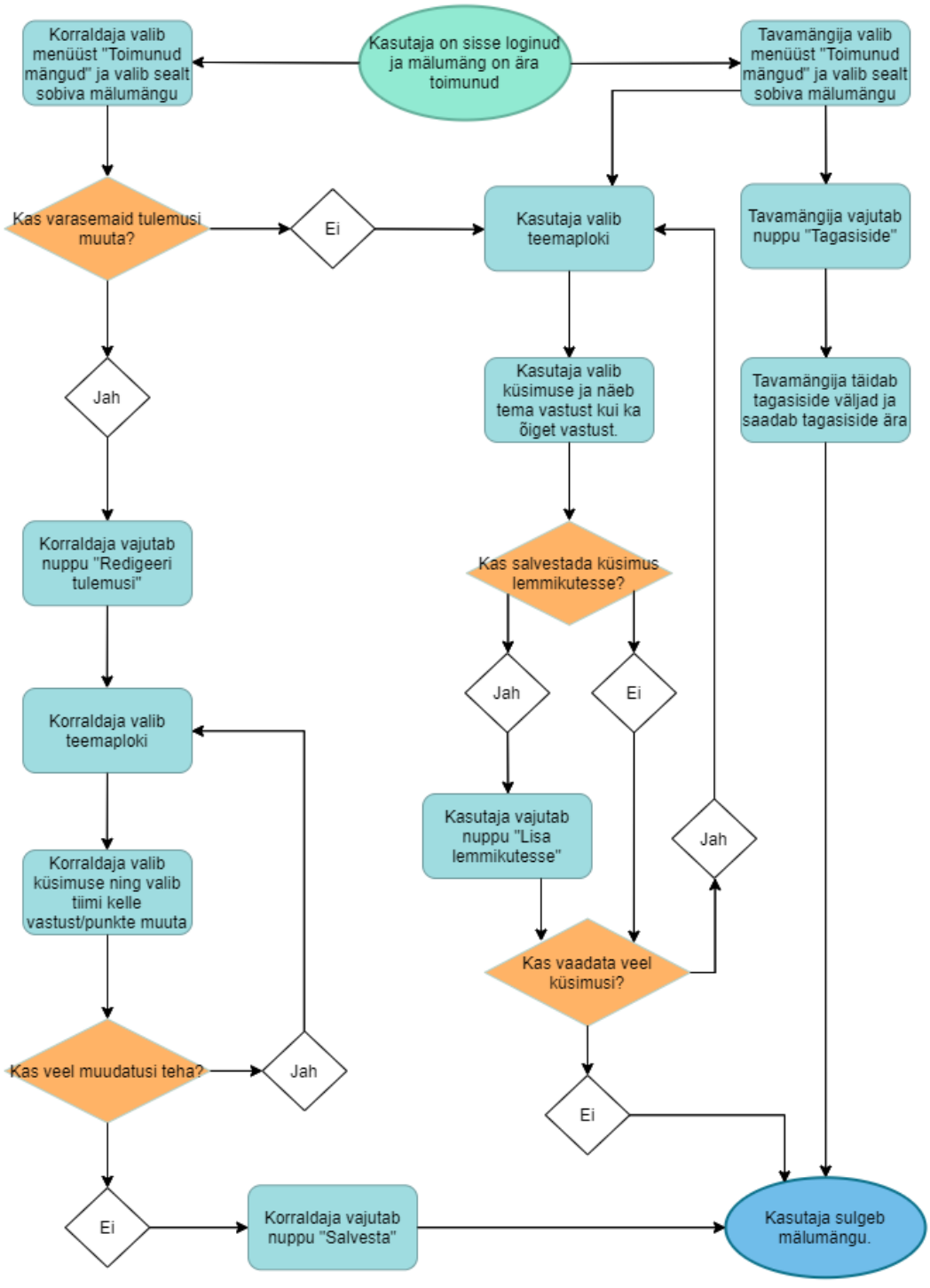
Joonis 6. KJ05- Mälumängu toimimine.

3.3.6 KJ06 – Varasemate mängude vaatamine

Ülevaade: Algab toimunud mälumängu valimisega ja lõppeb selle sulgemisega.

Tegutsejad: Korraldaja, tavamängija.

Eeltingimused: Kasutaja on rakendusse sisse loginud ja mälumäng on ära toimunud.



Joonis 7. KJ06- Varasemate mängude vaatamine.

4 Tehnoloogiate analüüs

Antud töö käigus loodav rakendus on mõeldud eelkõige mugavalt töötama suurematel seadmetel nagu süle-/tahvelarvuti. Rakenduse otstarbe tõttu pole mõistlik luua mobiilirakendust vaid veebirakendus. Seega analüüsitavate tehnoloogiate valikul on arvestatud, et loodav rakendus on veebirakendus.

4.1 Serveripoolsed(*back-end*) tehnoloogiad

Back-end on kood, mis jookseb serveril ning võtab klientrakenduse poolt vastu andmeid ning sisse ehitaud loogika abil saadab kliendi poolt soovitud andmed tagasi. *Back-end* on enamasti kolme põhikomponendiga: andmebaas, kuhu salvestatakse kindla struktuuri alusel andmeid, server, mis kuulab sissetulevaid päringuid jooksutab rakendust ning rakendus ise, kus hoitakse loogikat, mida sissetulevate päringutega teha [5].

4.1.1 Veebi rakendusliidese (*Web API*) arhitektuur

Erinevad rakendused vajavad omavaheliseks suhtluseks vahendajaid. Selle jaoks luuakse rakendusliideseid, mis annavad võimaluse ühel süsteemil pääseda ligi teise süsteemi andmetele või funktsionaalsusele. Selle jaoks, et rakendusi kiiresti ja mahukalt integreerida, on loodud erinevad protokollid ja nõuded, mida kutsutakse rakendusliidese arhitektuuriks. Kolm analüüsitavat arhitektuuri on SOAP, REST, GraphQL [6].

4.1.1.1 SOAP protokoll

SOAP on XML-formaadis rangelt standardiseeritud protokoll. Oma massiivse sõnumistruktuuri tõttu on see peale vaadates kõige keerulisem API formaat. Seda kasutatakse enamasti tähtsate transaktsioonide puhul, kus tähtis on turvalisus ja täpsus [7].

SOAP eelised:

- Funktsionaalsus veebipõhiste teenuste loomiseks võimaldab suhtlust keelest ja platvormist sõltumata.

- Seotud erinevate transpordiprotokollidega, et sobida erinevate kasutusjuhtude jaoks.
- Sisseehitatud tõrkeotsing, mis vea olemasolul tagastab veakoodi ja selgituse.
- Palju turvaelemente, tagades tehingute privaatsuse, terviklikkuse.

SOAP puudused:

- Võimaldab ainult XML struktuuri.
- XML failide suuruse tõttu on kasutab SOAP teenused vajavad palju andmemahtu.
- SOAP API serverite loomine nõuab paljude erinevate kaasatud protokollide tundmist, mis nõuab palju spetsiifilisi teadmisi.

4.1.1.2 REST tarkvaraarhitektuur

REST tarkvaraarhitektuur on SOAP'ga võrreldes palju leebemalt standardiseeritud, mis võimaldab seda kasutada palju laialdasemalt. REST teeb serveripoolsed andmed kättesaadavaks esitades erinevates lihtsates vormingutes nagu JSON, XML ja HTML [8].

REST eelised:

- REST üritab võimalikul palju klienti ja serverit lahutada. See tagab abstraktse ja paindliku süsteemi, mida on võimalik aja jooksul kergesti edasi arendada, püsides samas stabiilsena.
- Kliendi ja serveri vaheline suhtlus kirjeldab kõike nii, et REST API-ga suhtlemise mõistmiseks pole vaja välist dokumentatsiooni. REST tagastab palju metaandmeid, et klient saaks lihtsalt teada kõik vajaliku rakenduse oleku kohta päringu vastusest.
- Paljude HTTP-tööriistade taaskasutuseks on REST ainus stiil, mis võimaldab andmete vahemällu salvestamist HTTP-tasemel. Seevastu vahemälu rakendamine mis tahes muus API-s nõuab täiendava vahemälumooduli konfigureerimist [6].
- Võimalus kasutada mitut vormingut andmete salvestamiseks ja vahetamiseks.

REST puudused:

- REST API loomiseks puudub kindel viis. Milliseid ressursse ja kuidas neid modelleerida sõltub situatsioonist. Seetõttu on REST teoorias lihtne kuid praktikas veidi keerulisem.

- REST päringu vastused võivad sageli sisaldada kas liiga palju või liiga vähe andmeid, mis tekitavad vajaduse teise päringu järele.

- Ei võimalda andmeid küsida valikuliselt.

4.1.1.3 GraphQL

GraphQL on üks uuemaid laialdaselt kasutusel olevaid API arhitektuure. Selle süntaks on keeruline ja kirjeldab väga täpselt andmepäringu edastamist, võimaldades teha päringuid andmemudelitele, kus paljud objektid viitavad üksteisele. Seda kasutatakse enamasti mobiilse API või erinevate keeruliste andmesüsteemide jaoks [9].

GraphQL eelised:

- Suuteline pärima keerulisi andmemudeleid, mille relatsioonid võivad skeemis ulatuda sügavale.

- GraphQL'il puudub versioonimine. Sellel on ainult üks pidevalt arendatav versioon, mis annab juurdepääsu uustele funktsioonidele ning aitab kaasa puhtamale ja kergemini hallatavale koodile.

- Üksikasjalikud veateated. Tõrketeated sisaldavad kõiki vigaseid elemente ja viitavad täpsele päringule, mis vea tekitas.

- Võimaldab kliendile teatud päringuid valikuliselt paljastada, jättes privaatsed/mittevajalikud andmed välja.

- Automaatne API dokumentatsioon.

GraphQL puudused:

- GraphQL ohverdab oma võimsuse keerukuse vastu. Liigsed omavahel seotud väljad päringus võivad põhjustada süsteemis ülekoormuse.

- GraphQL ei kasuta HTTP vahemällu salvestamise meetmeid, mis raskendab oluliselt vahemällu salvestamist.

- Väiksemate rakenduste puhul pole nii keerulist süsteemi vaja.

- GraphQL on üsna uus ja erineb teistest API arhitektuuridest oluliselt, mis nõuab aega, et sellega kohaneda.

4.1.2 *Back-end* rakenduse programmeerimiskeeled ja raamistikud

Käesolevas alapeatükis analüüsitakse erinevaid raamistike ja keeli millele need tuginevad, tuues välja nende eelised ja puudused.

4.1.2.1 Python + Django

Python on üldotstarbeline interpreteeritav programmeerimiskeel, mis loodi Guido van Rossumi poolt esialgselt skriptimiskeeleks. Tänaseks võimaldab see kasutada erinevaid programmeerimisstiile nagu protsentuaalne, objektorienteeritud või funktsionaalne. Python on lihtne ning kergesti õpitava ja loetava süntaksiga, mis meenutab sageli tavalist inglise keelt. Seetõttu on ka Pyhtonis loodud projekte odavama hallata. Pyhton on dünaamiliselt trükitud, mis tähendab, et koodis ei pea rangelt muutuja tüüpe määrama. Lisaks sellele ei kasutata selles keeles loogilisi sulge, vaid järgitakse taande reegleid. Python toetab mooduleid ja teeke, mis soodustab programmi modulaarsust ja koodi taaskasutust. [10].

Pyhton on interpreteeritav ja seetõttu teistest kompileeritavatest keeltest aeglasem, kuna interpreteeritud käsu täitmiseks kulub palju rohke juhiseid kui otse masinkoodi lugeda ja rakendada [11]. Sellest vaatamata on Python üks populaarsemaid programmeerimisekeeli ja leiab igapäevaselt nii tavakasutuses kui ettevõtetes kasutust.

Kõige populaarsem Pyhtonile loodud raamistik on Django. Django on kõrgetasemeline avatud lähtekoodiga *back-end* raamistik, mis järgib MVC mustrit ning kasutab REST arhitektuuri API loomisel. Django sobib keerukate ja funktsionaalsete andmebaasil põhinevate rakenduste loomiseks [12]. Djangot kasutavad suured rakendused nagu Instagram, Spotify, Disqus, Bitbucket jne.

Django eelised:

- Palju valmiskirjutatud tarkvarapakette, mis vähendavad koodi kirjutamist ja võimaldab keskenduda unikaalse funktsionaalsuse loomisele. Sinna kuuluvad näiteks autentimise, administraatorilehe ja sessioonihaldus paketid [13].

- Django on väga paindlik raamistik. See ei kasuta C*onvention over configuration* põhimõtteid ning võimaldab jooksvalt raamistiku seadistada vastavalt vajadusele [13].

- Djangos rõhutatakse rohkem otsesele programmeerimisele kui kaudsele, mis võimaldab teha kiireid muudatusi [13].

- Masinõppevõime – paljud masinõppe algoritmid on kirjutatud Pythonis ning Django toetab neid pakette [13].

Django puudused:

- Djangos puuduvad kokkulepitud arenduse tavad/printsiibid, mis võib loob olukordi, kus erinevad projektid ja nende komponendid hakkavad omavahel tõrkuma, kuna on seadistatud erinevalt [14].

- Django pole mõeldud väiksemate süsteemide loomiseks, kuna nõuab palju koodi kirjutamist [13].

- Django pole suuteline tegema mitmeid päringuid samal ajal [13].

- Pyhton on ise aeglane keel ja halvasti seadistatud Django raamistik võib luua olukorra kus loodav rakendus on väga aeglane[14].

4.1.2.2 .NET Core

.NET Core on .NET Framerworki uuem versioon, mis on peale Windowsi mõeldud töötamaks ka machos ja Linuxi operatsioonisüsteemidel. See on tasuta, avatud lähtekoodiga üldotstarbeline arendusplatvorm. Mida haldab Microsoft. Seda saab kasutada erinevate rakenduste loomiseks nagu mobiili-, arvuti-, veebi-, pilverakendused, mikroteenused, mängud masinõpe jne. .NET Core kirjutati nullist, et see oleks modulaarne, kerge, kiire ja platvormide vaheline raamistik. See sisaldab ainult põhilise .NET Core põhifunktsioone, mis on vajalikud minimaalse rakenduse käitamiseks. Muud funktsioonid on saadaval NuGet'i pakettidena, mida saab vastavalt oma vajadusele

rakendusse lisada. Selline lähenemine annab :NET Core raamistikule parema jõudluse, vähendab mälukasutust ning seda on lihtsam hooldada [15]. Seda raamistiku kasutavad paljud tuntud rakendused ja nende loojad nagu Stack Overflow, Microsoft, Visual Studio, Dell, W3schools jne.

.NET Core suurim kasutajaskond kirjutab seda C# keeles. See on üldotstarbeline, objektorienteeritud, tüübikindel programmeerimiskeel, mille töötas välja Microsoft ise. See aitab arendajatel robustseid rakendusi .NET ökosüsteemi. C# sarnaneb oma olemuselt väga Javale, kuid on oma funktsionaalsuselt modernsem.

:NET Core eelised:

- Platvormide vaheline tugi. Toetab peamisi platvorme nagu Windows, Mac ja Linux.
- .NET Core kasutab tehnoloogiaid, mis nõuavad vähem koodikirjutamist. See tähendab, et rakendus on võimalik luua lihtsamalt ja vähema ajaga ning muudab koodihaldus läheb sellevõrra lihtsamaks.
- Suurepärane jõudlus võrreldes konkurentidega. .NET Core ja Kestrel veebiserveri koostöö ning pidevad uuendused raamistikus teevad sellest ühe kiireima laidalaselt kasutatava raamistiku. Lisaks võimaldab :NET Core kasutada asünkroonseid programmeerimismustreid [16].
- Toetab mitmeid erinevaid keeli nagu C#, F# Visual Basic .NET.
- Suur hulk käsurea tööriistu mugavamaks ja automaatsemaks arenduseks [15].
- Microsoftil on hea dokumentatsioon ja kasutajatugi.

:NET Core puudused:

- Paljud vanemad projektid on loodud vanematele .NET raamistike versioone, mis ei võimalda kõiki .NET Core eeliseid ära kasutada [16].
- Järsk õppimiskõver. Lihtne alustada, kuid raske täielikult omandada.

- Paljud arenduse tööriistad .NET jaoks on loodud Windowsi jaoks. Kuigi need on viiakse järk-järgult õle ka teistele platvormidele, töötavad enamus siiski paremini Windowsi peal [16].

- Paljud Microsofti arenduse tööriistad, mis lihtsustaks arendust, võivad olla tasulised.

4.1.2.3 Ruby + Ruby on Rails

Ruby on kõrgetasemeline programmeerimiskeel, loodud 1995. aastal Yukihiro Matsumoto poolt, mis tasakaalustas funktsionaalse programmeerimise imperatiivse programmeerimisega. Ruby on interpreteeritavkeel, millel on ka *garbege collection*, ning see on dünaamiliselt tüübitud. Selle süntaks on lihtne, mistõttu on seda arendajal lihtne õppida [17]. Sarnaselt Pyhtoniga, jääb Ruby jõudlus alla kompleeritavatele programmeerimiskeeltele.

Ruby on Rails on vabavaraline Ruby programmeerimiskeelel põhinev serveripoolne veebirakenduste raamistik, mis järgib MVC mustrit ning on saadaval MIT litsentsi alusel. Selle fookus on pakkuda arendajatele puhast ja vaevatult kasutatavat raamistikku, mis erineks teistest konkureerivatest raamistikest. Ruby on Rails järgib programmeerimise printsiipe nagu *Do Not Repeat Yourself*(DRY) ja *Convention over Configuration*(COC) [18]. Seda raamistiku kasutavad paljud suured ettevõtted: GitHub, Airbnb, SoundCloud, Hulu jpt.

Ruby on Rails eelised:

- Ruby on Rails on vähe aega nõudev. Seda on oma lihtuse poolest kiire õppida ning palju valmiskirjutatud moodulid ja pluginad vähendavad koodikirjutamist [18].

- Raamistik on oma olemuselt järjepidev. Paika on pandud kindald standardid, mida arendades tuleb täita [19].

- Kõrgetasemeline turvalisus, mis haldab nii murdskriptimist, SQL süstimist, CSRF'i kui ka palju teisi turvaelemente [18].

- Ühildub paljude *front-end* raamistikega [18].

- Ruby on Rails on hästi eskaleeruv, suutes toimida ka suure hulga klientide teenindamisel [19].

Ruby on Rails puudused:

- Ruby on ise aeglane keel ja halvasti seadistatud raamistik võib luua olukorra, kus loodava rakenduse jõudlus jääb kõvasti alla konkureerivatele raamistikele [18].

- Komponentide ja moodulite vahelise suure sõltuvuse tõttu on lihtne luua tavalise veebirakendusi. See-eest spetsiifilise funktsionaalsusega rakenduste loomine võib esitada raskusi ja aeglustab arendusprotsessi [18].

- Vähenev populaarsus. Viimastel aastatel on Ruby on Rails populaarsus märgatavalt kahanenud ning varsti võib tekkida olukord, kus selle raamistiku asjatundjaid on järjest raskem leida [18].

- Raamistik pole oma järjepidevuse tõttu nii paindlik.

4.1.2.4 Node.js + ExpressJS

JavaScript on üks enim kasutatud klientrakendustes kasutatud programmeerimiskeel. See on mitme-paradigmaline objektorienteeritud keel, mille abil saab veebilehti interaktiivseks muuta. JavaScript kasutab loogilisi sulge, on dünaamiliselt trükitud ning toetab sündmustest lähtuvaid funktsionaalseid ja imperatiivseid programmeerimisstiile. Oma nimele vaatamata pole JavaScriptil ja Javal palju ühist [20].

Node.js on avatud lähtekoodiga ja platvormidevaheline JavaScripti käituse keskkond, mis võimaldab serveripoolseid rakendusi kirjutada JavaScripti abil. See on populaarne tööriist pea kõiki tüüpe projektide loomiseks. Node.js jookseb Chrome jaoks loodud V8 mootoril brauseriväliselt, mis annab sellele suure jõudluse paranduse. Node.js omab enda paktihaldurit NPM, mis muudab arenduse lihtsamaks, kuna inimesed saavad oma loodud pakettide kaudu funktsionaalsust teistega jagada ja omakorda teiste pakette kasutada. Node.js pole standardiseeritud ja suurem osa koodist tuleb kirjutada ise [21].

Node.js eelised:

- Lihtne õppida. JavaScript on üks populaarsemaid programmeerimiskeeli ning paljud on sellega juba klientrakenduste loomisel kokku puutunud.

- Võimalus luua nii serveripoolsed rakendused kui ka kliendipoolsed rakendused ühe keele abil. See lihtsustab ka veebirakenduste juurutamist, kuna enamus brausereid toetab JavaScripti [22].

- Suurepärane jõudlus. Google V8 JavaScripti mootor, mis kompileerib JavaScripti koodi otse masinkoodi. Koodi täitmine on veel omakorda kiirendatud käituskeskkonnas, kuna see lubab ühel protsessil korraga töötada mitme erineva päringuga [23].

- JavaScriptil on suur ja aktiivne kommuun ning detailne dokumentatsioon [22].

- Node.js on kergesti laiendatav, mis võimaldab Node.js oma nõudmiste järele kohandada [22].

Node.js puudused:

- Üks põhilisi probleeme, millega Node.js arendajad kokku puutuvad on API stabiilsus. Rakendusliidest muudetakse pidevalt ja luuakase uusi API versioone, mis pole sageli tahapoole ühilduvad [22].

- NPM paketihaldur on küll mahukas, aga sealsete pakettide kvaliteet ja usaldusväärsus pole sageli tagatud. See loob olukorra, kus paljud kasutatud paketid ei vasta tavalistele koodistandarditele. Tihti puudub neil ka hädavajalik dokumentatsioon [23].

- Vaatamata JavaScripti suurele kasutuskonnale, ei kasuta neist suurem osa Node.js, vaid kasutavad varasemalt tuttavaid serveripoolseid raamistike [23].

4.1.2.5 Java + Spring

Java on kõrgetasemeline staatiliselt trükitud objektorienteeritud programmeerimiskeel. Java järgib rangeid süntaksireegleid. Muutujate nimed tuleb deklareerida, koodiplokke tuleb alustada ja lõpetada loogiliste sulgudega jne. Java toetab paljudele põhimõtetele nagu polümorfism, kapseldamine, abstraktsioonid jpt, mis muudavad keele võimsaks [24].

Tänu Java virtuaalmasinale on see võimalik töötama mistahes platvormil võimalikult väheste sõltuvusteta. See lähtub printsiibist „kirjuta üks kord, käivita igal pool“. See teeb

Javast väga mitmekülgse programmeerimiskeele, mida saab kasutada erinevateks otstarveteks [24].

Kõige populaarsem Javale tuginev serveripoolne raamistik on Spring, mis muudab ettevõtte tasemel rakenduste loomise oluliselt lihtsamaks. Seda saab kasutada nii traditsiooniliste WAR'i juurutamiseks kui ka eraldiseisvate Java rakenduste jaoks. Spring võimaldab hakata kohe koodi kirjutama, raiskamata aega keskkonna ettevalmistamiseks ja sedaistamiseks. Spring pakub tugevat paketttöötlust, lihtsat töövoogu ja mitmesuguseid erinevaid arendustööriistu [25].

Spring eelised:

- Java on üks tuntuimaid programmeerimiskeeli, millel on suur abivalmis kogukond. Lisaks on Javal on kümneid tuhandeid vabavaralisi teeke, mida saab oma rakenduste loomisel kasutada [26].
- Võimalik luua kiirelt töötav rakendus ilma liigse seadistamiseta tänu auto-konfiguratsioonidele [26].
- Spring ökosüsteemis on palju töövahendaid nagu Spring Boot, Spring Data, Spring Security jne, mille integreerimine on lihtne [25].
- Spring on kerge, paindlik ja kiire [26].

Spring puudused:

- Spring raamistik võib olla väga keerukas oma paljude muutujate ja komplikatsioonide poolest. See teeb Springist suure õppimiskõveraga raamistiku, mida on kerge tööle saada, aga suurte projektide loomiseks peab tegema suurel määral juurde õppima [27].
- Spring pakub arendajale suurel hulgal funktsionaalsust, aga vähese teadmise juures võivad need tekitada rohkem segadust kui neist kasu saab [27].
- Springil puuduvad suunised arenduseks. Nende dokumentatsioonis pole kirjas, kuidas erinevaid ohte vältida tuleks [27].
- Vajab palju teadmisi XML kohta [25].

4.1.3 Andmebaaside analüüs

Antud alapeatükis analüüsitakse erinevaid vabavaralisi andmebaase ja tuuakse välja nende eelised ja plussid.

4.1.3.1 PostgreSQL

PostgreSQL on avatud lähtekoodiga relatsiooniline andmebaaside haldussüsteem, mis loodi eesmärgiga olla väga laiendatav ja standarditele vastav. PostgreSQL on objektide ja relatsioonide andmebaas, mis tähendab, et see sisaldab ka erinevaid funktsioone. Seda on arendatud juba aastakümneid ja selle arhitektuur on tõestanud end olema usaldusväärne, tagades andmete terviklikkuse ja õigsuse. PostgreSQL ei ole ühegi suurkorporatsiooni omand ning selle lähtekood on tasuta saadaval. PostgreSQL järgib väga täpselt SQL standardeid järgides oma dokumentatsiooni kohasel 170 nõuet 177st, mis on vajalikud täielikuks põhivastavuse järgimiseks. PostgreSQL on tuntud oma efektiivsuse poolest suuremate andmekogumitega, aga väiksemate andmebaaside jaoks on kiiremaid tööriistu. Lisaks sellele, luuakse iga kliendi ühenduse korral uus protsess, millele määratakse ligikaudu 10MB mälu ning paljude ühenduste korral võib mälu kiiresti otsa saada [28][29].

4.1.3.2 SQLite

SQLite on tarkvarakogum, mis pakub relatsiooniliste andmebaaside haldussüsteemi. Lite viitab sellele, et SQLite on väiksemahuline, kerge seadistada ja administreerida ning vajab vähe resursse. SQLite ei vaja töötamiseks serverit, vaid on integreeritud rakendusega, mis andmebaasile ligipääsu vajab. Rakendus suhtleb SQLite andmebaasiga, lugedes ja kirjutades otse kettale salvestatud failidest. SQL vajab minimaalset tuge operatsioonisüsteemidelt või välistest teekidelt [30]. See mis võidab SQLite oma lihtsuselt, annab ta ära funktsionaalsuselt. SQLite jääb hätta suurte andmetöötlustega ning tihti räägitakse SQLite'i ebaturvalisusest. Ainult üks protsess võib korraga andmebaasis muudatusi teha. Lisaks puudub kasutajahaldus, kuna kõik andmed kirjutatakse otse kettal olevatesse failidesse, ning ainsad juurdepääsuõigused langevad sellele operatsioonisüsteemile, kus andmebaas asub [29].

4.1.3.3 MariaDB

MariaDB on MySQL populaarne kõrvalarendus, mille on loonud MySQL'i esialgsed arendajad. See toimus tekkinud mureküsimusi, kui Oracle Corporation ostis MySQL täielikult enda omandiks. Kuna MariaDB täielikult avatud lähtekoodiga relatsiooniline andmebaaside haldussüsteem, mis suudab tagada nii väikesete andmetöötlusülesannete soorituse kui ka ettevõtte tasemel. MariaDB suudab pakkuda sama funktsionaalsust, mis MySQL suudab, ning ka uut funktsonaalsust[31]. Lisaks sellele on MariaDB suuremas osas ühilduv MySQL´ga, mis võimaldab oma andmebaasid ilma vaevata üle viia MariaDB'sse. MariaDB'd peetakse väga turvaliseks oma pidavate turvauuenduste tõttu ning jõudluse poolest üks hinnatumaid [32].

4.1.3.4 MongoDB

MongoDB on tasuta kättesaadav avatud lähtekoodiga andmebaas. Erinevalt tavalistest relatsioonilistest andmebaasi haldussüsteemidest, põhineb on MongoDB mitte-relatsioonilisel andmebaasil, kus kasutatakse andmete salvestamiseks võti-väärtus paare. See tagab hea kättesaadavuse, suure jõudluse ning automaatse skaleeritavuse [33]. MongoDB päringute süntaks on lihtsam ja arusaadavam kui SQL ning MongoDB installimine ja seadistamine on märkimisväärselt lihtne. MongoDB'l pole eelnevalt määratletud andmebaasisüsteemi ja seetõttu struktureerimata andmete ja salvestusmeetodite poolest väga dünaamiline. Tänases pidavalt muutuvas andmekeskses keskkonnas võib paindlik andmemudel olla kasulik. Mitte-relatsioonilse andmebaasiga kaasnevad ka omad probleem. Näiteks pole MongoDB's võimalikud transaktsioonid ning *JOIN* meetodid pole samal tasemele mis relatsioonilistes andmebaasides. Mitme kollektsiooni *JOIN'*mine nõuab mitmeid erinevaid päringuid, mis viib segase koodi ja pikkade viivitusteni [34].

4.1.4 .NET integreeritud arenduskeskkondade analüüs

Antud alapeatükis analüüsitakse erinevaid .NET IDE-sid. Analüüsitavad IDE-d on valitud pärast serveripoole keele ja raamistiku valimist.

4.1.4.1 Visual Studio

Visual Studio on Microsofti poolt loodud IDE, millel on kolm versiooni – kogukonna versioon, professionaalne versioon ja ettevõtte versioon. Kogukonna versioon on tasuta, samas kui teiste versioonide eest tuleb maksta. Kogukonna versioon on mõeldud individuaalseks arenduseks, professionaalne väikstele tiimidele ja projektidele ning

ettevõtte versioon on mõeldud suurtele ettevõttetele massiivseks agiilseks arenduseks. Visual Studio on olnud pikka aega oma funktsionaalsuselt, mugavuselt ja tööriistade poolest juhtiv.NET IDE. Visual Studio on aga ennekõike mõeldud arenduseks Windowsi masinal ning selle MacOS versioon jääb oma funktsionaalsuselt paljuski alla [35].

4.1.4.2 Visual Studio Code

Visual Studio Code on Microsofti poolt loodud vabavaraline lähtekoodiredaktor, mis on mõeldud töötama nii Windowsil, MacOS-il kui ka Linuxi operatsioonisüsteemidel. Visual Studio Code on üldotstarbeline IDE, mis võimaldab oma massiivse laienduste koguga kirjutada koodi pea kõikides erinevates keeltes ja raamistikes. Visual Studio Code on üsna täielik valmislahendus, kuid võrreldes suurte tasuliste .NET IDE'dega, jääb ta funktsionaalsuselt ja tööriistade poolest alla. See-eest on Visual Studio eeliseks see, et see võimaldab mugavalt arendada nii *fornt-end* kui ka *back-end* rakendusi ning ei spetsialiseeri ainult üksikutele keeltele ja raamistikele [36].

4.1.4.3 JetBrains Rider

JetBrain on viimase aastakümne jooksul tõusnud üheks juhtivamaks tarkvaraarenduse tööriistade loojaks. JetBrains Rider on nende tasuline IDE mõeldud just .NET rakenduste arendamiseks. Selle funktsionaalsus ja mugavus on võrreldav Visual Studio valikuga ning võib isegi öelda et parem. Rider on väga kohandatav oma maitsele, saab luua valmiskoodiga projekte *create-and-run* põhimõttel, lihtne paktihaldussüsteem projektidele, sisseehitatud verisoonihaldus, koodi valideerimine ja refaktoreerimine, unit testide tugi, massiivne laienduste valik, koodi navigeerimine jne. Paljud ettevõtted on läinud üle JetBrainsi toodetele, kuna see aitab neil säästa nii raha kui ka aega [37][38].

JetBrains tooted on tudengitele tasuta.

4.2 Kliendipoolsete tehnoloogiate analüüs

4.2.1 Klientrakenduse raamistike analüüs

Antud alapeatükis analüüsitakse kolme peamist klientrakenduse raamistikku veebirakenduse arendamiseks. Valikus on oluline TypeScripti toetus

TypeScript on vabavaraline programmeerimiskeel, mida arendab Microsoft. See põhineb JavaScriptil, lisades sellele staatilise tüübikirjelduse. Tüübid võimaldavad kirjeldada

objekte, pakkudes paremat dokumentatsiooni ja võimaldab TypeScriptil valideerida koodi korrektset töötamist. TypeScripti kood kompileeritakse kompilaatori abil JavaScriptiks ning jookseb igal pool kus JavaScript toetatud on [39].

4.2.1.1 Angular

Angular (tuntud ka kui Angular 2) on Google poolt välja loodud avatud lähtekoodiga raamistik, mis tuli välja esmakordselt 2016. aastal ning toetub MIT litsentsile. See on uuem ja modernsem versioon kui selle eellane AngularJS (Angular 1). Angular plaanib olulise muudatusi iga kuue kuu tagant ning toetab vanemaid versioone vähemalt kuus kuud, mis annab aastase puhvri viia vajalikud muudatused, kui neid peaks üldse vaja minema. Angulari framework on kolmest analüüsitavast kõige suurema mahuga, kuid siiski suure jõudlusega. Angulari on kolmest analüüsitavast raamistikust kõige keerulisem õppida [40] [41].

Angulari projektid on struktureeritud mooduliteks, komponentideks ja teenusteks. Igal rakendusel on vähemalt üks juurmoodul ja juurkomponent. Iga komponent sisaldab malli - klass mis määrab rakenduse loogika, ning metaandmeid, mis ünäitavad Angularile, kust leida vajalikud osad oma vaate loomiseks ja kuvamiseks. Angulari mallid on kirjutatud HTML-is, kuid sisaldavad ka spetsiaalseid direktiive, et kuvada ajakohasemat ja detailsemat infot kui puhas HTML võimaldaks. Angulari teenused delegeerivad äri-loogikat, näiteks andmete küsimine või sisendite valideerimine, mida kasutavad komponendid. Angular on ehitatud TypeScriptis, seega on soovitatav arenduskeel TypeScript, kuid ka JavaScript on toetatud [41].

4.2.1.2 Vue.js

Vue.js on avatud lähtekoodiga raamistik, mis on loodud Google endise töötaja Evan You poolt 2014. aastal ning toetub MIT litsentsile. Aastate jooksul on Vue populaarsus märgatavalt kasvanud, kuigi see pole ühegi suurkorporatsiooni halduses, ning selle kaasautoreid toetatakse annetuste. Vue on võrdlemisi stabiilne, kuid suuremate uuendustega(Vue 1, Vue 2, Vue 3) on esinenud migratsiooniprobleeme. Vue on väiksemahuline ja suure jõudlusega raamistik. Vued on nendest kolmest kõige lihtsam õppida selle lihtsuse ja intuitiivse süntaksi tõttu [40][41].

Vue keskendub põhiliselt ainult vaate kihile. Vued peetakse progressiivseks raamistikuks, kuna selle funktsionaalsust saab laiendada ametlike ja kolmanda osapoole pakettidega.

Vue disain on osaliselt inspireeritud MVVM(*Model-View-ViewModel*) mustriga. Vuega tööd tehes tegeletakse enamasti *ViewModel* kihiga, mis võimaldab ajakohast vaadet kuvada. Vue mallide sõntaks võimaldab luua vaate komponente ja see ühendab HTML'i spetsiaalsete direktiivide ja funktsioonidega. Vue komponendid on väikesed ja neid saab rakenduses taaskasutada. Vue on loonud soovitatavad tavad koodi organiseerimiseks. Kuna Vue 3 on loodud TypeScriptis, siis soovitatav keel on rakenuse arenduseks TypeScript, kuid ka JavaScript on toetatud [41].

4.2.1.3 React

React on avatud lähtekoodiga raamistik, mis on loodud Facebooki poolt ja tuli esmakordselt välja 2013 aastal. Facebook kasutab Reacti enamustes nende toodetes. React on MIT litsentsi all. Reacti jaoks on stabiilsus üks olulisemaid faktoreid, mistõttu uuendused raamistikus võetakse vastu alati äärmise tähelepanuga, et võimalikult vähe oleks konflikte varasema koodiga. React on võrdlemisi väikse mahuga raamistik. Reactis puudub kindel projektistruktuur, mis teeb selles küllaltki paindlikuks. React on samuti suure jõudlusega. Kolmest analüüsitavast on Reacti kõige lihtsam konfigureerida ja selle süntaks on arenduseks lihtne, kuid teatud nüansid arenduse käigus nõuavad lisa tähelepanu[40][41].

Reacti Elemendid on kõige väiksemad Reacti rakenduse osad. Need on võimsamad kui DOM elemendid, kuna neid uuendatakse muutuste korral efektiivsemalt. Reacti Komponendid on Recti rakenduse suuremad osad, mis defineerivad kogu rakenduses kasutatavad sõltumatud ja korduvkasutatavad tükid. Nad võtavad vastu sisendeid, mida kutsutakse *props*'ideks ning toodavad HTML elemente, mis kuvatakse kasutajale. Reactil on JSX tugi, mis võimaldab luua HTML ja JavaScripti korraga sisaldavaid elemente, kuid seda saab teha ka Reacti enda funktsionaalsuse kaudu [41].

5 Tehnoloogiate valik

Tehnoloogiate valimisel on arvesse võetud järgnevaid tegureid:

- Nõuded projektile. Valitud tehnoloogia abil peaks saama lahendada rakendusele määratud nõudeid. Igal tehnoloogial võib olla eripärasid ja võimalusi, mida teistel pole ning neid tuleks infosüsteemi luues arvesse võtta.

- Ajaline piirang ja olemasolevad tehnilised teadmised. Paljude uute tehnoloogiate õppimine võib olla ajaliselt kulukas ning töö tähtaega arvestades pole see sageli mõistlik tegu. Võimalusel tuleks eelistada tehnoloogiaid, mida infosüsteemi looja tunneb.

- Edasiarenduse võimalused. Töö käigus loodav rakendus peab olema edasiarendatav ja selle jaoks tuleks valida tehnoloogiad, mis oleks ka tulevikus arvestatavad või mida arendatakse edasi vastavalt tekkivatele vajadustele.

- Tehnoloogia dokumentatsioon, kogukond ja selle tugi. Arenduse ajal tekkivatele probleemidele ja küsimustele peab olema võimalik leida lahendus. Tehnoloogia funktsionaalsus võib olla suurepärane, kuid kui seda puuduva dokumentatsioon/toe abil rakendada pole võimalik, siis arenduses pole sellest kasu.

- Turvalisus. Tehnoloogiad peaksid olema turvalised nii rakenduse kasutaja kui ka haldaja huvides.

- Ühilduvus teiste tehnoloogiatega.

5.1 Serveripoolsete tehnoloogiate valik

5.1.1 Veebi API arhitektuuri valik

Veebi rakendusliidese arhitektuuriks on valitud REST tarkvaraarhitektuur, mille abil hakatakse rakenduste vahel infot JSON kujul üle kandma. REST arhitektuuri paindlikus muudab selle lihtsasti arusaadavaks ja kasutatavaks ning töötava süsteemi saab luua võrdlemisi kiiresti. SOAP protokollist on see oluliselt kiirem. Samuti SOAP'i täpsust ja turvalisust transaktsioonidega antud rakenduses vaja ei lähe. Heaks alternatiiviks

REST'ile võiks olla GraphQL, kuid kuna tegemist on üsna uue arhitektuuriga, siis selle õppimine nullist oleks ajaliselt liiga mahukas. Lisaks pole loodav rakendus nii keeruline, et läheks vaja komplekksseid päringuid, mida pakub GraphQL. Lisaks on autoril eelnev kogemus REST arhitektuuriga.

5.1.2 Serveripoolse programmeerimiskeele ja raamistiku valik

Serveripoolseks raamistikuks on valitud .NET Core raamistik, kus arendatakse antud töös loodavat rakendust C# programmeerimiskeeles. Valikus arvestati peamiselt :NET Core raamistiku jõudlust ja mugavust. Töötava :NET rakenduse loomiseks ei kulu palju aega ning see raamistik sobib antud rakenduse skoobiga. Lisaks on C# objektorienteeritud ning võimaldab koodi jaotada erinevateks mooduliteks, mida taaskasutada. Microsofti poolt loodud dokumentatsioon on hea ning kogukonnalt tugi on suurepärane. Töö autor on seda ka varem kasutanud, seega kulub vähem aega õppimisele ja saab pühendada rohkem aega kvaliteetsema rakenduse loomiseks.

5.1.3 Andmebaasi valik

Andmebaasi valikust jätsin esimesena välja MongoDB ja SQLite. MongoDB kasutamisel võivad tekkida raskused keeruliste päringute tegemisel ning autoril puudub varasem kogemus mitte-relatsiooniliste andmebaasidega. Kuna töös on vaja luua kasutajagruppe ,õiguseid ning hoiustada isikuandmeid, siis SQLite pole oma serverita olemuselt selleks kõlblik. Valik osutus MariaDB peale just oma suure funktsionaalsuse tõttu ning selle jõudlus on parem kui PostgreSQL'i jõudlus antud rakenduse skoobis. Töö autoril on kogemus mõlema andmebaasiga, mis ei määranud otsustamisel rolli.

5.1.4 IDE valik .NET arenduseks

Lõputöö kirjutamise vältel on toot autor registreeritud tudengina, mis võimaldab saada tasuta ligi kõikidele JetBrains toodetele ning teades, et need on ühed kõige võimekamad IDE-d , mis on saadaval , siis on serveripoolseks IDE-ks valitud JetBrain Rider. Lisaks sellele on töö autor sellega juba varasemalt tööd teinud.

5.2 Kliendipoolsete tehnoloogiate valik

5.2.1 Klientrakenduse raamistiku valik

Kõik kolm analüüsitavat raamistiku on kiired, neil on hea kasutajatugi ja dokumentatsioon ning funktsionaalsuselt teevad ära kõik, mis ühe klientrakenduse loomiseks on vaja. Seega kaheks otsustamisel tehtavaks faktoriks olid lihtsus ja autori enesekindlus antud raamistikuga. Kõige paremini vastas nendele kriteeriumitele React, mida on lihtne seadistada ning autoril on sellega kõige suurem varasem kogemus.

5.2.2 Klientrakenduse IDE valik

Klientrakenduse IDE jaoks on valitud Visual Studio Code, kuna rakenduse loojal on sellega varasem React rakenduse kogemus olemas. Lisaks annavad selle IDE suur hulk laiendusi igati täisväärtusliku arendusekogemuse. Samuti on Visual Studio Code tasuta, seega edasiarenduse jaoks sobib hästi.

6 Süsteemi arendus

Arendus on jagutud kahte peamisse peatükki: serveripoolse rakenduse arendus ja klientrakenduse arendus.

6.1 *Backend* arendus

Serveripoolse rakenduse ülesandeks on HTTP päringutele vastamine, andmete töötlemine ja äriloogika.

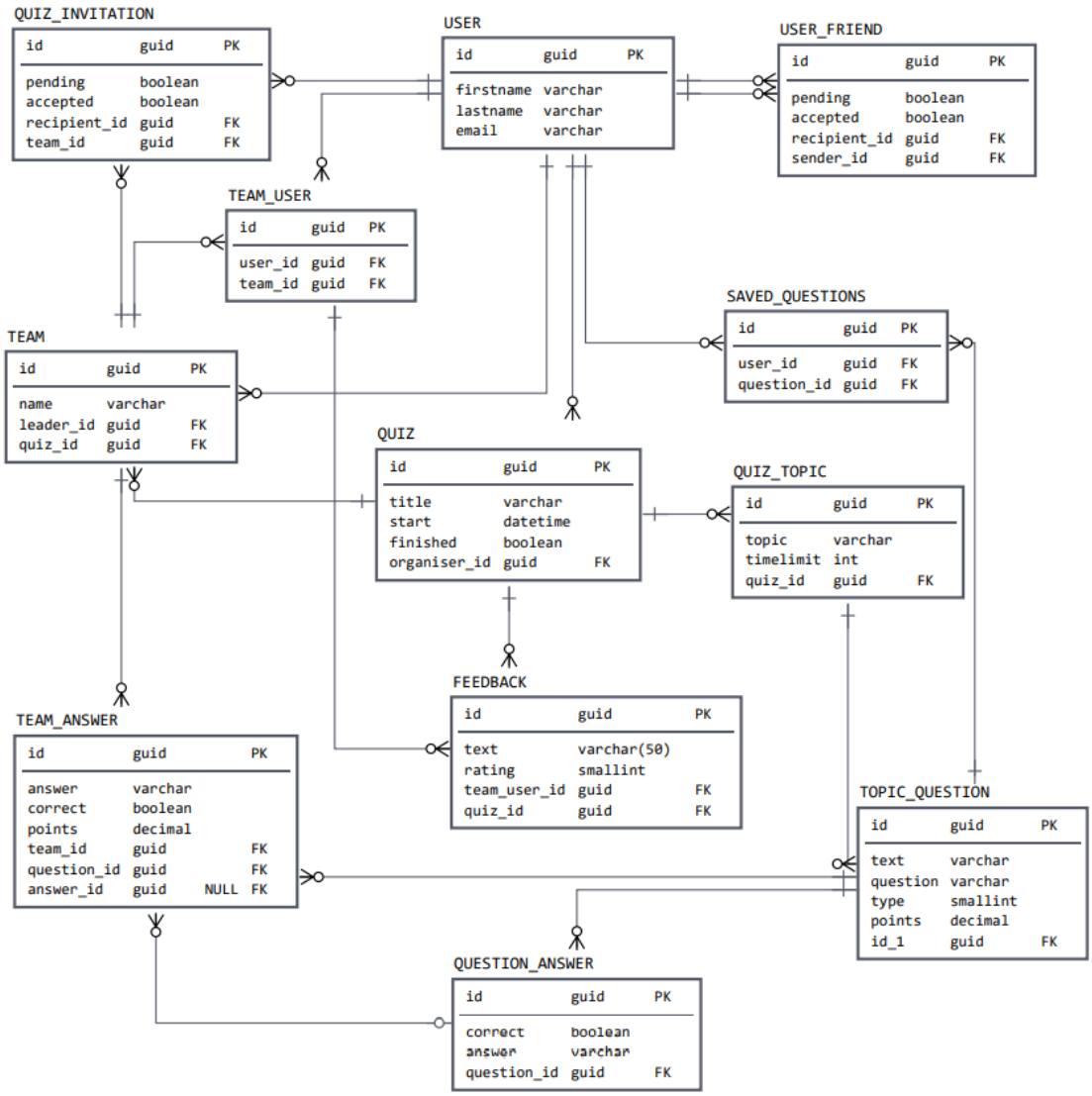
6.1.1 Rakenduse põhja loomine

Esialgse projekti loomiseks on kasutatud Rider IDE projekti genereerimise funktsionaalsust. NET rakenduste jaoks. Rakenduse tüübiks on valitud veebi API ning autentimiseks on kasutatud individuaalse autentimise tüüpi. Selle alusel luuakse juba töötav rakendus, kuhu on võimalik kasutaja luua ja sisse logida. Osa loodud failidest on loodava rakenduse jaoks ebaolulised ja võib kustutada, nt. kontrollerid ja mudelid, kuna need ei ole loodava rakenduse jaoks.

.NET rakendus on määratud lahenduses(*solution*), mis koosneb erinevatest osadest, mida kutsutakse projektidest. Need on kogumid erinevatest failidest, mis kompileeritakse kas käivitatavateks failideks, teekideks või veebilehtedeks. Iga projektil on erinevad seadistusfailid ning kasutavad erinevaid *NuGet* pakette [42]. Genereeritud lahendusel on üks peamine projekt, kus asuvad veebirakenduse käivitus- ja seadistusfailid. Hiljem lisanduvad sinna ka kõik API kontrollerid. Peamine rakenduse seadistamine käib läbi *Startup.cs* faili. Seal on juba genereerimisel loodud seadistused ja teenused, ning arenduse käigus lisatakse sinna neid vajadusel juurde. Kõik rakenduses loodud projektid on loodud kõige uuema .NET raamistiku peale, milleks on .NET 5.0.

6.1.2 Andmebaas ja selle mudelid, Enity Framework

Andmemudelite koostamiseks ja nende kirjeldamise metoodikaks on valitud ERD (*Entity Relationship Diagram*). Joonisel 8 on näidatud loodud olemi-suhte diagramm. Skeemilt puuduvad ASP.NET poolt automaatselt loodavad autentimise jaoks vajalikud tabelid(kasutajarollid, kasutaja identifikaatorid jne.).



Joonis 8. ERD-skeem.

Loodava serveripoolse rakenduse jaoks kasutatakse andmebaasiga suhtlemiseks *Entity Frameworki* (EF).EF on objekt-relatsiooniline kaardistaja, mis lihtsustab andmebaasi tabelite ja veergude andmed tarkvaralise objektide kujule. Andmebaaside loomiseks lähenetakse kood-enne meetodil. See tähendab, et mudelid luuakse tarkvaras ning selle alusel luuakse andmebaas ja selle migratsioonid.

Selle lähenemise eeliseks on see, rakendust arendades saab loogika kirjutada tavalises mõistetavas programmeerimiskeeles. Ei ole tarvis kirjutada koodi SQL lauseid andmebaasiga suhtlemiseks, vaid saab kasutada sisseehitatud meetodeid, mille abil EF genereerib ise SQL laused. Need meetodid on ühised kõikide andmebaasitabelite jaoks, mis vähendab oluliselt koodi/SQL kirjutamist. Andmebaasi poolt tulevad vastused

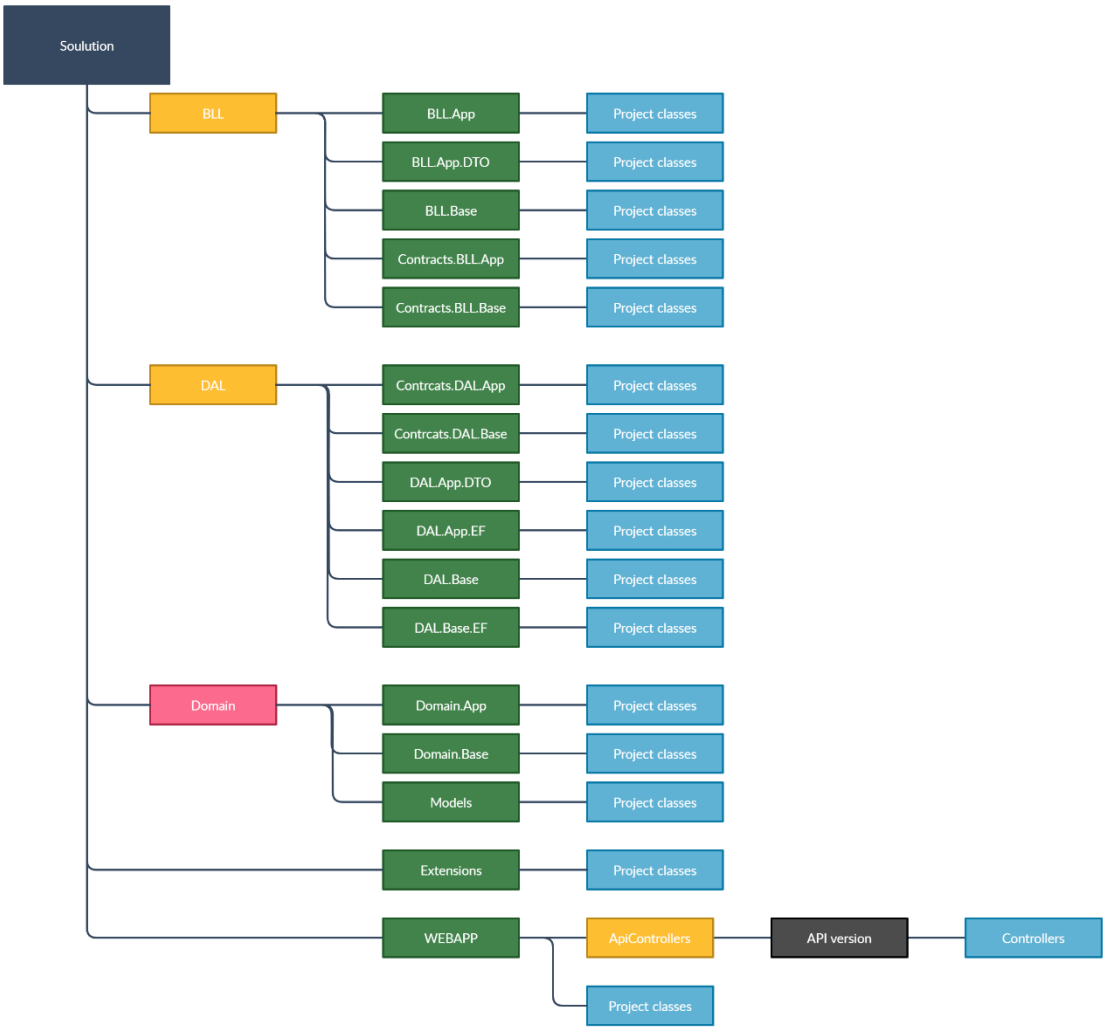
kaardistatakse samuti automaatselt tarkvara poolseteks mudeliteks. Tarkvarapoolsetele mudelitele on lihtne kirjutada erinevat äriloogikat andmete töötlemiseks, mida klientrakendusele edasi anda[43].

Tarkvarasiseseid mudeleid on EF abil väga lihtne luua. Need on tavalised klassid väljadega, kuhu saab annotatsioonide abil määrata erinevaid andmebaasi jaoks vajalike spetsifikatsioone nagu (maksimaalne suurus, komakohad, primaarvõti, välisvõti jne). Kõik tüübid nagu *string, decimal, integer, boolean* kaardistatakse andmebaasi jaoks vastavale kujule, vastavalt andmebaasile, millega suheldakse.

EF jaoks kõige olulisem klass on andmebaasi kontekst klass (*AppDbContext*). See suhtleb andmebaasiga ja võimaldab tavalisi CRUD operatsioone. Seal määratakse ära milliseid mudeleid kaardistada ning vajadusel saab kirjutada keerulisemat loogikat andmebaaside mudelite loomiseks.

6.1.3 Mitmetasandilist arhitektuur

Loodaval rakendusel on kasutatud mitmetasandilist arhitektuuri. See on üks populaarsemaid arhitektuurimustreid, mis aitab kaasa tänapäeval tavaliseks saanud keerukate rakenduste loomisele, muutes töötamise oluliselt agiilsemaks. See põhineb rakenduse jagamist erinevateks kihtideks, millel igaühel on oma ülesanne ja tähtsus. Kuna iga tasand on üksteisest eraldiseisev, siis on rakenduse ümbertöötamisel võimalik tegeleda ühe kindla kihiga, selle asemel, et kogu rakendus ümber kirjutada. Lisaks on palju kihte ja seal olevat loogika korduvkasutatav, seega on uute rakenduste loomine selle võrra lihtsam. Mitmetasandilise arhitektuuri kasutamine väldib seda, et veebi API kaudu saaks suhelda otse andmebaasiga, tagades oluliselt parema turvalisuse [44]. Loodava rakenduse käigus on kasutatud kõige tavalisemat kolmetasandilist arhitektuuri: andmete juurdepääsukiht DAL(*Data Access Layer*), rakendus/äriloogikakiht BLL(*Business Logic Layer*) ning andmeedastuskiht.



Joonis 9. Serveripoolse rakenduse struktuur.

6.1.3.1 Andmete juurdepääsukiht (DAL)

Rakenduse jaoks loodud andmete juurdepääsukiht tagab suhtluse andmebaasiga, luues vastavad ühendused ja ligipääsud ning kaardistades andmed vajalikule kujule. See tegeleb peamiselt andmete salvestamise ja küsimisega andmebaasist, veendudes , et ainult lubatud toimingud/andmed oleksid kättesaadavad. See kiht tagab suhtluse mistahes andmebaasiga, tagastades alati samaväärsed andmed mistahes päringu korral. See on rakenduse kõige madalam kiht, mis eraldab äriloogika andmetest. Eraldi suhtlus rakenduskihiga tagab, et andmete juurdepääsukihti jõuavad ainult korrektsed andmed, mis väldib vastuolusid andmebaasiga[44].

6.1.3.2 Rakenduskiht(BLL)

BLL kihis on kõik rakendusega seotud loogika. Rakenduskihi klassid täidavad andmeedastuskihist saadud ülesandeid. Selle jaoks küsitakse vajalikud andmed DAL kihis, töödeldakse andmed loodud äriloogika abil sellele kujule nagu neid andmeedastuskihist küsiti ning saadetakse need siis juba õigel kujul tagasi. BLL valideerib andemete korrektsuse ja saadab ainult lubatud kujul andmed edasi andmete juurdepääsukihti.

6.1.3.3 Andmeedastuskiht

Andmeedastuskiht on rakenduse kõige ülemine kiht, see kiht lõppkasutajale (klientrakendusele) kättesaadav kiht, mille abil saavad kasutajad vajaliku informatsiooni küsida. Loodava rakenduse puhul on see REST arhitektuuril põhinev veebi API, mille jaoks on loodud kontrollerid erinevate API lõpp-punktidega. Kliendi pöördumisel suunatakse töö HTML päringu URI(ühtne ressursiidentifikaator) alusel õigesse kontrollerisse vastavasse lõpp-punkti meetodisse, kus siis juba antakse HTML päringust saadud andmete alusel juhtnöörid BLL kihile. BLL tagastab õiged andmed kontrolleri meetodile, mis omakorda edastab need kliendile. Programmeerimise head tavad näevad ette, et kontrollerites loogikat ei kirjutata. Sellel on kolm ülesannet: võtab päringu andmed vastu, edastab selle rakenduskihile, ja saadab päringu vastusega küsitud andmed tagasi. Loodavas rakenduses andmeedastuskiht võtab vastu ja saadab kliendi andmeid JSON formaadis, mis kaardistatakse vajalikkudeks objektideks.

6.1.4 JSON Web Token tugi

Rakenduses on seadistatud JWT(*JSON Web Token*) abil autentimise tugi. See on standard, mida kasutatakse rakenduse juurdepääsulubade loomiseks. JWT on krüptograafiliselt allkirjastatud, seega on garantii, et JWT kättesaamisel on selle autentsus korrektne, kuna vahendaja ei saa neid pealt kuulata ega muuta. See töötab põhimõttel, et server genereerib kasutaja identiteeti tõendava loa ja ning saadab selle kliendile. Loodavas rakenduses tehakse see kasutaja sisselogimisel. Iga järgneva päringu, mis kasutaja teeb, annab ta tema jaoks genereeritud JWT kaasa ning serveril on selle järgi võimalik tuvastada kellelt see pärineb. See aitab serveril tagada, et kasutaja pääseb ligi ainult temale kättesaadavale informatsioonile [45].

6.1.5 Swagger tugi

Korralikul avalikul veebi API-l peab olema ka kõikidele kättesaadav ja loetav dokumentatsioon. Selle jaoks on rakenduses seadistatud Swagger. Swagger võimaldab kirjeldada API struktuuri nii, et ka masinatel oleks võimalik seda lugeda ja aru saada. Selle abil on võimalik luua automaatselt visuaalselt kena ja interaktiivne dokumentatsioon, koos automaatse testimise funktsionaalsusega. Loodavas rakenduses on loodud Swagger dokumentatsioon XML kommentaaride alusel, mis on kirjeldatud API lõpp-punktides asuvatel meetoditel ning nende meetodite sisend ja väljund JSON alusel [46].

6.2 Klientrakenduse arendus

Uue projekti põhja loomiseks kasutati Reacti ja Node käsuliidest.

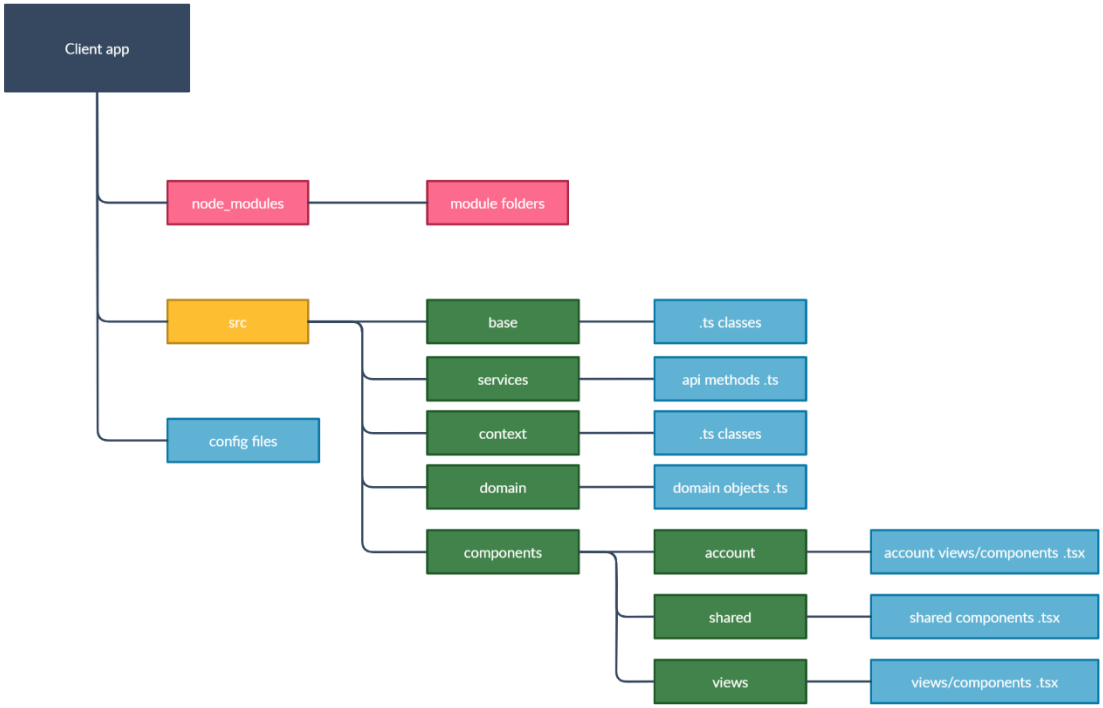
6.2.1 Klientrakenduse struktuur

Loodud rakenduse juurkaustas asuvad seadistusfailid. Need hoomavad nii paketihalduseks vajalike faile kui ka keele ja rakendusespetsiifilisi faile. Seadistusfailid on JSON kujul. Lisaks on juurkaustas kaks alamkausta: *Node* moodulite kasut, kus asuvad kõik installitud moodulid, mida on vaja rakenduse toimimiseks. Ning rakenduse lähtekaust.

Lähtekaustas on kaks rakenduse põhifaili. *Index.tsx* ja *App.tsx. Index* failis saab alguse rakenduse töö kuvades kasutajale rakenduse, mis on kirjeldatud failis *App.tsx* ning imporditakse kõik vajalikud globaalsed moodulid. *App* failis on ära defineeritud ruuter, mis suunab kogu rakenduse sees erinevatele kirjeldatud lehekülgedele URI abil. Lisaks seadistatakse seal globaalsed komponendid, mida kuvatakse igal leheküljel ning ka kontekst, kus on andmed , millele saab ligi igal pool rakenduse sees. Lähtekaust on omakorda jagatud erinevateks osadeks, et lihtsustada arendust ja failihaldust: baaskaust, komponentide kaust, kontekstide kaust, domeenkaust ja teenuste kaust.

Baaskaustas on erinevad abiklassid ja liidesed. Kontekstide kaustas on ära defineeritud erinevad kontekstid, mille abil saab erinevate lehekülgede ja komponentide vahel infot jagada. Domeenkaustas on kõik objektid, mida kasutatakse rakenduses ja ka API'ga suhtlemiseks. Teenuste kaustas on kõik teenused, mille abil tehakse päringuid serveripoolses rakenduses asuva veebi API poole. Komponentide kaust on kõige suurem,

mis jaguneb omakorda kolmeks osaks. Kasutaja komponendid ja vaated, mis on seotud just kasutajapõhiste teenustega nagu sisselogimine, registreerimine jne. Teine osa on Jagatud komponendid, kus on kõik komponendid, mida saab erinevates vaadetes taaskasutada. Kolmas osa on vaated, kus on eraldi kõik vaated koos nende spetsiifiliste komponentidega.



Joonis 10. Klientrakenduse struktuur.

6.2.2 Bootstrap

Rakenduse autor ei ole õppinud veebidisaini ning varasemad kogemused HTML kujundamisel CSS abiga on vähesed. Seega oli vaja rakenduse arenduseks juba valmistehtud raamistiku, mis aitaks klientrakendust visuaalselt paremaks muuta. Selle jaoks valati arenduse käigus tasuta raamistik Bootstarp.

Bootstrap on maailma kõige populaarsem *front-end* disaini tööriist, mis võimaldab arendada reageerivaid saite kiiresti, kasutades suurel hulgal käepäraseid ja korduvkasutatavaid HTML, CSS ja JavaScripti koodijuppe. Selle abil säästab arendaja suurel mahul disainile kuluvat aega ning saab tegeleda rohkem rakenduse funktsionaalsusega [47].

6.2.3 JQuery

JQuery on JavaScripti teek, mis järgib „kirjuta vähem, tee rohkem“ printsiipi. See võtab palju tavalisi ülesandeid, mille lahendamiseks oleks vaja kirjutada palju JavaScripti ridu ja loob nendest meetodid, mida on võimalik kustuda välja ühe koodireaga. Jquery on üks kõige populaarsemaid ja laiendatavaid JavaScripti teeke, millel suur kasutuskonna tugi [48]. Loodavas rakenduses kasutatakse ära väga palju JQuery funktsionaalsust.

Põhiline Jquery funktsionaalsus:

- HTML/DOM elementidega töötamine.
- CSS muutmine.
- HTML sündmustega tegelemine.
- Erinevad efektid ja animatsioonid.
- AJAX (*Asynchronous JavaScript And XML*) tugi.

7 Testimine

Enamus rakenduse testimist on toimunud arenduse käigus käsitsi testides erinevaid situatsioone. Selle abiks täidetakse andmebaas serveripoolse rakenduse esmakordsel käivitamisel andmetega, et vähendada käsitsi andmete sisestamise töö. Rakendust testiti klientrakenduse loomisel. Luues uue komponendi, prooviti selle kasutust erinevate situatsioonide järgi, näiteks puudulikude andmete korral, korrektse andmete korral, suurte andmemahtude korral jne. Vigade leidumisel viidi rakendusse sisse muudatused.

Lisaks käsitsi testimisele viidi 26. aprillil läbi testimälumäng loodud rakenduse abiga. See hõlmas väikest kinnist seltskonda, kuna antud olukorraga Eestis ei tohi suurtele gruppidele üritusi korraldada. Üks osalistest oli mängu läbiviija ja teised mängisid mälumängu. Töö autor üritas protsessi käigus võimalikult vähe abistada, et näha, kas rakendust on võimalik kasutada ka iseseisvalt. Kõik osalejad said kasutaja luua ja mälumänguga liituda ning korraldajal suuremaid probleeme ei esinenud.

Tagasiside oli enamasti positiivne, kuna kõik toimis üsna kiiresti. Negatiivse tagasisidena mainiti, et rakendus võiks visuaalselt ilusam ja modernsem olla. Lisaks veel, et mälumängu loomine rakenduses võiks olla sujuvam ning ka mõned väiksemad probleemid. Tagasiside alusel on hea rakenduse arendamist jätkata ning seda parandada.

8 Kokkuvõte

Käesoleva töö eesmärgiks oli luua lihtne rakendus mälumängude korraldamiseks. Töö autor ei loonud rakendust lähtudes kasumi teenimisele, vaid loodab selle abil muuta mälumängu huvilistele ja korraldajatele ning ka iseendale elu lihtsamaks, võimaldades luua intuitiivseid mälumänge, mida on võimalik ka pärast toimumist kõikidel osalejatel üle vaadata.

Lõputöö eesmärgi täitmiseks loodi põhjalik analüüs. Selle käigus uuriti olemasolevaid lahendusi, mille puuduste ja eeliste alusel sai määrata loodava rakenduse funktsionaalsuse ja skoobi. Lisaks loodi ka analüüs erinevatele tehnoloogiatele, mida süsteemi arendusel kasutada. See tagas selle, et loodav rakendus oleks jätkusuutlik ning seda oleks võimalik edasi arendada.

Loodud rakendus on väga hea põhi edasiarenduseks. See vajaks veel palju tööd välimuse ja disaini poole pealt, kuna töö autoril puudub suurem kogemus veebidisainis. Lisaks sellele sooviks autor tulevikus lisada rakendusele funktsionaalsust ning parandada ka olemasolevat.

Kasutatud kirjandus

[1] „MäluMäng,“ [Võrgumaterjal]. Available: https://www.malumang.ee/

[2] M. Weimer, „Multiple-Choice Tests: Revisiting the Pros and Cons,“ 21 veebruar 2018. [Võrgumaterjal]. Available: https://www.facultyfocus.com/articles/educational-assessment/multiple-choice-tests-pros-cons/

[3] „Quiz-Maker,“ [Võrgumaterjal]. Available: https://www.quiz-maker.com/

[4] „Online Visual Paradigm,“ [Võrgumaterjal]. Available: https://online.visual-paradigm.com/

[5] CodeAcademy, „Back-End Web Architecture,“ [Võrgumaterjal]. Available: https://www.codecademy.com/articles/back-end-architecture

[6] Altexsoft, „Comparing API Architectural Styles: SOAP vs REST vs GraphQL vs RPC,“ 29 mai 2020. [Võrgumaterjal]. Available: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/

[7] Altexsoft, „What is SOAP: Formats, Protocols, Message Structure, and How SOAP is Different from REST,“ 20 august 2019. [Võrgumaterjal]. Available: https://www.altexsoft.com/blog/engineering/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/

[8] Restfulapi, „What is REST“ [Võrgumaterjal]. Available: https://restfulapi.net/

[9] Altexsoft, „GraphQL: Core Features, Architecture, Pros and Cons,“ 23 märts 2019. [Võrgumaterjal]. Available: https://www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/

[10] Pyhton Documentation, „What is Python? Executive Summary,“ [Võrgumaterjal]. Available: https://www.python.org/doc/essays/blurb/

[11] D. Kumar, „Why is Python slower than other languages?,“ 4 jaanuar 2019. [Võrgumaterjal]. Available: https://www.tutorialspoint.com/why-is-python-slower-than-other-languages

[12] J. Clark, „Top 10 backend frameworks, Django,“ [Võrgumaterjal]. Available: https://blog.back4app.com/backend-frameworks/

[13] S. Bhatt, „Pros and Cons of Django Framework for App Development,“ 31 august 2020. [Võrgumaterjal]. Available: https://dzone.com/articles/pros-and-cons-of-django-framework-for-app-developm

[14] M. Poczwardowski, „Django Framework Review - Pros and Cons,“ 21 juuni 2020. [Võrgumaterjal]. Available: https://www.netguru.com/blog/django-pros-and-cons

[15] TutorialsTeacher, „.NET Core Overview,“ [Võrgumaterjal]. Available: https://www.tutorialsteacher.com/core/dotnet-core

[16] Ukad Group, „ASP.NET Core 8 Pros and 3 Cons,“ [Võrgumaterjal]. Available: https://www.ukad-group.com/blog/aspnet-core-8-pros-and-3-cons/

[17] Official Ruby Website, „About Ruby,“ [Võrgumaterjal]. Available: https://www.ruby-lang.org/en/about/

[18] Codica Team, „Pros & Cons of Using Ruby on Rails for Web Development,“ 2 märts 2020. [Võrgumaterjal]. Available: https://medium.com/codica/pros-cons-of-using-ruby-on-rails-for-web-development-b973f0dcfcf8

[19] J. Clark, „Top 10 backend frameworks, Ruby,“ [Võrgumaterjal]. Available: https://blog.back4app.com/backend-frameworks/

[20] Mozilla official documentation, „JavaScript Introduction,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

[21] NodeJS official documentation, „Introduction to Node.js,“ [Võrgumaterjal]. Available https://nodejs.dev/learn/introduction-to-nodejs

[22] K. Malvi, „The Positive and Negative Aspects of Node.js Web App Development,“ 1 juuni 2018. [Võrgumaterjal]. Available https://www.mindinventory.com/blog/pros-and-cons-of-node-js-web-app-development/

[23] M. Demchenko, „Review of Node.JS: Pros and Cons,“ 2 november 2020. [Võrgumaterjal]. Available https://ncube.com/blog/review-of-node-js-pros-and-cons

[24] CoreJavaGuru, „Java Overview,“ [Võrgumaterjal]. Available: http://www.corejavaguru.com/java/basic/overview

[25] SCAND, „Pros and Cons of Using Spring Boot,“ 26 juuni 2020. [Võrgumaterjal]. Available: https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/

[26] JavaTPoint, „Java Spring Pros and Cons,“ [Võrgumaterjal]. Available: https://www.javatpoint.com/java-spring-pros-and-cons

[27] T. Watson, „Java Spring Framework – Pros, Cons, Common Mistakes,“ 1 november 2019. [Võrgumaterjal]. Available: https://skywell.software/blog/java-spring-framework-pros-cons-mistakes/

[28] TutorialsPoint, „PostgreSQL - Overview,“ [Võrgumaterjal]. Available: https://www.tutorialspoint.com/postgresql/postgresql_overview.htm

[29] M. Drake and unknown user, „SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems,“ 19 märts 2019. [Võrgumaterjal]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

[30] SQLiteTutorial, „What is SQLite?,“ [Võrgumaterjal]. Available: https://www.sqlitetutorial.net/what-is-sqlite/

[31] TutorialsPoint, „MariaDB - Introduction,“ [Võrgumaterjal]. Available: https://www.tutorialspoint.com/mariadb/mariadb_introduction.htm

[32] S. Bocetta, „Comparing 3 open source databases: PostgreSQL, MariaDB, and SQLite,“ 15 jaanuar 2019. [Võrgumaterjal]. Available: https://opensource.com/article/19/1/open-source-databases

[33] W3Schools, „MongoDB - Overview,“ [Võrgumaterjal]. Available: https://www.w3schools.in/mongodb/overview/

[34] R. Cookson, „The Pros and Cons of MongoDB,“ 8 august 2019. [Võrgumaterjal]. Available: https://www.virtual-dba.com/blog/pros-and-cons-of-mongodb/

[35] Microsoft, „Visual Studio 2019,“ [Võrgumaterjal]. Available: https://visualstudio.microsoft.com/vs/

[36] Microsoft Docs, „Why did we build Visual Studio Code?,“ 31 märts 2021. [Võrgumaterjal]. Available: https://code.visualstudio.com/docs/editor/whyvscode

[37] JetBrains, „Rider - fast & powerful cross-platform .NET IDE,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/rider/

[38] R. Perez, „Visual Studio versus Rider,“ 12 märts 2018. [Võrgumaterjal]. Available: https://stackify.com/visual-studio-rider/

[39] TypeScript Docs , „What is TypeScript?,“ 2021. [Võrgumaterjal]. Available: https://www.typescriptlang.org/

[40] S. Daityari, „Angular vs React vs Vue: Which Framework to Choose in 2021,“ 15 märts 2021. [Võrgumaterjal]. Available: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

[41] A. Pattakos, „Angular vs React vs Vue 2021,“ 25 jaanuar 2021. [Võrgumaterjal]. Available: https://athemes.com/guides/angular-vs-react-vs-vue/

[42] Microsoft Docs, „Solutions and projects in Visual Studio,“ 31 detsember 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/visualstudio/ide/solutions-and-projects-in-visual-studio?view=vs-2019

[43] TutorialsPoint, „What is Entity Framework?,“ 31 detsember 2020. [Võrgumaterjal]. Available: https://www.tutorialspoint.com/entity_framework/entity_framework_overview.htm

[44] Packt Editorial Staff, „What is a multi layered software architecture?,“ 17 mai 2018. [Võrgumaterjal]. Available: https://hub.packtpub.com/what-is-multi-layered-software-architecture/

[45] F. Copes, „JSON Web Token (JWT) explained,“ 13 detsember 2018. [Võrgumaterjal]. Available: https://flaviocopes.com/jwt/

[46] Swagger Docs, „What Is Swagger?,“2021. [Võrgumaterjal]. Available: https://swagger.io/docs/specification/2-0/what-is-swagger/

[47] A. Ouellette, „What is Bootstrap: A Beginner's Guide,“ 26 märts 2021. [Võrgumaterjal]. Available: https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/

[48] W3schools, „jQuery Introduction,“ [Võrgumaterjal]. Available: https://www.w3schools.com/jquery/jquery_intro.asp

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

Mina, Robert Tõnisson

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

16.05.2021

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

Lisa 2 – Klientrakenduse ja serveripoolse rakenduse versioonihaldus

Front-end:

https://github.com/roberttonisson/QuizApplications/tree/main/client-quiz-app

Back-end:

https://github.com/roberttonisson/QuizApplications/tree/main/QuizOrganizeSolution