TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Erko Radik 185980IACB

SERVERI RAKENDUSE LOOMINE TEADMISTE JAGAMISE PLATVORMI JAOKS

Bakalaureusetöö

Juhendaja: Nadežda Furs
MBA

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: Erko Radik

13.05.2021

Annotatsioon

Käesoleva lõputöö eesmärk on välja töötada serveri rakendust teadmiste jagamiste platvormi jaoks ehk esimese iteratsiooni realiseerimine selleks, et ehitada alust edasi arendamiseks.

Antud töös otsustakse, milline rakenduse arhitektuur vastab esitatud nõutele, kas parem kasutada klient/server mudeli andmete opereerimiseks ning milline protokoll on parem kasutada andmete edastamiseks.

Samal ajal toimus tehnoloogiline lahenduse valik serveri rakenduse kirjutamise jaoks. Alguses oli kavas valida juba olevate *tech-stack* mustritest, kui analüüsi käigus oli otsustatud valida eraldi iga komponent, kui mitte üks täis tehnoloogiline lahendus ei vastanud esitatud nõuetele.

Tehnoloogiliste lahenduse analüüs toimus teoreetiliste andmete põhjal. *Back-end* lahenduse kirjutamiseks oli tähtis programmimiskeele arhitektuur ehk programmerimiskeele töötamise viis ja mitmelõimeline töötlus.

Pärast raamastiku valikust oli korraldatud andmebaaside analüüs selleks, et otsustada kas *SQL* põhiline või *noSQL* põhiline andmebaas vastab rohkem püstitatud ülesandele.

Viimaseks oli tehtud serveri osa realisatsioon, *API* kirjutamine tuleviku kasutamisjuhtude jaoks ja kirjatud funktsionaali testimine vastavalt kasutajalugude kirjandusele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 38 leheküljel, 8 peatükki, 9 joonist, 4 tabelit.

Abstract

Creation of server application for knowledge-sharing platform.

This thesis aims to develop a server application for a knowledge-sharing platform. That is to implement the first iteration to build a foundation for further platform development.

In this thesis, it is determined which application architecture meets the established requirements, whether it is better to use the client/server model to manage the data transfers, and which protocol is more suitable to use for data transmission.

At the same time, a tech stack was chosen for writing a server application. The initial course of action was to choose from the pre-existing tech stack. However, during the analysis, it was decided to select each component separately as not one complete tech-stack solution did meet the established requirements.

The analysis of the tech-stacks was conducted using theoretical data. For the development of the back-end, programming language architecture was of utmost importance. Crucial parts include how programming language operates and the support of multi-threading.

After the selection of the framework was completed, it was time to analyse databases to decide whether SQL or NoSQL based databases suits the established goals.

As the final note, the server part of the project was complete. API was written for future use cases. Drafted functionality has gone through testing following the User-Stories description.

The thesis is in Estonian and contains 38 pages of text, 8 chapters, 9 figures, 4 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
Back-end	tagaprogramm
Boot	Alglaadimine
BSD	*Berkeley Systems Distribution*
CORBA	Common Object Request Broker Architecture, Ühine objekti päringu maakleri arhitektuur
DCOM	*Distributed Component Object Model,* hajus-komponentobjektimudel
Front-end	eessüsteem
IDE	*Integrated development environment*, integreeritud programmeerimiskeskkond
JSON	*JavaScript Object Notation.* JavaScripti Objekti esitus
Linux	*Linuxi* tuumal põhinevad operatsioonisüsteemid, näiteks *Debian*, *CentOS*, *RedHat Enterprise*
MOOC	*Massive open online course,* Massiivne avatud online kursus
MVC	*Model–view–controller,* Mudeli vaade kontroller
MVP	*Minimum viable product, Minimaalne* elujõuline toode
noSQL	*"not only SQL",* mitte ainult SQL
ORM	*Object–relational mapping,* Objekti relatsioonilane kujutus
RPC	*Remote Procedure Call*, kaugprotseduurikutse
RunTime	Käitusaegne
SDLC	*Systems development life cycle,* Süsteemi arenduse etapid

SQL	*Structured Query Language*, Struktuurpäringukeel
Tech-Stack	Tehnoloogiline lahendus
XML	*Extensible Markup Language*, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Hariduse ja igapäevaste oskuste küsimus oli alati populaarne, aga *COVID-19* pandeemia ajal veelgi tähtsam. Praegune situatsioon nõuab võimalust arendada nii professionaalseid oskusi kui ka teadmisi kuidas parandada oma arvuti, joota kondensaatoreid jne. Mõned küsimused vajavad ainult paar tekstilõiku vastamiseks, teised aga sõltuvalt küsimuste komplekssusest vajavad suuremat hulka selgitusi. *MOOC* platvormid eksisteerisid ammu enne pandeemia algust, kuid nende praegune olemasolu on elutähtis ning kasulik. Juba 2002. aastal hakkas esinema maailma võrgus loengute salvestused *OpenCourseWare* projekti raames. [1]

Praegu enamus olemasolevatest kursustest on kas päris eriala põhilised või on samal tasandil ülikooli kursustega. Sellest ilmub võimalik arenduse vektor *startupi* arendamiseks. Oli märgitud probleem, et puuduvad lihtsamad võimalused oskuste ja teadmiste saamiseks. Samas puudub võimalus professionaalidelt kergesti vastust saada.

Selle probleemi lahendamiseks oli otsustatud luua uut platvormi, kus kasutajad saaksid lihtsal viisil küsimusi esitada, ning ka samal platvormil osaleda kursustel või töötubadel. Projekti eesmärgiks on ka anda võimalust vabaks suhtlemiseks osalejate ja mentori vahel, mis enamustel MOOC esindajatel tänapäeval puudub.

Ülalpool toodud kirjelduse põhjal, autor püstitas järgmist eesmärki: serveri rakenduse loomine teadmiste jagamiste platvormi jaoks. Laimelt kirjeldatakse järgmistes peatükis.

1.1 Ülesanne püstitus

Selle töö ülesandeks on vajaliku tööriistade, raamastiku ja andmebaasi valik serveri rakenduse kirjutamiseks ja esimese iteratsioon realisatsioon.

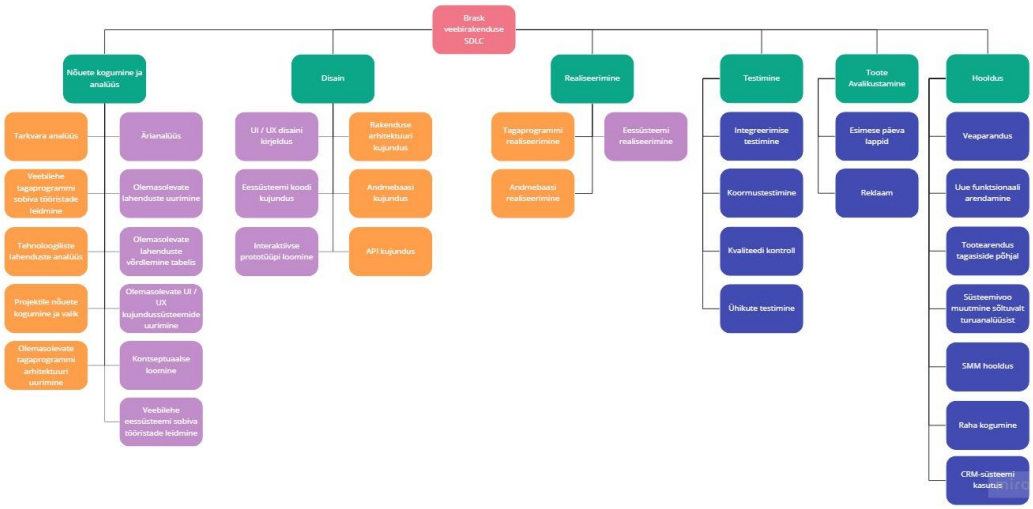
Sotsiaalne võrgustik teadmiste vahetamiseks on uus idee, mis pakutakse välja Stefan Kuzmini (Tallinna Tehnikaülikool, IT süsteemide arendus eriala, 186078IADB) töös. See lõputöö on tihedalt seotud ülalpool toodud tööga, kust autor võtab põhilisi tingimusi oma töö kirjutamiseks.

Selles töös autor tegeleb *tech-stack*'i analüüsiga ja valimisega, vastavalt ettemääratud nõutele. *Back-end* ja serveri osa loomine teadmiste jagamiste platvormi *startupi'*i jaoks.

Rakenduse serveri osa loomine toimub esimese iteratsiooni raames. Mis on kirjeldatud *EPIC* ja kasutajalugude põhjal. Ainult kasutajalood omandavad prioriteeti omadust, sest *EPIC's* kirjeldavad veebilehe funktsionaalsust ehk skeletti. Andmebaasi realisatsioon ja ühendus. Vaata lähemalt peatükis 2.1

1.2 Töö skoop ja rollide jaotus

Töö jagamiste jaoks oli loodud *SDLC* skeem, mis kirjutab projekti kirjutamise etappide nimetusi ning millised ülesanded toimuvad igas etapis.



Joonis 1. Töö jaotus *SDLC* faaside järgi

Tumeroheline – arenduse etappide nimetused

Oranž – serveri ja andmebaasiga seotud punktid, mis autori jaoks oli vajalik sooritada lõputöö kirjutamise jooksul.

Orhidee – *front-end* ja ärindusega seotud punktid, mis sooritas minu kolleeg.

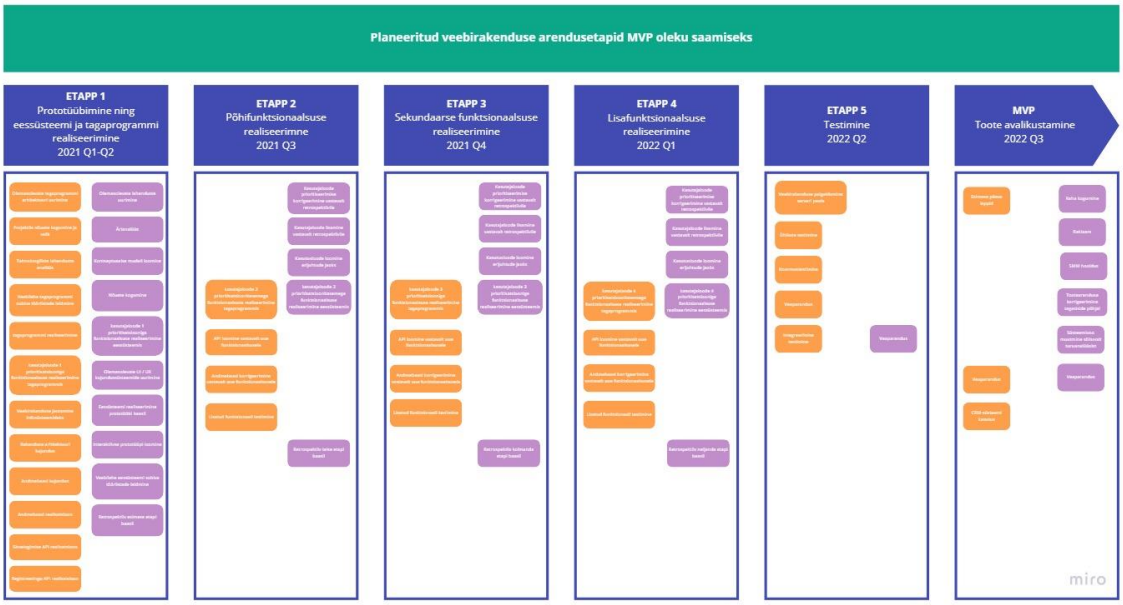
Tumesinine – punktid, mis on kavas sooritada pärast diplomitöö kaitsmist.

(*Out of scope*)

Esitatud skeem kirjutab töö jaotust *SDLC*(*Software Development Life Cycle*) faaside järgi, mis on ISO / IEC / IEEE 12207: 2017 [4] standardi raames ja põhineb *SDLC* definitsioonil [4]. Täisskeem on nähtav Lisas 2. Skeem oli tehtud Stefan Kuzmini 186078IADB lõputöö raames. Antud skeem on esitatud autori töös selle jaoks, et demonstreerida tööjaotutust.

Skeemi kasutakse detailseks tööjaotamiseks rakenduse arendamisel. Antud skeem näitab üldist töömahtu, mis on vajalik *MVP* seisundi jõudmiseks. (Selle kohta põhjalikult on kirjeldatud Stefan Kuzmini 186078IADB töös). Mitte kõik skeemi pesi realiseeritakse selle lõputöö raames. Täpsemate töö protsesside jaotuse jaoks oli valmistatud skeem (Joonis 2) Stefan Kuzmini 186078IADB lõputöö raames. Mis näitab rakenduse arendamise protsessid.

Kaks esimest *SDLC* faasi valmistatakse lõputöö jooksul vastavalt tööjaotusele (Oranžiga märgitud pesad on selle lõputöö autori kohus). Töömaht, mis on esitatud kolmas faasis on jagatud etappideks. Ülalpool on toodud tabel, mille põhiline analüüs toimub Stefan Kuzmini 186078IADB lõputöös. Aga selle skeemi demonstratsioon on tähtis autori töös sest, et skeem konkreetselt kirjutab töömahtu, mis oli läbitud autori lõputöö raames.



Joonis 2. Arenduse etappide skeem

Antud skeem näitab vajalike etappide loetelu ettemääratud ajavahemiku piires. Põhilisem analüüs on selle skeemi looja töös. Lõputöö raames autor sooritas esimese etappi esitatud skeemis. (Joonis 2) Täisskeem on nähtav Lisas 3.

2 Arhitektuurilahenduse arendus vastavalt olemasolevate nõuetele

Selles peatükis toimub probleemi püstitus, ülevaade projekti vajalikkude infosüsteemidest. Tuuakse välja põhjusi, miks iga infosüsteem on iseseisev oma funktsionaalsuses ning toimub sissejuhatus võimaliku rakenduse arhitektuuridest, suhtlemise mudelist ja juurdepääsu *API*-st.

2.1 Probleemi püstitus nõuete põhjal

Selles diplomitöös käsitletavaks põhiliseks probleemiks on sobivate tehniliste lahenduste leidmine serveri veebirakenduse arendamiseks ja esimese iteratsiooni veebilehe versiooni realisatsioon. Antud töös ei luua *MVP* produkti versiooni, sest et antud olukorras autor ei leidnud võimalust ega õiget *MVP* versiooni rakenduse realiseerimiseks.

Projekti raames kirjutatud *EPIC*'ud näitavad, millised veebilehed on vajalikud produkti realiseerimiseks. Mõned lehed on omavahel sarnased, aga nende funktsionaalsus sõltub sellest kas kasutaja on postituse looja või vastaja. *EPIC*'ute kirjutamiseks oli kasutatud *Atlasse* [7] juhiseid. Allpool on toodud näidis, kuidas *EPIC*'ud näevad välja.

Tabel 1. *EPIC*'ute tabeli näidis

Epic ID	**Epic name**
EP_1	Veebilehele sisse logimine
EP_2	Registreerimine veebilehel
EP_3	Esileht

Täistabel on toodud Lisas 4.

Kasutajalugude kirjutamise eesmärk on abstraktselt kirjeldada kuidas iga veebileht peaks funktsioneerima. Abstraktsus seisab selles, et see pole sõna-sõnalt kirjeldus, kuidas iga funktsioon töötab, aga selle piisavalt täpne eesmärk, mida on vajalik saavutada. Kasutajalugude kirjutamiseviisi alus on *Atlasse* [8] juhised.

Tabel 2. Kasutajalugude tabeli näidis

Epic ID	User Story ID	Priority	As a [persona]	I [want to]	[so that]
EP_1	US_1	1	Kasutaja	Sisse logida e-posti ja parooli abil	Mul on juurdepääs oma isikuandmetele
EP_1	US_2	2	Kasutaja	Sisse logida, kasutades Google'i sisselogimist	Ma pääsen oma isikuandmetele juurde Google API abil
EP_1	US_3	2	Kasutaja	Sisse logida, kasutades Facebooki sisselogimist	Ma pääsen oma isikuandmetele juurde Facebooki API kaudu

2.2 Projekti jaoks vajalike infosüsteemide ülevaade

Selles peatükis kirjeldatakse, mille viisi järgi peaks olema eraldatud selle rakenduse funktsionaalsust. Nagu oli kirjeldatud (Vaata peatükk 1.2 tööjaotust, suurem osa funktsionaalsusest hakkab realiseerima mitte selle lõputöö kontekstis.)

- Kasutaja tuvastamise infosüsteem
- Kursusele registreerimise infosüsteem
- Kursuse loomise ja administreerimise infosüsteem
- Suhtlemiskeskkonna infosüsteem
- Hinnangu ja tagasiside andmise infosüsteem
- Populaarsuse hinnangu infosüsteem
- Sertifikaadi välja andmise infosüsteem
- Teadete infosüsteem

- Maksetöötluse ja maksesooritamise infosüsteem
- Töötubade otseülekanne infosüsteem

2.3 Infosüsteemide eraldamise põhjus

Nende infosüsteemide eraldamise viisi põhjus seisneb selles, et iga ülapool toodud infosüsteem on iseseisev ja sõltumatu teistest infosüsteemidest funktsioneeriv. Näiteks kui tehniliselt teadete ja populaarsuse infosüsteemid vajavad informatsiooni, mis tuleb teisest infosüsteemist, tööpiirkond on kõikidel infosüsteemidel omapärane. Ehk teadete infosüsteem ainult edastab sõnumit sertifikaadi saamise kohta.

2.4 Olemasolevate arhitektuurilahenduste võrdlemine ja analüüs

Tänase päeva seisuga maailmas on olemas erinevad arhitektuursed lahendused antud töös käsitletava probleemi lahendamiseks nagu: *Monolithic* arhitektuur, *Microservices* arhitektuur ja *Onion* arhitektuur. Igal arhitektuuril on oma eeldused ja puudused, mis põhinevad sellest, kuidas toimub andmevahetus rakenduse kihtide vahel.

Töös kasutatakse objektorienteeritud programmeerimise printsiip, kuna struktuurset programmerimist pole mõtet kasutada antud tehnilise rakenduse jaoks. Põhjuseks on see, et objektorienteeritud programmeerimine annab võimalust kasutada sama koodi erinevate ülesannete lahendamiseks ja kirjutada iga protsessi omadusi ja funktsionaalsust.

2.4.1 Arhitektuuri lahenduse valiku kriteeriumid

Arhitektuuri valiku kriteeriumid põhinesid järgmistel kriteeriumitel: võimalus eraldi arendada rakenduse osasid teiste rakendu osade mõjutamata ehk ilma kogu rakenduse koodi ümber kirjutamiseta. Veebileht peab olema keskendatud informatsioonile allikas, kust iga kasutaja saab enda jaoks vajaliku informatsiooni saada.

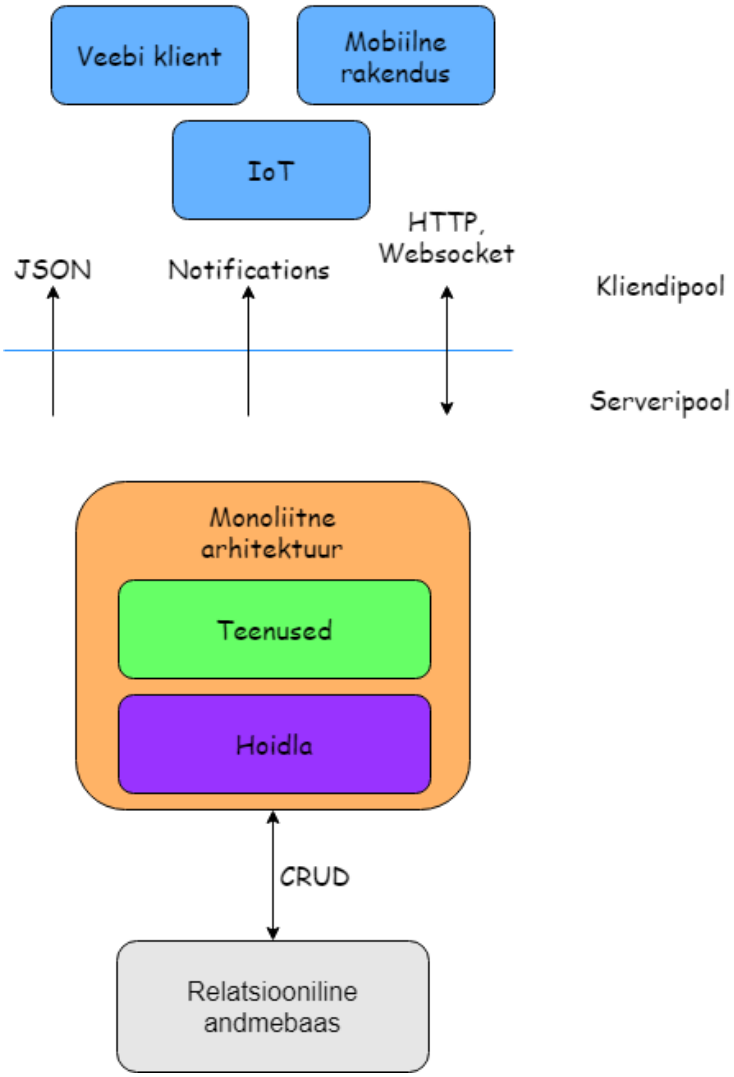
2.4.2 Olemasolevate arhitektuuri lahenduste ülevaade

Monoliitne arhitektuur:

Monoliitne arhitektuur on tarkvaraprogrammi kujundamise traditsiooniline ühtne mudel. Monoliitne tähendab selles kontekstis kõik ühes tükis koosnevat.

Monoliitne tarkvara on loodud iseseisvaks. Programmi komponendid on omavahel ühendatud ja üksteisest sõltuvad, mitte lõdvalt seotud nagu moodulprogrammide puhul. Tihedalt ühendatud arhitektuuris peavad kõik komponendid ja nendega seotud komponendid olema kohal, et kood saaks käivitatuks või kompileerituks.

Kui programmi mõnda komponenti tuleb värskendada, siis tuleb kogu rakendus ümber kirjutada. Monoliitsel arhitektuuril on mõjutavad eelised ka. Monoliitsed programmid on tavaliselt parema läbilaskevõimega kui modulaarsed, näiteks mikroteenuste arhitektuur (*MSA*). Samas neid saab kergelt testida ja siluda, sest vähemate elementide korral on vähem muutujaid, mis tulevad mängu. Allpool on toodud joonis, mis kirjutab monoliitse arhitektuuri tööprintsi. Joonise peal on nähtav, et ainult monoliitse rakenduse läbi toimub andmevahetus kliendi poolega ning andmebaasiga. Ehk andmevool on üks kindel tee, mis saab muutuda ainult siis, kui rakenduse ümber kirjutada. Vaata joonis 3.



Joonis 3. Monoliitne arhitektuur

Mikroteenuste arhitektuur

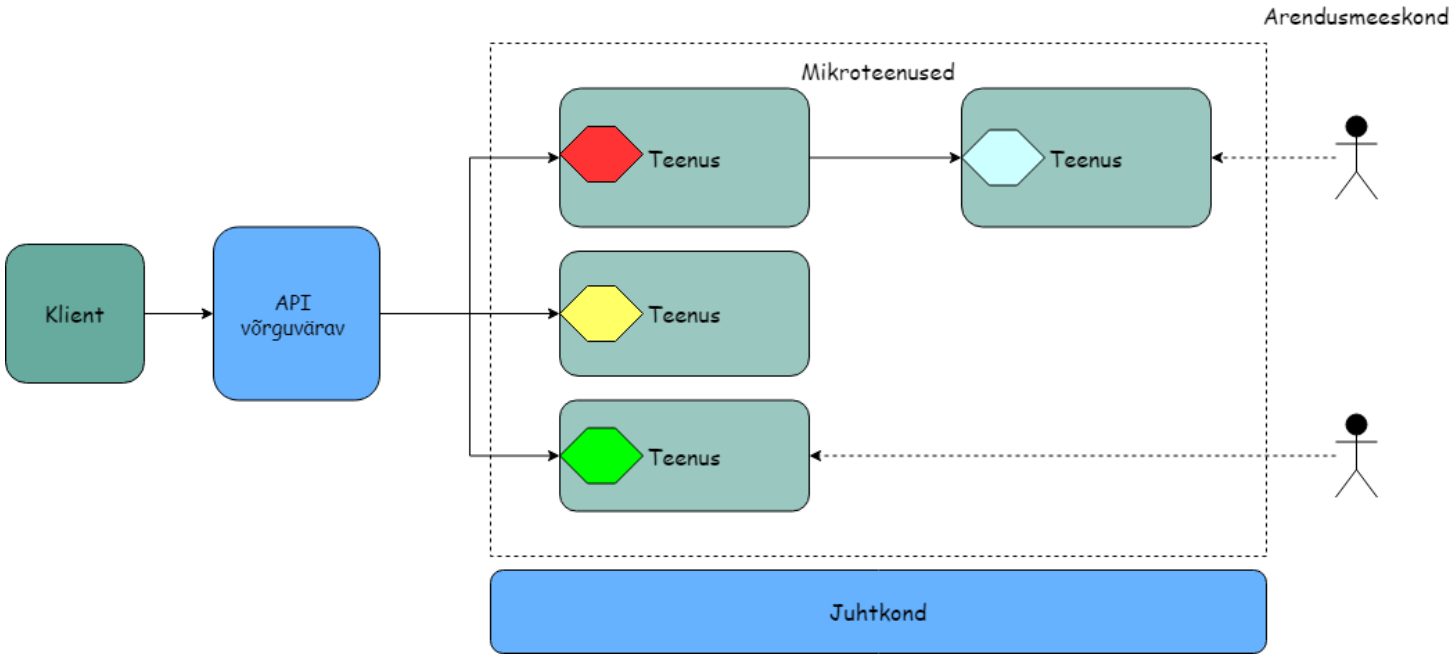
Mikroteenuste arhitektuur koosneb väikeste autonoomsete teenuste kogumist. Iga teenus on iseseisev ja peaks rakendama ühte ärivõimalust piiratud kontekstis. Piiratud kontekst on loomulik jagunemine ettevõtte sees ja annab selgesõnalise piiri, milles domeenimudel eksisteerib.

Ülalpool autor toob mikroteenuste arhitektuuri omadusi:

- Mikroteenused on väikesed, sõltumatud ja lõdvalt seotud. Väike arendajate meeskond saab teenust kirjutada ja seda halata.
- Iga teenus on eraldi koodibaas, mida saab toetada väike arendusmeeskond.
- Teenuseid saab juurutada iseseisvalt. Meeskond saab olemasolevat teenust värskendada ilma kogu rakendust ümber ehitamiseta ja ümber paigutamiseta.
- Iga teenus vastutab oma andmete või välise oleku säilitamise eest. See erineb traditsioonilisest mudelist, kus andmete püsivusega tegeleb eraldi andmekiht.
- Toetab polüglottide programmeerimist. Näiteks ei pea teenused jagama sama tehnoloogiat, raamatukogusid ega raamistikke.

Moodulrakenduses saab eraldi moodulit (näiteks mikroteenust) muuta programmi teiste osade mõjutamata. Modulaarsed arhitektuurid vähendavad riski, et ühe elemendi sees tehtud muudatus tekitab teiste elementide sees ootamatuid muudatusi, kuna moodulid on suhteliselt sõltumatud. Moodulprogrammid sobivad iteratiivsetele protsessidele ka kergemini kui monoliitsed programmid.

Allpool on toodud mikroteenuste arhitektuuri tööprintsiipi joonis. See joonis kirjutab mikroteenuste arhitektuuri andmevoogu. Klient suhtleb *API*-ga, mis kontrollib milline teenust peab töötama antud andmeteega. Võrreldes monoliitse arhitektuuriga, andmesuund pole kindalasti määratud. Vaata joonis 4.



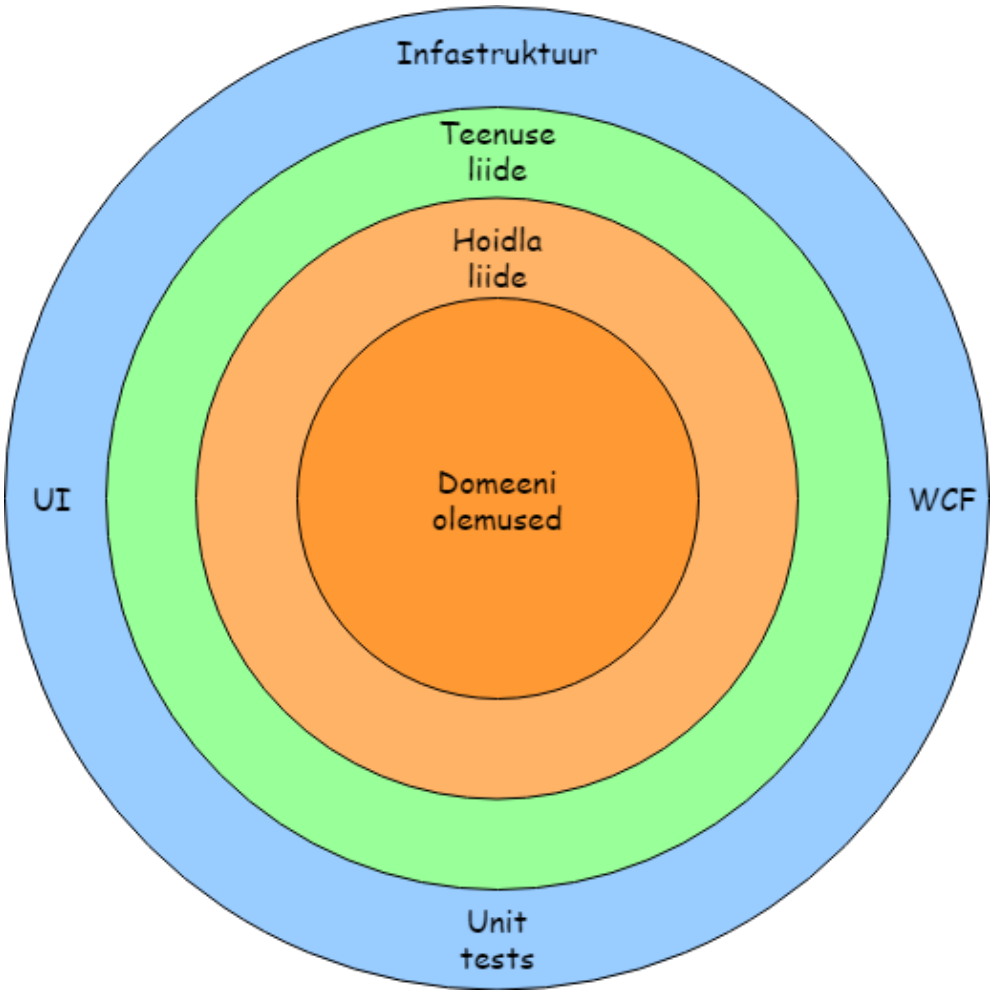
Joonis 4. Mikroteenuste arhitektuur

Onion arhitektuur või sibulaarhitektuur

Sibulaarhitektuur põhineb juhtimispõhimõtte inversioonil. Sibulaarhitektuur koosneb mitmest konkreetsest kihist, mis ühendavad üksteist domeeni esindava südamiku suunas. Arhitektuur ei sõltu andmekihist nagu klassikalistes mitmetasandilastes arhitektuurides, vaid tegelikest domeenimudelitest.

Traditsioonilise arhitektuuri kohaselt suhtleb kasutajaliidese kiht äriloogikaga ja äriloogika edastab andmeid andmekihile ning kõik kihid asuvad segamini ja sõltuvad üksteisest. Kolmeastmeline ja n-taseme arhitektuuri korral pole ükski kihtidest sõltumatu. Selliseid süsteeme on väga raske mõista ja halata. Selle traditsioonilise arhitektuuri puuduseks on tarbetu ühendamine.

Allpool toodud joonis näitab sibulaarhitektuuri tööprintsiipi. Sellises arhitektuuris andmevoog toimub väljastpoolt sissepoole. Antud arhitektuuris objektid on tähtsamad olemasolud, mis lähendab mõned probleemid tavaliste arhitektuuriga, kui esinevad omapärased probleemid ka, nagu kasutajaliides laiem kasutus. Vaata joonis 5.



Joonis 5. Sibulaarhitektuur kihid

Sibulaarhitektuur lahendab neid probleeme määratledes kihte südamikust infrastruktuurini. Antud struktuur rakendab põhireeglit, liigutades kogu siduri keskpunkti poole.

See arhitektuur on kahtlemata objektorienteeritud programmeerimise suhtes kallutatud ja see asetab objektid kõigist teistest ette. Sibulaarhitektuuri keskmes on domeenimudel, mis esindab äri- ja käitumisobjekte. Domeenikihi ümber on muud kihid, millel on rohkem käitumisviise.

2.4.3 Hostide võimalikud vastastikmõjud

Klient-server mudel

Kliendi-serveri arhitektuur on arvutusmudel, milles server majutab, edastab ja haldab enamikku kliendi tarbitavatest ressursse ja teenuseid. Seda tüüpi arhitektuuril on üks või mitu klientarvutit, mis on võrgu kaudu ühendatud keskserveriga,

Kliendi-serveri võrk hõlmab mitut klienti ehk tööjaama, mis on ühendatud vähemalt ühe keskserveriga. Enamik andmeid ja rakendusi on paigaldatud serverisse. Kui kliendid vajavad juurdepääsu nende ressurssideni, pääsevad nende juurde serveri kaudu. Serveritel on sageli nii erakasutajate kataloogid kui ka mitmed avalikud kataloogid. Kliendi-serveri võrkudel on tavaliselt suurem juurdepääsukiirus, kuna need on mõeldud suure andmevoogu toetamiseks. Tarkvarasid ja faile on lihtsam uuendada, kuna neid hoitakse ühes arvutis. Kogu süsteemi hõlmavaid teenuseid saab pakkuda serveritarkvara kaudu.

Peer-to-Peer mudel

Peer-to-peer arhitektuur (või *P2P* arhitektuur) on kõige sagedamini kasutatav arvutivõrgu arhitektuur, milles igal tööjaamal või sõlmel on samad võimalused ja vastutus. *Peer-to-peer* võrgud kaasavad kahte või enamat arvutit, mis kasutavad individuaalseid ressursse .Need jagatud ressursid on saadava kõikidele võrgu arvutitele. Iga arvuti toimib nii kliendi kui ka serverina, suheldes otse teiste arvutitega. Näiteks, *Peer-to-peer* võrgus olev arvuti saab kasutada teistel arvutil oleva printeri

P2P-võrkudel on palju rakendusi, kuid kõige levinum on sisu levitamine. See hõlmab tarkvara avaldamist ja levitamist, sisu edastusvõrke, voogedastusmeediumit ja multiedastust voogude edastamist, mis hõlbustab tellitavat sisu edastamist.

2.4.4 Veebirakenduse juurdepääsu protokollid

SOAP

SOAP tugineb sõnumside teenuste pakkumisel ainult *XML*-i. Alguses *Microsoft* töötas välja *SOAP*-i, et asendada vanemaid tehnoloogiaid, mis Internetis hästi ei toiminud, näiteks *DCOM* ja *CORBA*. Need tehnoloogiad ei õnnestunud, kuna nad toetuvad binaarsõnumitele. *SOAP*-is kasutatav *XML*-sõnumside töötab Internetis paremini.

Nõrgus on selles, et *SOAP* on väga laiendatav kui arendaja kasutab ainult konkreetse ülesande jaoks vajalikke tükke. Näiteks kui kasutate avalikku veebiteenust, mis on kõigile vabalt kättesaadav, pole teil *WS*-Security järele suurt vajadust.

SOAP-is taotluste esitamiseks ja vastuste saamiseks kasutatav *XML* võib muutuda äärmiselt keerukaks. Mõnes programmeerimiskeeles arendaja peab neid taotlusi käsitsi koostama, mis muutub problemaatiliseks, kuna *SOAP* ei talu vigu.

SOAP eelised on:

1. Standarditud
2. Sisseehitatud tõrkeotsing
3. Keele, platvormi ja transpordi sõltumatus (*REST* nõuab *HTTP* kasutamist)
4. Toimib hästi hajutatud ettevõttekeskkondades (*REST* eeldab otsest punkt-punkt-suhtlust)
5. Pakub märkimisväärset eelehitatud laiendatavust *WS* standardite näos.
6. Automatiseeritud, kui selle jaoks kasutatakse teatud programmeerimiskeeli.

REST

REST pakub kergema kaaluga alternatiivi. Paljud arendajad leidsid, et *SOAP* on tülikas ja seda on raske kasutada. Näiteks, see tähendab, et *JavaScripti* abil *SOAP*-iga töötamine nõuab hulga koodi kirjutamist lihtsate ülesannete täitmiseks, sest peate iga kord looma vajaliku *XML*-struktuuri.

Selle asemel, et iga päringu jaoks teha *XML*-i, tugineb *REST* (tavaliselt) lihtsale *URL*-ile. Mõnes olukorras peate esitama lisateavet, kuid enamik *REST*-i kasutavaid veebiteenuseid tugineb ainult *URL* lähenemisviisile.

REST eelised:

1. Sile õppimiskõver
2. Kiire (ei nõua põhjalikku töötlemist)
3. Veebiteenusega suhtlemiseks pole vaja kalleid tööriistu
4. Lähemal muudele veebitehnoloogiatele disainifilosoofias

2.5 Arhitektuurilahenduse valiku põhjendus

Kui plaaneritud veebirakendus ei nõua suurt arvutamisevõimsust ja on kavas edasi arendada uut funktsionaalsust iteratiivselt, valisime Mikroteenuste põhilise arhitektuuri. Juurdepääsu protokolli valik on *Rest API*, põhjuseks on kasutusmugavus ja parem õppimiskõver, kui ka võimalus lihtsalt *URL* kasutamist.

Oma lõputöö projekti realiseerimise jaoks autor valis klient-server mudeli, kuna autoril on keskendatud platvorm, kust iga kasutaja võib kätte saada oma jaoks vajalikuid andmeid. Kas osaleda kursuses või vaadata otseülekannet.

3 Kasutatud tarkvara riistade analüüs ja valik valitud arhitektuuri jaoks

Selles peatükis toimub vajaliku tarkvara riistade loendamine. Kirjeldatakse, milliste kriteeriumite järgi oli valitud tarkvara riistaid. Toimus väike sissejuhatuse populaarsusse *tech-stack* (tehnoloogiliste lahenduste) kohta.

3.1 Vajaliku tarkvara riistade valik

Lõputöö kirjutamise esimene eesmärk oli valida vajalikuid tarkvara riistu. Peamine riist iga rakenduse kirjutamise jaoks on *IDE*. *IDE* valikust sõltub kui kiir ja mugav on rakenduse arendus protsess. *IDE* valik mängib suurt rolli, sest IDE peab pooldada programmerimist määratud programmerimis keeles. Teiseks oli vajalik valida tehnoloogiline lahendust, mis sobib projekti nõuetele ning annab kiire arenduse võimalust. *Tech-Stack* tavaliselt sisaldab *front-end* raamastikku, *back-end* raamastikku, andmebaasi ja serveri.

Front-end raamistik vastutab veebilehe välja nägemise eest ja äriloogika töötamise viisi eest. *Back-end* raamistik vastutab andmevoo ja infotöötluse toimumise eest. Andmebaas on nagu nimelt arusaadav – andmete salvestuse koht, kus *SQL* baasi juhul salvestatakse andmeid tabelites, *noSQL* baasi juhul andmeid salvestatakse *JSON* objektidel. *JSON* objekt võib omandada graafilise kuju.

3.2 Tarkvara riistade valiku kriteeriumid

Tarkvara riistade valiku kriteeriumid olid järgmised: peavad olema sobilik kiire arenduse protsessi jaoks, suure andmevoo toetusega ja paindlikusuga. *IDE* peab toetama arendust valitud programmeerimiskeeles.

Selle lõputöö ei pööra suurt tähelepanu *IDE* valikusele, kui see on rohkem maitse ja programmirimiskeele toetuse küsimus. Integreeritud programmeerimiskeskkonnaks oli valitud *Visual Studio*.

3.3 Populaarsed tech-stack'id (Tehnoloogiline lahendus)

Autor alustab populaarsete tehnoloogiliste lahendustega tutvustamisega. Sellises peatükis käsitletakse ideaalseid tehnoloogilise lahenduste mudeleid. Praktikas võib sarnases projektis kasutada erinevaid *front-end* ja *back-end* raamistikke koos. Ehk näiteks *Python*/*Django* ja *Node.js*. Kuna antud lõputöö raames valmistatakse ainult esimest etappi, siis operatsioonide süsteemide valik ei ole oluline.

Esialgne arenduse protsess toimub *Windows* operatsioonisüsteemi põhjal, kui tulevikus on kavas migreeruda *Linux* või *BSD* põhilise operatsioonisüsteemile oma hea turvalisuse tõttu.

Analüüsimiseks valitud tehnoloogilised lahendused hõlmevad suurt osa esitatud võimalustest.

LAMP on täiesti avatud lähtekoodiga ja hea dokumentatsiooniga, aga päris endisaegne lahendus õppimise raskustega.

MEAN ja *MERN* on head näited, kuidas *JavaScript* on võimalik kasutada igas rakenduse kirjutamise etapis. Erinevus seisneb *front-end* raamastikus, mis muutus, kuidas arenduse protsess käib.

Ruby on Rails ja *.NET* pakkuvad täisehitatud lahendused rakenduse arendamiseks. Mõlemad lahendused on avatud lähtekoodiga, aga uue funktsionaalsuse allikad on erinevad. *Ruby on Rails* on rohkem ühiskonna põhiline projekt, kus dokumentatsioon pole alati seostatud realeluga.. *.NET* on *Microsofti* moodustus, mida toetavad *Microsofti* oma arendajad. Allikad hajuvad, kas *.NET* kasutamise on tasuta ärinduse jaoks.

JAMStack on kliendi põhiline tehnoloogiline arendus, kui suur osa arvustustest toimub kliendi pool. Mis esitab probleemi töö jagamiseks ja *API* kirjutamisega.

Esitatakse veel mõned eksootilisi lahendusi rakenduse kirjutamiseks nagu *Flutter* ja *The Serverless Stack.* Probleem seisneb selles, et nende õppimiskõver on nii omapärane, et saadud kogemust on raske kuskil edasi kasutada.

3.3.1 *LAMP*

LAMP on tehnoloogiliste lahenduste tööstuslik standard. Veebiarenduses pakub see parimat kulutõhusust, paindlikkust ja jõudlust. *LAMP* on lühend, mis tähistab:

Linux (operatsiooni süsteem)

Apache (HTTP server)

MySQL (andmebaas)

PHP (programmerimisekeel, võib olla *Perl* või *Python* ka)

Kõik need tehnoloogilise lahenduse kihid on tasuta ja avatud lähtekoodiga. Selle *tech-stack'i* saab kasutada mis tahes operatsioonisüsteemis, samal ajal on võimalik selle *tech-stack'i* komponente muuta selleks, et rohkem sobitada esitatud nõutele.

***LAMP* eelised ja puudused:**

+ Järeleproovitud ja õige

+ Avatud lähtekoodiga ja dokumentatsiooniga

- Järsk õppimisekõver

- Omapärane arhitektuur, mis võib kahjustada produktiivsust.

LAMP omandab omapärast õppimiskõverat, kui hea dokumentatsioon võib siluda õppimise protsessi. *PHP* kasutamise *back-end* osa kirjutamiseks pole tänapäeval väga populaarne turvalisuse tõttu, *PHP* vajab eraldi koodi kirjatumist selle probleemi lahendamiseks. Samal ajal autor sai aru, et see on kõige paindlikum võimalus rakenduse arendamiseks, kui ka oma ajal problemaatiline.

3.3.2 *MEAN*

Üks tuntud *tech-stack MEAN* toob veebiarendajatele palju eeliseid. Nende eeliste hulgas on ühe keele (*JavaScripti*) kasutamine ning esialgsed tehnoloogiad on avatud lähtekoodiga ja tasuta. *MEAN* koosneb:

- *MongoDB* (*NoSQL* andmebaas)
- *Express.js backend* veebi raamastik
- *Angular.js frontend* raamastik
- *Node.js* avatud lähtekoodiga, kross-platvorm server

MEAN pakub praktilist lähenemist kiirete ja ülitõhusate rakenduste loomiseks. Kõik komponendid kasutavad andmete edastamiseks ja moodulite teeki tasuta juurdepääsuks *JSON*-i (*JavaScripti* objektimärked). Antud *tech-stack* aitab ehitada skaleeritavaid tarkvaratooteid. See sobib suurepäraselt igat tüüpi veebilehtedele ja interaktiivsetele rakendustele.

***MEAN* eelised ja puudused:**

+ Kasutusel on ainult *JavaScript* programmeerimisekeel.

+ Rakenduse kood on isomorfne.

+ Väike osa mälust hõivatud.

+ *MongoDB* on ehitatud pilve rakenduse arendamiseks.

- Mittesoovitatav suureandmevooga rakenduse jaoks.

- Puuduvad *JavaScript'i* spetsiifilised juhised.

- Ühelõimeline, kasutab virtuaalmasinat.

MEAN on esimene esindaja, mis pakub kasutada *JavaScripti* kõikide rakenduse osade kirjutamiseks. See on päris kergesti omandatav tehniline lahendus, mis annab võimalust isomorfse printsiipi kasutada.

Kui autor pöörab tähelepanu sellele, et *MEAN*'iga rakenduse arendus vajab suurt osa oma poolt kirjutatud koodi, kui turvalisuse tase sõltub rohkem arendaja realisatsioonist. Kasutab *TypeScripti*, mis on tegelikult sarnane *JavaScript'*iga, aga iga muutuja või objekti tüübi saab kindlasti määrata. See vähendab võimaliku vigade arvu.

3.3.3 *MERN*

Sisuliselt on *MERN* sarnane *MEAN'*iga. Erinevus seisneb selles, et kasutatav *front-end* raamistik on Angular.js asemel React.js, mis on arendatud *Facebook'i* poolt.

MERN-i kasutamise peamisteks eelisteks on *React*-integreerimine, võimalus kasutada koode üheaegselt brauserites ja serverites, võimas teek ja täispika arendusvõimalus (*frond-end* ja *back-end*).

Lisaks on *React* tuntud oma paindlikkuse ja jõudluse poolest interaktiivsete kasutajaliideste väljatöötamisel. Aga probleem *React.js* kasutamisega seisneb selles, et puhtalt *HTML* pole võimalik kasutada templati kirjutamise jaoks, kuid aga *Angular* pakub võimalust kasutada *HTML + TypeScript*.

***MERN* eelised ja puudused:**

+ Kasutusel on ainult JavaScript programmeerimisekeel.

+ Rakenduse kood on isomorfne.

+ Rakenduse kood on kasutatav serveril ja brauseril.

+ *MERN* kasutab *MVC* arhitektuuri.

- Mittesoovitatav suureandmevooga rakenduse jaoks.

- Puuduvad JavaScript'i spetsiifilised juhised.

- Ühelõimeline, kasutab virtuaalmasinat.

MERN tehnilise lahendusele kuuluvad sama kommentaarid nagu *MEAN* kohta. Kui *React* raamistik nõuab ainult oma poolt ehitatud funktsionaalsuse kasutamist *front-end* kirjutamiseks. Autori poolne ettepanek: Angular.js pakub rohkem funktsionaalsust kui *React*, mis on põhiliselt mõeldud ainult kasutajaliides disaini jaoks.

3.3.4 Ruby on Rails

„Ruby on Rails“ on programmeerimiskeskkond, mis kasutab dünaamilist programmeerimiskeelt *Ruby*. *RoR*-i abil saate kogu arendusprotsessi lihtsustada.

Serveripoolne veebirakenduse arendus, mis on kirjutatud *Ruby* programmeerimiskeeles, nõuab sellega töötamiseks vähem teadmisi. See võimaldab andmebaaside haldamiseks ja veebilehtedeks kasutada väikestruktuure.

„*Ruby on Rails“* töötab kasutajaliideste loomisel hästi koos *HTML*-i, *CSS*-i ja *JavaScripti* ning andmete edastamiseks *JSON*-i või *XML*-iga.

***Ruby on Rails* eelised ja puudused:**

+ Kasutab *MVC* arhitektuuri.

+ Kergelt jälgida muutusi rakenduses.

+ Paindlik ja eksisteerib palju pistikprogramme.

+ *„All-in-One“* lahendus. (Kõik komponendid on ühes pakendis)

- Aeglane suure andmevoogu töötlus.

- Küsitatav mitmelõimeline toetus

- *RunTime* ja *Boot* kiirus.

Ruby on Rails on väga omapärane produkt, mis annab esialgse platvormi iga etappi arendamiseks, kui vajab teiste tehnoloogiate integratsioon või kasutamine lõpule jõudmiseks. Lühidalt aitab rakenduse ehitamises, kui funktsionaalsuse dokumentatsioon vaevab. Õppimiskõver on sarnane *LAMP* tehnoloogilise lahendusega.

3.3.5 *.NET*

.NET on *Microsofti* loodud ja toetatud tarkvaraarenduse raamistik ja ökosüsteem, mis võimaldab kerget töölaua- ja veebirakenduste arendamist. See on populaarne ja tasuta platvorm, mida praegu kasutatakse paljude erinevat tüüpi rakenduste arendamise jaoks, kuna see pakub programmeerimiskeskkonda enamiku tarkvaraarenduse etappide jaoks. *.NET* sobib kõige paremini ettevõtetele, kes otsivad mitmesuguseid funktsioone, nagu veebipõhised teenused, töölaua tarkvara ja pilve infrastruktuuri toetus.

.*NET* eelised ja puudused:

+ Universaalne .*NET* standart

+ Kross-platvormilinne

+ Puhverdamise süsteem

+ Automaatne monitooring kasutades *ASP.NET*

- Lähtekood pole täiesti avatud.

- Tarnija lukk (Uued tunnused sõltuvad tarnija soovist)

- Suletud ökosüsteem

.NET omandab ainulaadse mainet. Ühelt poolt, mõned funktsioonid on täiesti omapärased ja väga kasulikud arenduses. Teiselt poolt arendajate ühiskonnas eksisteerivad sama hea populaarsemad lahendused parima reputatsiooniga. Autor kahtleb *.NET* tuleviku arendamise eest.

3.3.6 *JAMStack*

JAMStack sai alguse ideest, et arendajad vajasid viisi, kuidas sõna “staatiline” potentsiaalsete negatiivsete stigmasid vältida. *JAMStack* annab võimaluse kirjeldada, kuidas kaasaegne dünaamiline veebirakendus on üles ehitatud.

Kontseptsioon põhineb arhitektuuril, kus veebilehte saab edastada staatiliselt, näiteks *HTML*-i esitamine staatilisest sisalduses, kuid dünaamilise sisu ja interaktiivse kogemuse pakkumine selliste tööriistade kaudu nagu *JavaScripti*. Termin ise tähistab *JAM*-i veebilehel: *JavaScript*, *API*-d ja *Markup*.

***JAMStack* eelised ja puudused:**

+ Edastamise kiirus. Suurem osa rakendusest on kliendipõhiline.

+ Skaleeritav. Staatilised päringud on kiirem kui dünaamilised.

+ Suhteliselt turvaline.

- Alalise skripti teostamine

- Probleemid *API* sisalduse edastamisega

JAMStack on kliendi põhiline tehnoloogiline lahendus, mis pakub paremat tootlikkust, kui esialgse ettevalmistuse etapi suurus on vähenenud. Autor kahtleb, kas sellega seotutud arendamise oskused saab kuskil edasi kasutada. Ning kliendipoolne realisatsioon takistab tööjaotutust.

3.4 Tech-stack'i analüüsimetoodika

*Tech-stack'*i analüüsi jaoks on tähtis määrata kriteeriume, mille põhjal on võimalik otsustada, milline *Tech-stack* vastab ettemääratud nõuetele. *Back-end* programmeerimiskeele produktiivsus on üks selline võimalik kriteerium, kuid esiteks on vaja määrata mis produktiivsus üldse tähendab.

Produktiivsus võib määrata koodi sooritamise kiirus, kuid selles olukorras on vajalik iga programmeerimiskeele tundmine, et teha õige võrdlemist. Seetõttu on otstarbekam võrrelda nende arhitektuurne ehitus. Kas see on kompileeritav, tõlgendatav programmeerimiskeel või koodi sooritamine toimub virtuaalmaasinas? Ehk rakenduse skaleeritavuse tase siis, kui andmevoog suureneb. Kas edasi arendamise võimalused on üldse olemas.

Teine kriteerium on kas tehnoloogilises lahenduses kasutavaid raamastikud on avatud lähtekoodiga ja/või litsentsivabad. Sellest sõltub kui paindlik on kasutusse võetud *tech-stack* ja kas on vajalik mõelda kulude kohta *startupi* algfaasis.

Kasutusel olev andmebaas mängib suur rolli ka. *SQL* andmebaasid on ehitatud ainult tabelite põhjal, aga *noSQL* andmebaasid töötavad erinevate andmestruktuuriga. Andmete salvestuse viis on mittetabulaarne ja mitterelatsiooniline ehk on võimalik salvestada andmeid *JSON* objektidel ja graafiku kujul. Aga kui kasutava andmebaasi on päris kergelt vahetatav, ei pööra selle peale tähelepanu.

Tähtis on veel sellised faktorid nagu arendaja ja/või ühiskonna toetus. Kui mingisugune toetus puudub, siis tehnoloogilise lahenduse kasutamine on mõttetu. Projekt ei saa olla elujõuline.

Kui tehnoloogiline lahendus pole kergesti omandatav, siis sellega tekkivad projekti kirjutamise kiirusega probleemid. Järske õppimiskõver pole sobilik ettemääratud kriteeriumide järgi.

3.5 Tehnoloogiliste lahenduste võrdlemine ja analüüs

Selleks, et võrrelda tehnilisi lahendusi olid eelmises paetükis määratud tingimused, samal ajal oli kaalukad programmeerimiskeelte omapärased.

Tabel 3. Tehnoloogiliste lahenduste võrdlus

Nimetus → ↓Kriteeriumid	*LAMP*	*MEAN*	*MERN*	*Ruby on Rails*	*.NET*	*JAMStack*
Avatud lähtekoodiga	+	+	+	+	+*	+
Tasuta ärikasutusjuhtudele	+	+	+	+	+**	+
Skaaleritav lahendus	-***	+	+	+	+	+
Kergelt kasutatav	-	+	+	-	-	+
*Tech-stack'*i ühiskond	+	+	+	+	+	+
Arendaja toetus	+	-****	+	+	+	+

* *.NET* mõned osad on kinnitud lähtekoodiga, kui üldiselt tehnoloogiline lahendus on avatud lähtekoodiga.

** Tehniliselt tehnoloogiline lahendus on tasuta, kui *IDE*, mis on soovitav tehnilise lahenduse kasutamiseks on tasuline.

*** *LAMP* järsk õppimiskõver ja omapärane arhitektuur ei soodusta skaleeritavat arendust.

**** *Google* kavatseb lõpetada Angular.js raamistiku toetust 2022.aastal.

Ülalpool toodud tabelist on võimalik näha, et *MERN* ja *JAMStack* vastavad eelmises peatükis toodud nõuetele, kuna neil on olemas omapärased probleemid ka. *React* raamistik ei anda võimalust puhta *HTML* kasutada visuaalse kujundamise jaoks. Ning selles tehnoloogilises lahenduses kasutakse *noSQL* andmebaasi, mis on problemaatiline kasutada määramata struktuuri tõttu. Vaata lähemalt peatükis 5.

JAMStack on pööratud rohkem kliendi poole ehk arvutused toimuvad kliendi pool. Kui see teeb raskemaks *API* integratsiooni. *JAMStack* on päris uus standart ja selle eluvõime põlu lõpuni tõendatud.

4 Nõutele vastavale tehnoloogiate valik

Eelesitatud tehniliste lahendus kohta autor võib öelda, et eksisteerivad veel mõned iseseisvad raamastikud, mis tegelevad serveri osa või kliendi osa realisatsiooniga. Kui eelmises peatüki järeldus oli et olemasolevad tehnilised lahendused ei vasta esitatud nõuetele.

4.1 Sissejuhatus *back-end* (tagaprogrammi) raamastikkudele

Järgmistes peatükkides toimub tutvustus taga programmi realiseerimise seotud tööriistadega. Tööriistade analüüsi valiku kriteeriumid oli järgmised: erinevate tehnoloogiate lahenduste esitamine või sama programmerimiskeele kasutus määratult erinevas kontekstis.

4.1.1 *Java Spring* raamistik

Spring raamistik on tegelikult lihtsa sõltuvuse süstimiskonteiner, millele on lisatud paar mugavuskihti (juurdepääs andmebaasile, puhverserverid, aspektidele suunatud programmeerimine, *RPC*, veebi *MVC* raamistik). See aitab Java-rakendusi kiiremini ja mugavamalt arendada.

Spring on ehitatud modulaarsuse printsibi järgi. Kuigi pakettide ja klasside arv on märkimisväärne, arendaja peab muretsema ainult vajalike pakettide pärast ja ülejäänud osa ignoreerima.

Spring ei mõtle ratast uuesti välja, vaid kasutab olemasolevat tehnoloogiat, näiteks mitut *ORM*-raamistikku, *Log-in*, *JEE-*, *Quartz* ja *JDK*-taimerit ning muid *View* tehnoloogiaid

Spring pakub mugavat *API*-d tehnoloogiaspetsiifiliste erandite tõlgendamiseks järjepidevateks, kontrollimata eranditeks.

Spring'is on hästi loodud veebi *MVC* raamistik, mis pakub suurepärast alternatiivi veebiraamistikele nagu *Struts* või muudele üle konstrueeritud või vähem populaarsetele veebiraamistikele.

4.1.2 *Django* raamistik

Django on *Pythoni*-põhine avatud lähtekoodiga veebiraamistik, mis kavatseb järgida mudeli-malli-vaate (*MVC*) arhitektuurset mudelit. *Django* eesmärk oli aidata arendajatel võimalikult kiiresti rakendusi ideest lõpuni viia. *Django* võtab turvalisust tõsiselt ja aitab arendajatel vältida paljusid levinud turvavigu.

Kiired arendusprojektid on võimalikud *Django* kasutamisega ja see on algajatele sõbralik, kellel on juba *Pythoni* mõistmine olemas. *Django* ehitati ja modelleeriti pragmaatilisel ja puhtal disainil ning sellega kaasnevad kõik keerukate veebirakenduste ehitamiseks vajalikud peamised komponendid.

4.1.3 *C#* ja *ASP.NET Core*

.*NET* funktsionaalsus on päris mahukas, aga selles peatükis vaadeldakse peamiselt serveri poolne funktsionaalsust.

See on platvormidevaheline avatud lähtekoodiga raamistik Interneti-ühendusega pilvepõhiste kaasaegsete veebirakenduste arendamiseks. Selle raamistiku kaudu on täiesti võimalik luua efektiivsemaid veebirakendusi ja teenuseid koos mobiilirakenduste taustaprogrammide ja isegi *IoT*-rakendustega.

4.1.4 *Node.js*

Node.js on avatud lähtekoodiga platvormitundetu käituskeskkond serveripoolsete ja võrgurakenduste arendamiseks. *Node.js* kasutab *MIT*-litsentsi. *Node.js* peal ehitatud rakendused on kirjutatud *JavaScripti* abil ja neid saab käivitada *Node.js*-i käitamise ajal *OS X*-is, *Microsoft Windowsis* ja *Linuxis*.

Node.js on asünkroonne ja sündmustepõhine - kõik selle teegi *API*-d on asünkroonsed, see tähendab, et need pole blokeeritud. Sisuliselt tähendab see, et *Node.js*-põhine server ei oota kunagi *API* tagastamist.

Server liigub pärast selle kutsumist järgmisele *API*-le ja *Node.js* sündmuste teavitusmehhanism aitab serveril saada vastust eelmisest *API*-kõnest.

Ühelõimeline, kuid hästi skaleeritav - *Node.js* kasutab sündmuste loopimisega ühte keermestatud mudelit. Sündmuste mehhanism aitab serveril reageerida mitteblokeerivalt ja muudab serveri väga skaleeritavaks, võrreldes traditsiooniliste serveritega, mis loovad

päringute käsitlemiseks piiratud niidid. *Node.js* kasutab ühte keermestatud programmi ja sama programm suudab teenust pakkuda palju suuremale hulgale taotlustele kui traditsioonilised serverid, näiteks *Apache HTTP Server*.

Puhverdamine puudub - *Node.js* rakendused ei puhverda kunagi andmeid. Need rakendused väljastavad andmed lihtsalt tükkidena.

4.2 *Node.js* ja *Django* võrreldus

Esialgselt määratud kriteeriumid tehnoloogiliste lahenduste jaoks ei muutnud, aga kui käsitletakse eraldi raamastikku, siis on tähtis arvestada programmeerimiskeelt ja paindlikust teiste tehnoloogiatega. Analüüsimiseks oli valitud *Django* ja *Node.js*, mis esindavad erinevaid programmismiskeeli, käitumisi ja arenduse viisi. Autori mõttes mõlemad valikud sobivad ilusasti tagaprogrammi kirjutamiseks, kui praegu on vajalik otsustad, mida paremini kasutada.

Tabel 4. *Django* ja *Node.js* võrreldus

Nimetus → Kriteerium↓	*Django*	*Node.js*
Arhitektuur	See järgib *MTV* mudeli malli vaadet. See aitab andmete kontrollimis eajal andmeid valideerida ja serveriga suhelda.	See on käituskeskkond ja töötab sündmustel põhineval mudelil. See töötab opsüsteemis, säilitades väikese hulga taotlusi.
Turvalisus	*Django* on turvalisem ja varustatud sisseehitatud süsteemiga, mis hoiab ära turva puudused.	*Node.js* ei ole nii turvaline kui *Django* ja nõuab turvavigade haldamiseks süsteemis käsitsi toiminguid.

Produktiivsus	See pakub paremat produktiivsust, kuna on sisseehitatud maja malli süsteem, mis hõlbustab vajaliku ülesande kiiret täitmist.	*Node.js* produktiivsus on samuti hea, kuna see võimaldab veebirakenduste spetsialiseeruvatel arendajatel rohkem vabadust. Kuid jällegi pikendab see rakenduse loomiseks kuluvat kogu aega
Keerukus	*Django* on keerulisem, kuna arendaja peab probleemide lahendamiseks liikuma kindlaksmääratud rada.	See süsteem on vähem keeruline. Siin on arendajal vabad käed probleemide lahendamiseks neile meelepärasena
Efektiivsus	See raamistik on tõhusam ja pakub kiiret kiirust. Ja nii on see kulutõhusam.	Seda raamistikku on lihtne õppida, kuid see võtab rohkem tööaega. Seega on see vähem kulutõhus variant.
Paindlikus	See raamistik pakub piiratud paindlikkust ja sellel on üsna ranged arendusfunktsioonid.	Tänu ulatuslikule *JavaScripti* teegile on *Node.js* mitmesuguseid tööriistu ja funktsioone. Võite isegi *JS*-põhiseid rakendusi ehitada nullist.

Full-stack arenduse võimalused	*Django* ei paku *Full-stack* arendust.	Tänu oma võimele rakenduse loomine nii *front-end* kui ka *back-end* osi on võimalik kasutades ainult ühte programmeerimiskeelt – *JavaScripti*. Sellepärast on *Node.js* üks eelistatumaid veebirakenduste arendamise tehnoloogiaid.

Node.js annab rohkem paindlikkust rakenduse arendamiseks, kui see on põhimõttelist tühi lõuend ning õppimisekõver on siledam, kui *Django* oma. Samal ajal turvalisuse realisatsioon on parem tehtud *Django* raamastikus ehk pole vaja eraldi turvalisuse küsimusega tegeleda. *Django* on kiirem, kui Node.js, see omadus tuleb *Python* programmiskeelest. Sealt tuleb *Django* modulaarsus ka. Moodulaarsus tähendab, et *Python* võib kasutada teise keele peal kirjutatud moodulid, mis soetab Mikroteenuste arhitektuuri kasutamist ja suure efektiivsust. Esialgne arendamise protsess võib olla natuke kiirem kui kasutada *Node.js*.

Ülalpool toodud analüüsi võttes, autori arvates *Django* sobib rohkem alguses määratud nõuetele tagaprogrammi osa arendamiseks.

5 Andmebaasi realisatsioon

Selles peatükis analüüsitakse *SQL* ja *noSQL* andmebaasi tüüpe. Otsustakse, milline andmebaasi tüüp vastab paremalt projekti nõuetele. Põhjendatakse, mille järgi oli tehtud valik. Esitatakse andmebaasi skeemi.

5.1 Sissejuhatus andmebaaside tüüpidesse

Selles peatükis kirjeldatakse andmebaaside tüüpe ja nende eripärasid.

5.1.1 *SQL* andmebaasid

SQL andmebaasid kasutavad andmete määratlemiseks ja manipuleerimiseks struktureeritud päringukeelt (*SQL*). Ühelt poolt on see äärmiselt võimas: *SQL* on üks kõige mitmekülgsemaid ja laialdasemalt kasutatavaid võimalusi, mis muudab selle turvaliseks valikuks ja eriti hea keeruliste päringute korral. Teiselt aga võib see olla piirav. *SQL* nõuab, et enne andmete töötamist arendajad kasutaksid oma andmete struktuuri kindlaksmääramiseks eelnevalt määratletud skeeme. Lisaks peavad kõik teie andmed järgima ühesugust struktuuri.

SQL andmebaasi eeldused ja puudused:

+ Standardne juurdepääs andmetele *SQL* kasutamise kaudu.

+ Normaliseerimise ja muude optimeerimisvõimaluste tõttu väheneb andmesalvestuse jalajälg. Sageli on tulemuseks parem jõudlus ja ressursside efektiivsem kasutamine.

+ Tugev ja arusaadav andmete terviklikkuse semantika *ACID* kasutamise kaudu (aatomilisus, järjepidevus, isoleerimine, vastupidavus).

+ Üldiselt paindlikum päringute toetus, mis suudab töötada suurema koormusega. SQL abstraheerib aluseks oleva rakenduse ja võimaldab mootoril optimeerida päringuid, et need vastaksid nende kettal kuvatavale representatsioonile.

- Jäigad andmemudelid, mis nõuavad hoolikat ülesehitust, et tagada piisav jõudlust ja vastupanu evolutsioonile - skeemi muutmine võtab sageli seisakuaega.

- Horisontaalne skaleerimine on keeruline - kas täielikult toetuseta või ainult suhteliselt toetatud.

Autori mõttes, *SQL* andmebaasitel on kindel struktuur, mis iga arendaja peab meeles pidama selleks, et arenduse protsessi lõpuni jõuda. Selles on olemas suur pluss, et pole vajalik midagi välja mõelda või leida struktuuri aluseks.

5.1.2 *noSQL* andmebaasid

NoSQL andmebaasides on struktureerimata andmete dünaamilised skeemid, ja selle andmebaasi tüübi korral andmeid salvestatakse mitmel viisil: need võivad olla veergudele orienteeritud, dokumendile orienteeritud, graafikapõhised või korraldatud *KeyValue* poena. See paindlikkus tähendab, et:

- On võimalik luua dokumente, ilma et peaks eelnevalt nende struktuuri määratlema
- Igal dokumendil võib olla oma ainulaadne struktuur
- Süntaks võib andmebaasist andmebaasini erineda
- On võimalik alati lisada „*field*“

Skaleeritav ja ülimalt kättesaadav - paljud *NoSQL*-i andmebaasid on üldjuhul loodud toetama sujuvat veebipõhist horisontaalset mastaapsust ilma oluliste üksikute tõrkepunktideta.

Paindlikud andmemudelid - enamik mitteseotud süsteeme ei nõua, et arendajad võtaksid andmemudelite suhtes ette kohustusi; skeemide olemasolu saab sageli muuta käigu pealt

noSQL andmebaasi eeldused ja puudused:

+ Hästi skaleeritav

+ Toetab suurt andmevoogu ja suure andmekogu rakendusi

- Väiksem toetust. Iga *NoSQL*-i andmebaas proovib olema avatud lähtekoodiga, vaid paar ettevõtet käsitsevad nende toetust.

- Pole lõpuni arendatud ehk pole valmid funktsionaalsuse mõttes.

Kui *noSQL* andmebaasitel puudub kindel arhitektuur, esialgne arenduse etappide keerukus mitmekordselt kasvab. Oma kogemuse ja teadmiste põhjal autor otsustas võtta *SQL* põhilise andmebaasi. *noSQL* andmebaaside paindlikus ja „skaaleritavus“ on päris kasulikud omadused, kuid ettemääratud struktuuri puudus on probleeme tekkitav küsimus, millest kujuneb raskusi andmebaasi projekteerimise jaoks. Aga *noSQL* andmebaas võiks olla projekti tulevikus kasutatud andmete puhverdamise jaoks.

5.2 Andmebaasi arhitektuuri valiku põhjendus

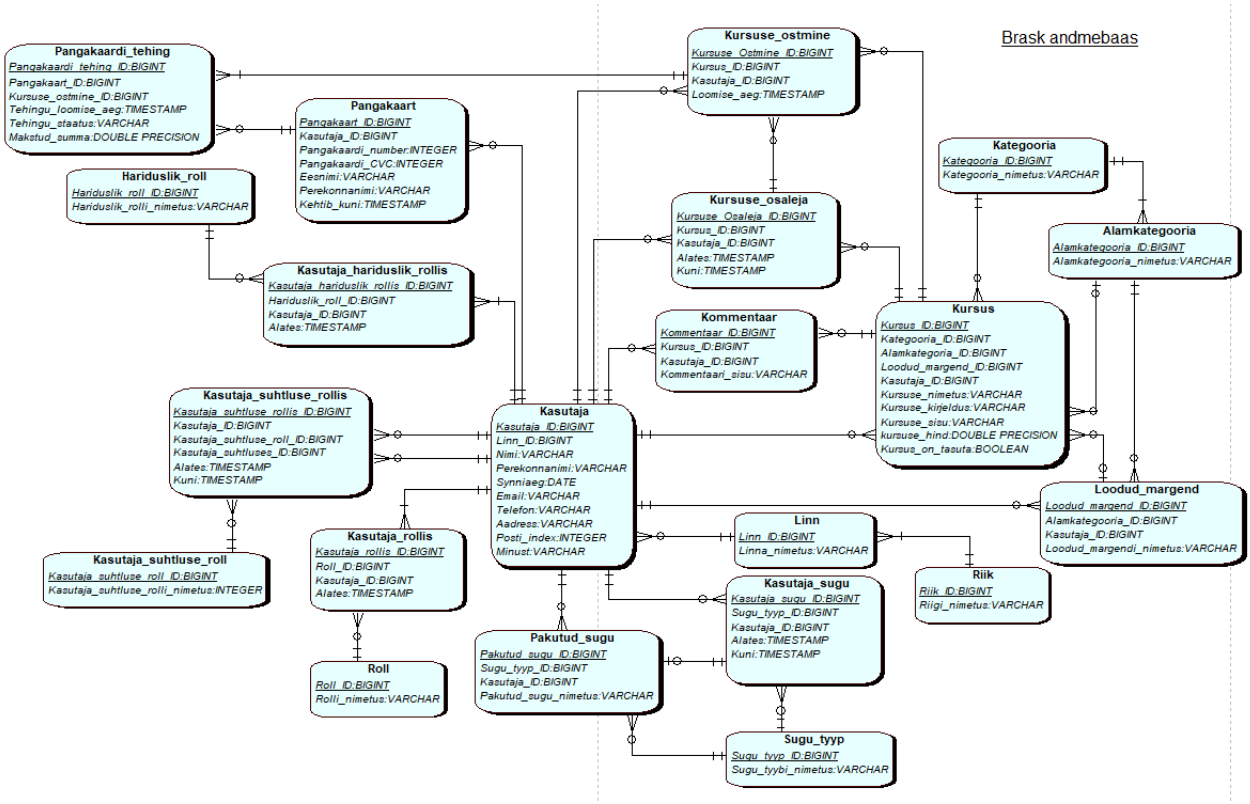
Kui autor kavatseb tulevikus osaleda *startup'i* arendamises, siis *SQL* andmebaasi valiku kriteerium oli tasuta funktsionaalsus. Projekti arendamiseks olid käsitletud järgmised andmebaasid:

- *MySQL*
- *PostgreSQL*
- *MariaDB*
- *CockroachDB*

Andmebaasi valikuks sai *PostgreSQL*. Põhjuseks on rangem suhe SQL standardiga, võrreldes näiteks *MySQL* andmebaasiga. Vaatamata sellele, et *Oracle Database* on tasuline produkt, see annab natuke rohkem funktsionaalsust, kui on olemas teistel *SQL* põhilistel andmebaasitel. Aga sellises võrdluses osalevad ainult tasulised *SQL* põhilised andmebaasid.

5.3 Andmebaasi skeem

Sellises peatükis käsitletakse rakenduse andmebaasi arendamiseks kasutatav skeem. Selle skeemi järgi oli ehitatud andmebaas, mis on kasutatud projektis. Skeem on kavas tulevikus täiendada. Täisskeem on nähtav Lisas 5.



Joonis 6. Rakenduse andmebaasi skeem

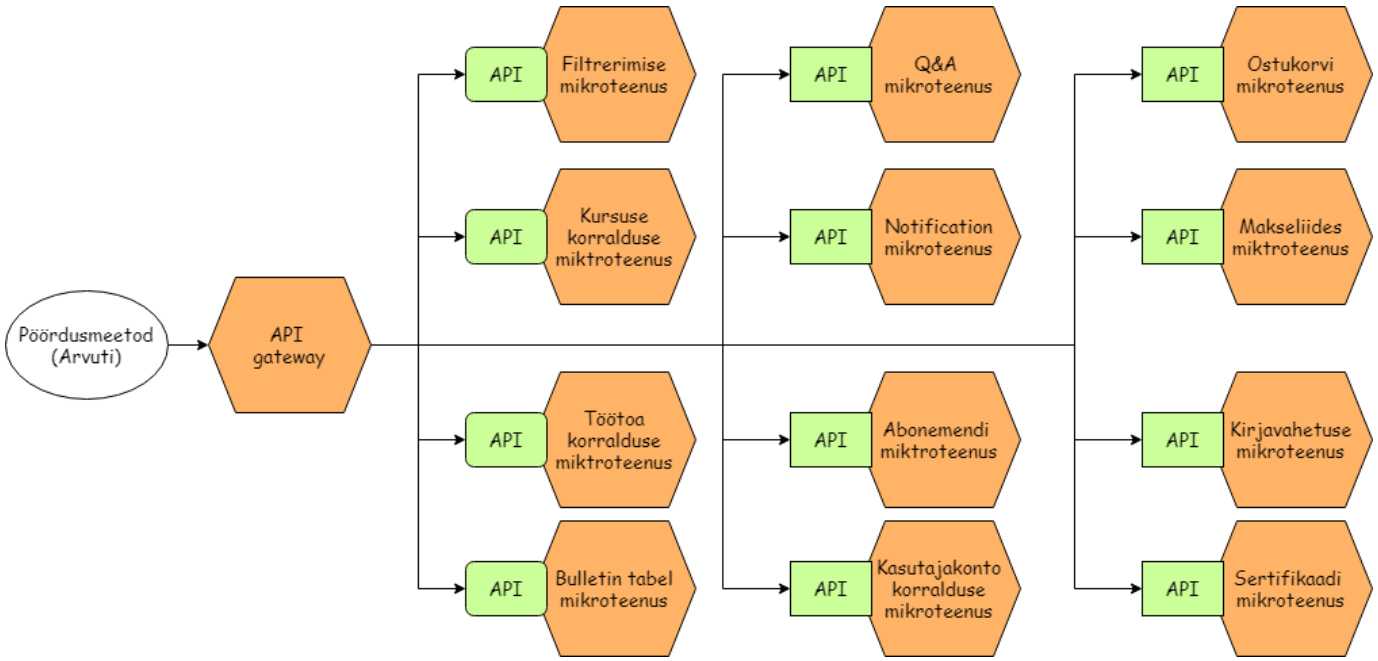
Andmebaasis kasutatakse järgmiseid andmetüüpe:

- *VARCHAR* – nimetuse ja erinevate stringide salvestamiseks.
- *BIGINT/INTEGER* – kasutatakse loenduri ja primaar võtme jaoks.
- *DATE* – kuupäeva salvestamine, kas sünniaeg või mingi süsteemi jaoks tähtis päev.
- *TIMESTAMP* – objekti loomise aja salvestamine
- *BOOLEAN* – salvestab lipu väärtust.
- *DOUBLE PRECISION* – kursuse hinna salvestamine (Ujukomaarv kahekordse täpsusega)

6 Serveri osa realisatsioon

Serveri osa realisatsiooni jaoks oli valitud kasutada *Django* raamastiku. *Django* paremini vastab lõputöös määratud nõuetele. Samuti pakub võimalust kasutada teiste keelte mooduleid selleks, et kiirendada andmetöötlust.

Oli valitud ehitada rakendust mikroteenuste arhitektuuri põhjal. Hostid suhtlevad üksteisega kasutades klient/serveri mudeli. Selleks, et visuaalselt aru saada, millega tuleb tegeleda, oli koostatud abi skeem. Ülalpool on toodud rakenduse arhitektuuri skeem.

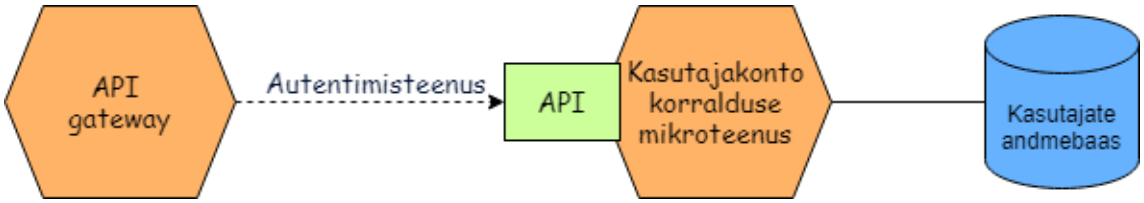


Joonis 7. Rakenduse mikroteenuste skeem

Antud skeem kirjeldab mikroteenuseid, mis on vajalik luua projekti realiseerimise jaoks. Kui on tähtis märkida, et lõputöö raames valmistatakse ainult analüüsi ja väike osa funktsionaalist. Koostatud skeem vastab töös esitatud *EPICs* kirjandusele, kui mikroteenuseid saab taaskasutada ehk sama funktsionaalsus on kasutatav erinevatel veebilehel, nende arv on koondatud võrreldes *EPIC*'dega.

6.1 Sisse logimise *API* realisatsioon

Selleks, et sisse logimise protsess oli võimalik, on vaja hoida kasutaja andmeid andmebaasis. Põhjuseks on vajadus tuvastada, et kasutaja on olemasolev isik andmebaasis ning samuti millised õigused kuuluvad isikule.

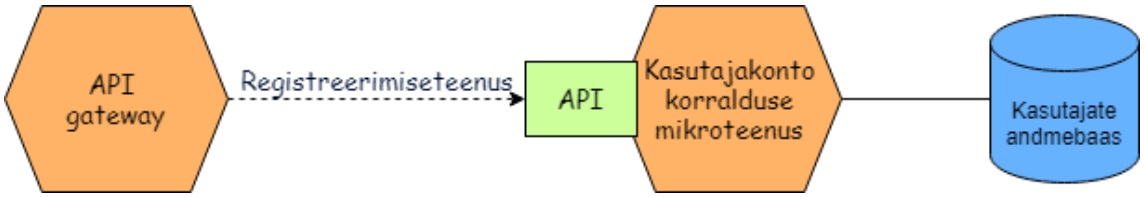


Joonis 8. Sisse logimise skeem

Sisse logimise on skeemi vaatest lihtne protsess, kui programmiskeelne realisatsioon vajab kontrollimisest igal etapil selleks, et vältida võimalikkuid turvalisuse probleeme. Näiteks kasutaja parool edastus peaks olema krüpteeritud selleks, et tagada kasutajakonto turvalisust.

6.2 Registreerimise *API* realisatsioon

Registreerimise on lihtsam protress, kui sisse logimine. Isik annab oma andmeid andmebaasile, andmeid salvestatakse kasutaja tuvastuse jaoks. Nüüd kasutajal on uued õigused ja funktsionaalsus, mis on võimalik kasutada ainult registreeritud kasutaja.



Joonis 9. Registreerimise skeem

See skeem kirjeldab registreerimise protsessi. Samuti nagu sisse logimise protsessiga, käib suhtlemiseprotsess kasutajakonto korralduse mikroteenustega. Siin aga luuatakse uut sisestust andmebaasile, mille õigsuse on vajalik kontrollida samuti kaitsta *SQL*-injektsioon eest. *SQL*-injektsioon on *SQL* süntaksi kasutamine kahjuliku eesmärgi mõttes.

7 Arendusevektorid

Selles peatükis kirjeldatakse võimalikuid edasi arendamise suundi. Mobiilrakenduse loomine on nii kui nii kavas, aga parema realisatsiooni viis pole veel valinud. Oma otseülekande video-mängija loomine on teine võimalik arenduse suund.

7.1 Mobiilirakenduse loomine

Esimene arendusvektor on päris loomulik. Eeldatakse, et kasutame *WebView Android* platvormi arenduse jaoks ja leiame alternatiivi *iOS* platvormi arenduse jaoks. *WebView* põhiidee seisneb selles, et mobiilrakendus ainult näitab veebilehte, mille kujundus sobib mobiilseadmete kasutamisele. Mobiilrakendus toimub üleminek ühest lehest teisele. See lahendusviis vähendab aegnõudeid mobiilrakenduse arendamiseks sest, et lõpetud rakendus on just ümbris platvormi spetsiifiliselt koodist ja erakujundusega veebileht.

7.2 Oma otseülekande video-mängija loomine

MVP versioonis on kavas kasutada *Workshop* korraldamise jaoks otseülekanne teisest platvormist nagu *Twitch* ja *YouTube*. Oma otseülekande video-mängija loomine on väga suuremahuline protsess, sest iga otseülekanne vajab omapärast *Stream Key* ja serveri hostimist, kuhu transleeritakse otseülekannet ja teised kasutajad saab vaadata. Otseülekande kvaliteet tugevalt sõltub serveri infrastruktuurist ja video- ning audio kodeki valikust. Veel üks komistuskivi on *HDR* toetus sest, see vajab suurt ülekandekiirust.

7.3 Edaspidine serveri osa arendus

Tulevikus kavandatakse täiendada serveri osa funktsionaalsust vastavalt määratud joonise 2 arenduse etappidele. Kui selle lõputöö raames toimus ainult esimese arenduse etapp.

8 Kokkuvõte

Vastavalt püstitatud eesmärkidele oli tehtud rakenduse arhitektuuri, tehnoloogiliste lahenduste, eraldi *back-end* raamastiku ja andmebaaside analüüs selleks, et vasta küsimustele, millised tehnoloogiad vastavad paremalt rakenduse arendamiseks esitatud nõuetele. Samal ajal lahendati andmebaasi projekteerimise probleemi, oli koostatud andmebaasi skeem, mis aitas andmebaasi realiseerimises. Pärast analüüsimist ja projekteerimist toimus esimese etapi kirjutamine. Lõputöö näitab kuidas toimus projekti planeerimine ja valmistatakse esimese etappi.

Rakenduse kirjutamiseks oli valitud mikroteenuste põhiline arhitektuur. Hostide vaheline suhtlus toimub kliendi/serveri mudeli põhjal. *Rest API* on juurdepääsu protokoll, mis juhib andmete liikumist. Lõputöö raames kirjutatud rakendus kasutab *Django* raamastikku ja *Pythoni* programmiskeelt. Valitud andmebaas oli *PostgreSQL* oma range suhe *SQL* standardi eest. Serveri rakenduse programmikood on saadav *Github* veebilehel, mis on viimane allikas kasutatud kirjanduses.

Selle töö põhjal on võimalik tulevikus edasi arendada teadmiste jagamiste platvormi funktsionaalsust. Kui see punkt on juba autori kavas sisse viidud.

Kasutatud kirjandus

[1] M. Weisfeld, *The Object Oriented-Thought Process*, 4th ed. Addison Wesley, 2014

[2] TheBestSchools, *History of Online Education*, 2021. [Online]. Loetud aadressil: https://thebestschools.org/magazine/online-education-history/ Kasutatud: 13.02.2020

[3] S. Kuzmin. *Teadmiste jagamise platvormi analüüs ja klientrakenduse loomine startupi jaoks*. Tallinn: TTÜ, 2021

[4] IEEE, *12207-2017 - ISO/IEC/IEEE International Standard*, 2021. [Online]. Loetud aadressil: https://ieeexplore.ieee.org/document/8100771 Kasutatud: 14.02.2020

[5] Tutorialspoint.com, *SDLC – Overview*, 2021. [Online]. Loetud aadressil: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm Kasutatud: 17.02.2020

[6] Modern Systems Analysis and Design, 8th edition

[7] Atlassian, *Agile epics: definition, examples, and templates*, 2021. [Online]. Loetud aadressil: https://www.atlassian.com/agile/project-management/epics Kasutatud: 18.02.2020

[8] Atlassian, *User Stories with Example and Template*, 2021. [Online]. Loetud aadressil: https://www.atlassian.com/agile/project-management/user-stories Kasutatud: 18.02.2020

[9] E. Eessaar, *Andmebaaside projektreerimine*. Tallinn: TTÜ kirjastus, 2008

[10] TechTarget, *monolithic architecture* ,2021. [Online]. Loetud aadressil: https://whatis.techtarget.com/definition/monolithic-architecture#:~:text=A%20monolithic%20architecture%20is%20the,and%20unable%20to%20be%20changed. Kasutatud: 21.02.2020

[11] Microsoft, *Microservices architecture style*, 2021. [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices Kasutatud: 21.02.2020

[12] Medium, *Software Architecture – The Onion Architecture,* 2021. [Online]. Loetud aadressil: https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec Kasutatud: 21.02.2020

[13] Codeguru, *Understanding Onion Architecture,*2021. [Online]. Loetud aadressil: https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understanding-onion-architecture.html#:~:text=Onion%20Architecture%20is%20comprised%20of,on%20the%20actual%20domain%20models. Kasutatud: 01.03.2020

[14] Veterinaryitsupport.com, *Peer-to-Peer vs Client-Server Networks,* 2021. [Online]. Loetud aadressil: https://www.veterinaryitsupport.com/peer-to-peer-vs-client-server-networks/#:~:text=Peer%2Dto%2Dpeer%20networks%20connect,the%20data%20and%20manage%20resources. Kasutatud: 24.03.2020

[15] TechTarget, *Peer-to-Peer Architecture (P2P Architecture),* 2021. [Online]. Loetud aadressil: https://www.techopedia.com/definition/454/peer-to-peer-architecture-p2p-architecture#:~:text=Peer%2Dto%2Dpeer%20architecture%20(P2P%20architecture)%20is%20a,are%20dedicated%20to%20serving%20others Kasutatud: 13.02.2020

[16] Restfulapi.net, *What is REST,* 2021. [Online]. Loetud aadressil: https://restfulapi.net/ Kasutatud: 10.03.2020

[17] https://datatracker.ietf.org/doc/html/draft-box-http-soap-00

[18] UberEngineering, *The Uber Engineering Tech Stack, Part 1: The Foundation,* 2021. [Online]. Loetud aadressil: https://eng.uber.com/tech-stack-part-one-foundation/ Kasutatud: 12.03.2020

[19] Medium, *Engineering the Instagram Stories Team,* 2021. [Online]. Loetud aadressil: https://instagram-engineering.com/engineering-the-instagram-stories-team-e16ec62e364d Kasutatud: 14.03.2020

[20] Medium, *How to build a secure chat app like whatsapp, wechat & viber? ,*2021. [Online]. Loetud aadressil: https://medium.com/@johnvincentt27/how-to-build-a-secure-chat-app-like-whatsapp-wechat-viber-1245fa29ab3 Kasutatud: 13.02.2020

[21] Full Scale, *Top5*2021. [Online]. Loetud aadressil: https://fullscale.io/blog/top-5-tech-stacks/ Kasutatud: 20.03.2020

[22] Kaya Ismail, CMSWIRE, *JAMstack vs. LAMP Stack vs. MEAN vs .NET: Tech Stacks Compared*,2021. [Online]. Loetud aadressil: https://www.cmswire.com/digital-experience/jamstack-vs-lamp-stack-vs-mean-vs-net-tech-stacks-compared/ Kasutatud: 24.03.2020

[23] Sovereign, *Full Stack Development: What is MERN Stack and it's Advantages?*,2021. [Online]. Loetud aadressil: https://www.sovereignconsult.com/blog/full-stack-development-what-is-mern-stack-and-its-advantages/#:~:text=Advantages%20of%20a%20MERN%20Stack,code%20and%20server%2Dside%20code. Kasutatud: 25.03.2020

[24] Ncube, Alex Melnichuk, *Pros and Cons of .NET Framework* 2021. [Online]. Loetud aadressil: https://ncube.com/blog/pros-and-cons-of-net-framework Kasutatud: 26.03.2020

[25] Altexsoft, *The Good and the Bad of .NET Framework Programming,* 2021. [Online]. Loetud aadressil: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-net-framework-programming/ Kasutatud: 27.03.2020

[26] Jamstack, *The modern way to build Websites and Apps that delivers better performance*, 2021. [Online].Loetud aadressil: https://jamstack.org/ Kasutatud: 28.03.2020

[27] Brightspot, Mark Hoover, *What is JAMstack, and what does it mean for web development?* ,2021. [Online]. Loetud aadressil: https://www.brightspot.com/resources/what-is-jamstack-pros-and-cons Kasutatud: 31.03.2020

[28] Stackshare, *Django vs Node.js*, 2021. [Online]. Loetud aadressil: https://stackshare.io/stackups/django-vs-nodejs Kasutatud: 10.04.2020

[29] UpGrad, Rohan Vats, *Django vs NodeJS: Difference Between Django and NodeJS*, 2021. [Online]. Loetud aadressil: https://www.upgrad.com/blog/django-vs-nodejs-difference-between-django-and-nodejs/#:~:text=Node%20JS%20is%20based%20on,real%2Dtime%20and%20more%20quickly. Kasutatud: 11.04.2020

[30] Educative, *Django vs. NodeJS*, 2021. [Online]. Loetud aadressil: https://www.educative.io/edpresso/django-vs-nodejs Kasutatud: 15.04.2020

[31] GeeksForGeeks. *Difference Between Django and Node.js*2021. [Online]. Loetud aadressil: https://www.geeksforgeeks.org/difference-between-django-and-node-js/ Kasutatud: 16.04.2020

[32] Intellipaat, *Django vs Node.js - Difference Between Django and Node.js,* 2021. [Online]. Loetud aadressil: https://intellipaat.com/blog/django-vs-nodejs-difference/ Kasutatud: 8.05.2020

[33] Dzone, *Node.js vs. Django: Is JavaScript Better Than Python?,* 2021. [Online]. Loetud aadressil: https://dzone.com/articles/nodejs-vs-djangois-javascript-better-than-python Kasutatud: 01.05.2020

[34] Monocubed,Jigar Mistry, *Django vs Node.js: A Comparative Analysis* ,2021. [Online]. Loetud aadressil: https://www.monocubed.com/django-vs-node-js/ Kasutatud: 13.02.2020

[35] Tutorialspoint, *Spring Framework – Overview,* 2021. [Online]. Loetud aadressil: https://www.tutorialspoint.com/spring/spring_overview.htm Kasutatud: 13.02.2020

[36] Marcbehler.com, *What is Spring Framework? An Unorthodox Guide*, 2021. [Online]. Loetud aadressil: https://www.marcobehler.com/guides/spring-framework Kasutatud: 13.02.2020

[37] Tutorialspoint, *Node.js - Introduction* 2021. [Online]. Loetud aadressil: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm Kasutatud: 18.03.2020

[38] Xplenty,Mark Smallcombe ,*SQL vs NoSQL: 5 Critical Differences* ,2021. [Online]. Loetud aadressil: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/#:~:text=SQL%20databases%20are%20relational%2C%20NoSQL%20are%20non%2Drelational.&text=NoSQL%20databases%20have%20dynamic%20schemas,graph%20or%20wide%2Dcolumn%20stores. Kasutatud: 01.04.2020

[39] IBM, *SQL vs. NoSQL Databases: What's the Difference?* [Online]. Loetud aadressil: https://www.ibm.com/cloud/blog/sql-vs-nosql Kasutatud: 11.04.2020

[40] Hadoop360, Jenny Richards, *Advantages and Disadvantages of NoSQL databases – what you should know*,2021 ,[Online]. Loetud aadressil: https://www.hadoop360.datasciencecentral.com/blog/advantages-and-disadvantages-of-nosql-databases-what-you-should-k Kasutatud: 15.04.2020

[41] GeeksForGeeks, *Advantages and Disadvantages of SQL,* 2020. [Online]. Loetud aadressil: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-sql/ Kasutatud: 24.02.2020

[42] E-Teatmik 2021. [Online]. Loetud aadressil: http://vallaste.ee/index.asp Kasutatud: 12.05.2020

[43] E. Radik, „Document,“ -, 31 Märts 2021. [Online]. Available: https://github.com/LevranST/Brask-Server-Side [Kasutatud 13 Mai 2021].

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

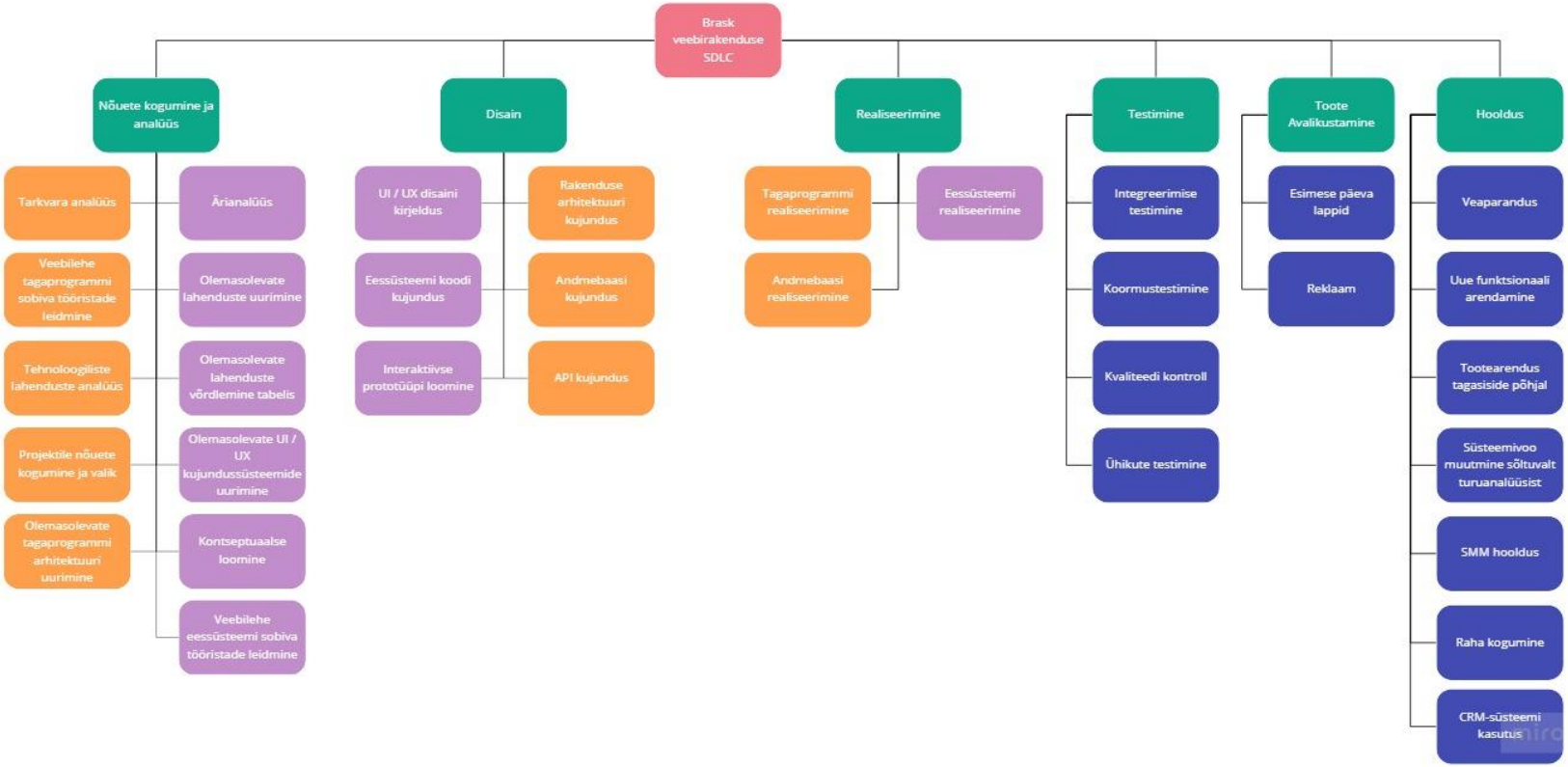
Mina, Erko Radik

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Serveri rakenduse loomine teadmiste jagamise platvormi jaoks“, mille juhendaja on Nadežda Furs
   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.

13.05.2021

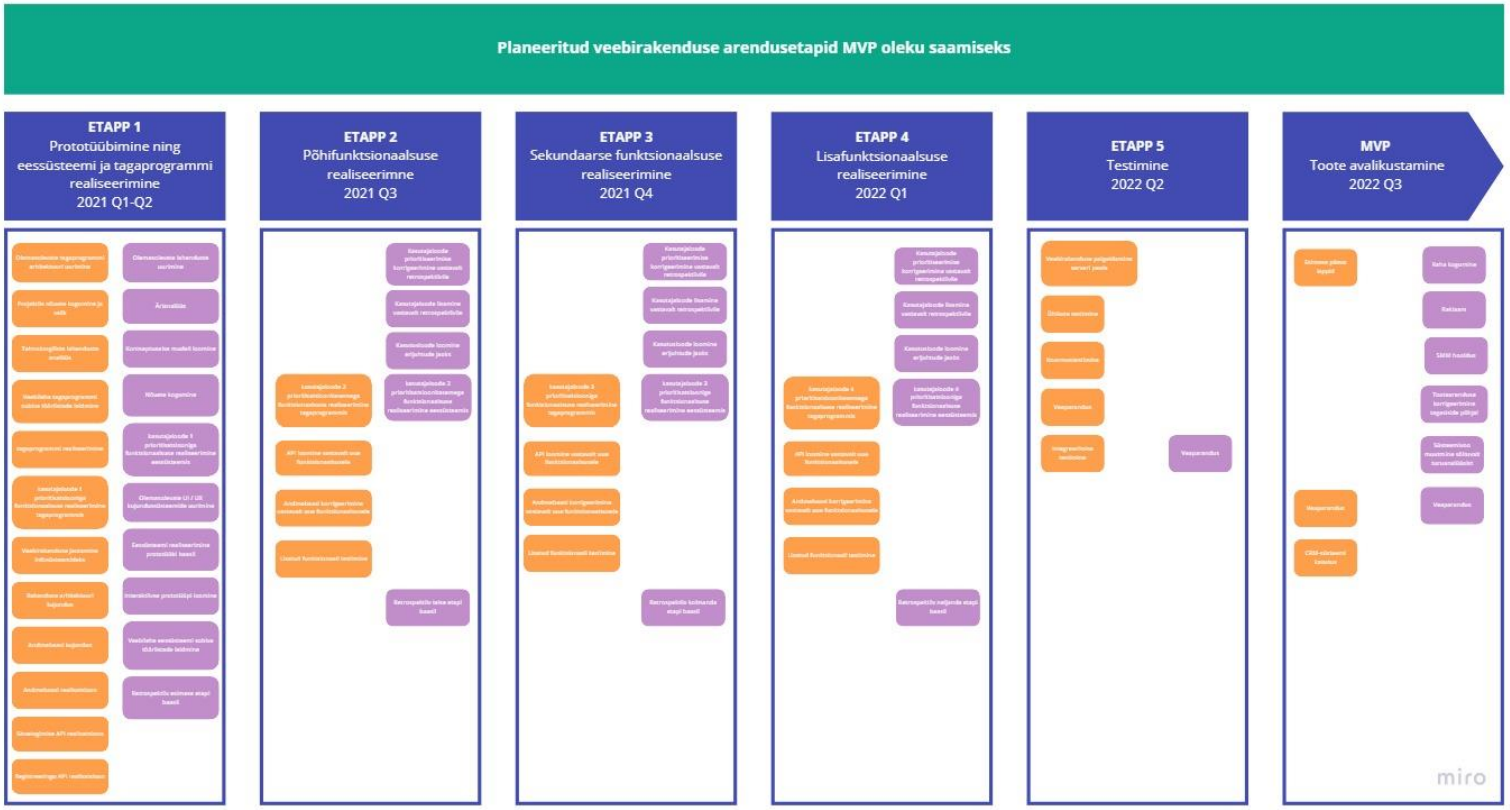
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või 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 - Töö jaotus *SDLC* faaside järgi



Joonis 1. Töö jaotus SDLC faaside järgi

Lisa 3 – Arenduse etappide skeem



Joonis 2: Arenduse etappide skeem

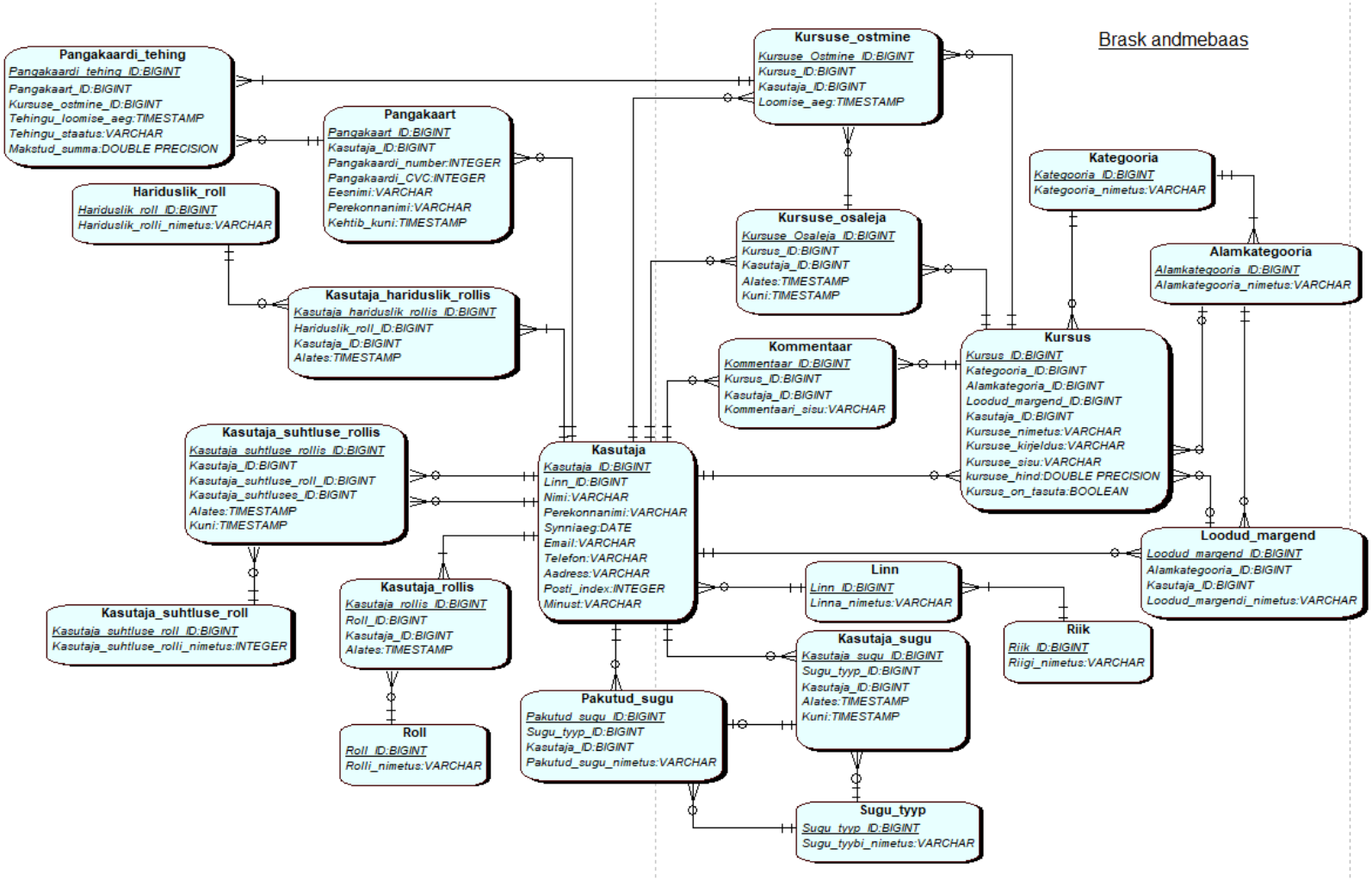
Lisa 4 – *EPICs* loetelu veebilehte kirjutamiseks

Tabel. *EPIC*'ute tabel

Epic ID	Epic Name	Epic ID	Epic Name
EP_1	Login to a Website	EP_23	Mentor certificates content page
EP_2	Registering in Website	EP_24	Courses content page
EP_3	Top-page	EP_25	Created courses page
EP_4	Main-category content page	EP_26	Workshop creation page
EP_5	Sub-category content page	EP_27	Workshop history page
EP_6	Q&A content page	EP_28	Student certificates content page
EP_7	Q&A question branch page	EP_29	Students' accomplishment page
EP_8	Bulletin content page	EP_30	Student accomplishment page
EP_9	Bulletin details	EP_31	Dashboard page
EP_10	Workshops content page	EP_32	Foreign account content page
EP_11	Account content page	EP_33	Subscription page
EP_12	Navigation bar	EP_34	Workspace page
EP_13	Course details	EP_35	"Ask a question" page
EP_14	Workshop stream page	EP_36	Mentions page
EP_15	Account side bar	EP_37	Payment system integration
EP_16	Search result page	EP_38	Q&A question branch creator page
EP_17	Account pop-up menu	EP_39	"Create a bulletin" page
EP_18	Shopping cart page	EP_40	Created bulletins page
EP_19	Messenger content page	EP_41	Bulletin details creator page

EP_20	Notification content page	EP_42	Filtration system
EP_21	Settings content page	EP_43	Ranking system
EP_22	Friends content page	EP_44	Course info page

Lisa 5 – Rakenduse andmebaasi skeem



Joonis 6: Rakenduse andmebaasi skeem