TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Liis Talsi 176236IDDR

Tõlkeprojektide haldusteenuse loomine Paxfuli näitel

Diplomitöö

Juhendaja: Meelis Antoi
Magistrikraad

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: Liis Talsi

15.05.2021

Annotatsioon

Käesoleva diplomitöö eesmärk on luua rakendus tõlkeprojektide haldamiseks Paxfulis. Paxful on kiiresti kasvav ja suuresti välisturgudele suunatud ettevõte, kus uusi tõlgitavaid projekte lisandub pidevalt. Hetkel on uute projektide tõlkesüsteemi lisamine keeruline ja aeganõudev tegevus ning sellele tuleks leida lahendus.

Antud töö põhirõhk on analüüsida sobivaid tehnoloogilisi võimalusi lahenduse elluviimiseks. Lisaks sellele tuuakse välja rakenduse funktsionaalsed ja mittefunktsionaalsed nõuded, kirjeldatakse arhitektuuri ning antakse soovitused edasiseks arenduseks.

Arenduse käigus luuakse veebirakendus, mis võimaldab uusi projekte kiiresti tõlkeprotsessi kaasata. Kliendipoolse lahendusena arendatakse üheleheküljeline veebirakendus, kuhu vastavaid andmeid sisestades saadakse edasised juhendid, kuidas uus projekt tõlkeprotsessi ühendada. Serveripoolse rakenduse ülesanne on tegeleda APIde abil ühenduse loomisega vastava tõlkefaili ja tõlkehaldussüsteemi vahel. Kogu arendusprotsessist antakse erinevates peatükkides põhjalik ülevaade. Arendusprotsessi tulemuseks on töötav rakendus, mida on võimalik hiljem vajadustele vastavalt täiustada.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 45 leheküljel, 6 peatükki ja 9 joonist.

Abstract

Development of a Translation Projects Management Service based on Paxful

The aim of current thesis is to create a web application for handling translation projects at Paxful. Paxful is a fast-growing company that targets foreign markets and new translation projects are needed constantly. The process of adding new translation projects into the established translation flow is currently complex and time-consuming, so a good solution needs to be found.

The emphasis of the work is on the analysis of possible technical solutions for implementing the application. Additionally, the functional and non-functional requirements are identified, architecture is described and future development possibilities are suggested.

During the development process a web application is created which will allow for a seamless set-up for onboarding new projects to the current flow. Client-side solution is a single page application that collects the necessary data about the project and returns instructions for connecting the project with the translation process. Server-side application takes care of the connections between the translation files and translation management system using API calls. The development process is described in different paragraphs. The outcome of the development process is a working web application that can be improved as needed in the future.

The thesis is in Estonian language and contains 45 pages of text, 6 chapters and 9 figures.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* – rakendustarkvara liides
Backend	Teenusepoolne keskkond (server, rakendus ja andmebaas)
CSS	*Cascading Style Sheets* - veebilehtede küljendamisel kasutatav märgistuskeel
DOM	*Document Object Model* – dokumendi objektimudel
Frontend	Kasutajaliides, kliendipoolne keskkond
HTML	*Hyper Text Markup Language* - veebilehe märgendamise keel
JSON	*Javascript Object Notation* - Javascriptil põhinev lihtsustatud andmevahetusvorming
MVC	*Model-view-controller* - mudel, mis jagab tarkvara kolmeks eraldiseisvaks osaks – mudeliks, vaateks ja kontrolleriks
OKTA	Pilvetarkvara, mis aitab ettevõtetel hallata ja kaitsta kasutajate autentimist rakendustes ning arendajatel ehitada identiteedikontrolli rakendustesse, veebisaitide veebiteenustesse ja seadmetesse.
REST	*Representational State Transfer* – tarkvara arhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid
SOAP	*Simble Object Access Protocol* - veebiteenusega suhtlemise liidese protokoll
SPA	*Single page application* - üheleheküljeline veebirakendus
Technology stack	Tehnoloogiapinu

Sisukord

Jooniste loetelu

1 Sissejuhatus

Selleks, et tänapäevases digitaliseeruvas maailmas ära kasutada rahvusvahelistele turgudele suunatud tarkvaratoodete ja -teenuste maksimaalset potentsiaali, tuleb aru saada nende lokaliseerimise olulisusest. Autori arvates on tarkvara toodete puhul eriti tähtis, et kasutaja tunneks ennast toodet kasutades võimalikult mugavalt. Kindlasti mängib siin olulist rolli üleüldine toote disain ja kasutajamugavus, kuid oma kindel osa on kindlasti ka tekstilisel sisul.

Uuringud on näidanud, et ainult 25,9% interneti kasutajatest maailmas kasutavad seda inglise keeles [1]. Seega, tuleks rahvusvahelistele turgudele suunatud toote puhul kindlasti arvestada sellega, et toode oleks ka kõikides teistes keeltes peale inglise keele arusaadav ja lihtne kasutada. Tänapäeval paljud ettevõtted, kaasaarvatud Paxful, arvestavad sellega ning on valmis panustama, et nende toode oleks pidevalt kõikides keeltes ajakohane ja kõlaks võimalikult naturaalsena igas keeles kasutaja jaoks. Seetõttu on oluline, et tarkvara toodete tõlkeprotsess oleks võimalikult kiire ja veatu. Siiani ei ole keegi sellise tõlkehaldussüsteemiga, kus kogu tõlgete haldamise protsess oleks täielikult automatiseeritud, veel välja tulnud ning paljud ettevõtted kulutavad tõlgete haldamisele palju väärtuslikku tööaega.

Käesolev diplomitöö annab ülevaate Paxfuli hetkel kasutusel oleva tõlkeprojektide haldamise protsessi probleemist ja puudujääkidest ning pakub lahenduseks veebirakenduse, mis võimaldab uusi tõlkeprojekte kiiresti olemasolevasse tõlkehaldussüsteemi ühendada.

Diplomitöö autor on ise lokaliseerimise projektijuhina Paxfulis töötanud veidi üle aasta ja on jõudnud järeldusele, et sellisest veebirakendusest oleks palju abi. See aitaks kindlasti kokku hoida palju väärtuslikku tööaega ja muudaks kogu protsessi kiiremaks ja lihtsamaks.

1.1 Taust

Paxful on ettevõttena tegutsenud veidi üle 5 aasta. Tegemist on platvormiga, mis võimaldab inimestel üle 300 erineva maksemeetodiga omavahel krüptorahadega kaubelda. Paxfuli peamiseks tooteks on veebirakendus, mis on hetkel tõlgitud 20nesse keelde, kuid nendele on peagi lisandumas veel 30 uut keelt. Lisaks veebirakendusele vajavad tõlkeid ka väiksemad tooted/projektid - nagu mobiilirakendused, kasutajatugi, erinevad turunduse materjalid jne. Nendele on pidevalt lisandumas veel palju väiksemaid tooteid/projekte mis samuti kõik tõlkimist vajavad.

1.2 Probleem

Kõik Paxfuli tooted/projektid on väga erineva ülesehitusega ning vajavad kõik eraldi ühendamist Paxfuli tõlkehaldussüsteemiga, mis on kolmanda osapoole toode, Crowdin. Crowdiniga ühenduse loomine on keeruline ja aeganõudev protsess. Iga uue projekti ühendamiseks tuleb kaasata tootejuht või arendaja ning seega kulutatakse väärtuslikku tööaega.

Et uus projekt Crowdinis tõlkimiseks üles seada, tuleb kõigepealt lokaliseerimise projektijuhil luua sealses süsteemi keskkonnas uus projekt. Seejärel tuleb toote arendajal luua endale Crowdini kasutaja, uurida põhjalikult Crowdini dokumentatsiooni, et aru saada kõigist võimalustest projekti seadistamiseks ning seejärel teha vastavad muudatused projektis. Peale seda saab lokaliseerimise projektijuht lisada projektile vajalikud keeled, luua kokkulepped tõlkijatega ning koolitada neid uue projekti spetsiifikaga. Kuna kogu see protsess on aeganõudev ja tüütu, siis oleks vaja leida lahendus, kuidas uue projekti tõlkimiseks ülesse seadmine kõikide osapoolte jaoks kiiremaks ja lihtsamaks muuta.

Kõigele lisaks on Crowdinist tõlgete tagasi saamine aeglane ja piiritletud. Crowdin võimaldab nimelt korraga maksimaalselt 20 üheaegset API päringut, millest jääb ettevõtte vajaduste täitmiseks väheks [2].

Eesmärk on luua süsteem, mis hoiab endas kõikide Paxfuli toodete/projektide *stringe* ehk sõnesid ning omab otse ühendust tõlkehaldussüsteemiga (Crowdin), muutes sellega uute toodete/projektide tõlkeprotsessiga ühendamise lihtsamaks ning tõlgete uuendamise toodetes kiiremaks, tagades sellega alati kõige ajakohasemad tõlked.

2 Metoodika

Käesoleva diplomitöö käigus selgitatakse esmalt lähemalt loodava süsteemi skoopi, pannakse paika nõuded, analüüsitakse uue lahenduse võimalusi ning arendatakse uuele süsteemile prototüüp eesmärgiga selgitada välja kas see täidab ettevõtte ootused ja vajadused.

Analüüsi käigus tuuakse selgelt välja süsteemile vajalikud nõuded ning need on grupeeritud vastavalt funktsionaalseteks ja mittefunktsionaalseteks nõueteks. Nendele lisandub ka nimekiri funktsionaalsustest, mida hiljem süsteemile lisada. Analüüsi teises pooles uurib diplomitöö autor uue süsteemi erinevaid võimalikke tehnilisi lahendusi ning põhjendab tehtud valikuid.

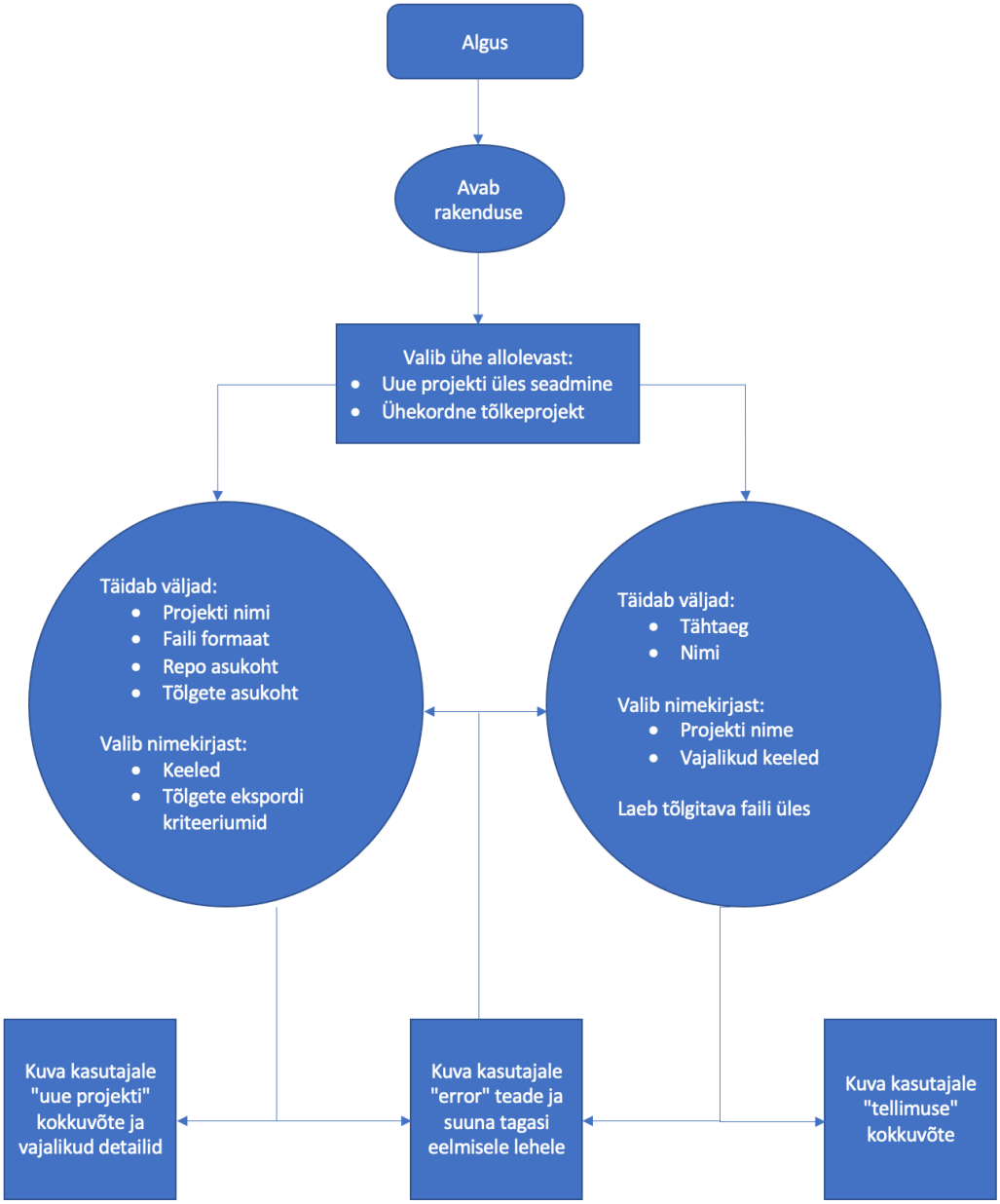
Rakenduse arenduse osas annab autor ülevaate loodud infosüsteemi erinevatest arenduse etappidest, kirjeldab nii kasutatud tehnoloogiaid, tööriistu, metoodikaid kui ka meetodeid. Töö lõpus annab autor hinnangu loodud infosüsteemi toimimisele ja suuna edasiste lisafunktsionaalsuste arendamiseks.

2.1 Skoop

Loodava rakenduse peamine eesmärk on lihtsustada uute projektide ühendamist olemasoleva tõlkeprotsessiga. Teiseks on oluline kiirendada tõlgitud tekstide uuendamist toodetes ning kolmandana on oluline luua kasutajaliides mille kaudu saab iga tootejuht ise kerge vaevaga uusi projekte tõlkeprotsessi lisada.

Rakendus peaks olema lihtsasti ligipääsetav kogu maailmas, kuna Paxfuli esindusi asub nii USAs, Euroopas, Aafrikas kui ka Aasias [3].

Valmis peab saama veebirakendus, mis oleks kõigile Paxfuli töötajatele kättesaadav, mille abil oleks igal ühel võimalik oma projekt tõlkimiseks kiiresti üles seada ja mis võimaldaks tõlkeid projektides pidevalt uuendada. Järgnevalt on joonisel 1 välja toodud süsteemi toimimist illustreeriv blokkskeem.



Joonis 1. Süsteemi toimimist illustreeriv blokkskeem

3 Analüüs

3.1 Nõuete määramine

Nõuete määramisel on arvesse võetud, et probleemi käsitletakse eelkõige Paxfuli kontekstis ning loodav lahendus peab ühilduma olemasolevate keskkondade ja teenustega.

Nõuded põhinevad kasutajalugudel, kus peamisteks rollideks on tootejuhid/arendusjuhid kes vajavad uute toodete ühendamist Paxfuli tõlkehaldussüsteemiga ja projektijuhid, kes aegajalt vajadusepõhiselt soovivad kampaaniate või eriprojektide raames mõne konkreetse dokumendi tõlkimist.

3.1.1 Funktsionaalsed nõuded

Tootejuhi-põhised funktsionaalsed nõuded:

- Tootejuhina soovin sisse ja välja logida.
- Tootejuhina soovin lisada uusi projekte tõlkehaldussüsteemi.
- Tootejuhina soovin määrata uuele projektile sobiva nimetuse.
- Tootejuhina soovin määrata uuele projektile vajalikud keeled.
- Tootejuhina soovin määrata millist faili formaati ma soovin kasutada.
- Tootejuhina soovin määrata tõlgitava faili (*source*) asukoha.
- Tootejuhina soovin määrata tõlgitud failidele asukoha.
- Tootejuhina soovin määrata kui tihti ja millises staadiumis tõlkeid uuendatakse.

Projektijuhi-põhised (ühekordne tõlge) funktsionaalsed nõuded:

- Projektijuhina soovin sisse ja välja logida.
- Projektijuhina soovin valida projekti kategooria (turundus/muu).
- Projektijuhina soovin vajaliku tõlkefaili üles laadida.
- Projektijuhina soovin määrata projektile vajalikud keeled.
- Projektijuhina soovin määrata projektile sobiva tähtaja.
- Projektijuhina soovin saada teavitust tõlgete valmisoleku kohta.

3.1.2 Mittefunktsionaalsed nõuded

Rakendusele kehtivad mittefunktsionaalsed nõuded:

- Süsteem peab toetama populaarsemaid seadmeid.
- Süsteem peab toetama populaarsemaid brausereid.
- Süsteem peab toetama populaarsemaid ekraani suuruseid.
- Süsteem peab olema arendatud inglise keeles, arvestades ettevõtte teiste töötajatega ning võimalike edasiarendustega rahvusvahelises meeskonnas.
- Süsteem peab olema lihtsasti edasi arendatav.
- Kasutajaliides peab olema kasutatav inglise keeles.
- Kasutajaliides peab olema lihtsasti arusaadav ja hea disainiga.

3.1.3 Hilisemad funktsionaalsused

Alljärgnevalt on välja toodud võimalused ja soovitused rakenduse edasiarendusteks:

- Ettevõtte töötajana soovin siseneda süsteemi OKTA[1] kontoga.
- Nii toote- kui projektijuhina soovin, et süsteem võimaldaks saada ülevaadet minu loodud projektidest ja neid vajadusel muuta või kustutada.
- Nii toote- kui projektijuhina soovin süsteemi sisse logides saada ülevaadet projekti(de) tõlgete progressist.

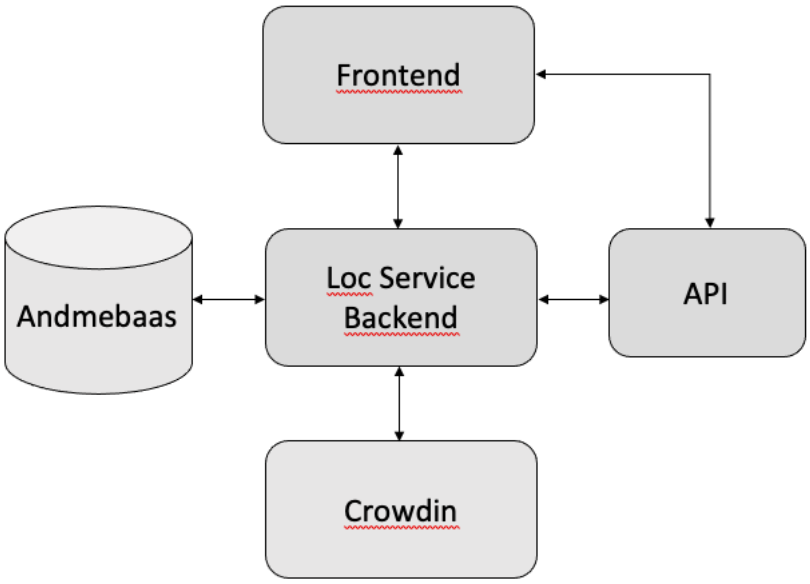
[1] Pilvetarkvara, mis aitab ettevõtetel hallata ja kaitsta kasutajate autentimist rakendustes ning arendajatele ehitada identiteedikontrolli rakendustesse, veebisaitide veebiteenustesse ja seadmetesse.

3.2 Arhitektuur

Loodava veebirakenduse üldine arhitektuur koosneb suures pildis järgnevatest komponentidest:

- Klientrakendus ehk *frontend*
- API
- Teenusepoolne keskkond ehk *backend*
- Andmebaas
- Tõlkehaldussüsteem - Crowdin

Järgnevalt on joonisel 2 välja toodud loodava veebirakenduse üldine arhitektuuriline ülevaade.



Joonis 2. Rakenduse üldine arhitektuuriline ülevaade

Tegemist on veebirakendusega, mis on mõeldud töötama kasutaja brauseris (*frontend*). Infovahetus rakenduse siseselt toimub läbi rakendusliidese (API). Andmebaasis hoitakse projektiga seonduvat infot võti-väärtus paaridena ning selle eesmärk on tagada andmete pidev kättesaadavus ja ajakohasus. Crowdinis toimub sõnede tõlkimine ning kogu tõlkeprotsessi haldus. Teenusepoolse keskkonna (*backend*) üldine ülesanne on tagada kõikide rakenduse komponentide omavaheline ladus toimimine. Alljärgnevalt analüüsib töö autor iga komponendi tehnilise lahenduse võimalusi.

3.3 Tehnoloogia valik

Sobiva tehnoloogia valimisel on väga oluline lähtuda konkreetse projekti vajadustest. Kindlasti ei ole soovitatav tehnoloogiat valida konkurendi lahendust kopeerides, isiklikel eelistustel põhinedes, eelmiste projektide kogemusest või internetist leitud soovitustest lähtudes. Tehnoloogia valimisel tuleks järgida järgnevaid kriteeriumeid [4]:

- Rakenduse tüüp - tuleb mõista kas tegu on lihtsa, keskmise või keeruka projektiga.
- Arenduse kulud - arendaja ajakulu ja teenuse hoolduskulud.
- Turvalisus - tuleb veenduda, et teenus järgib kõiki turvanõudeid.
- Paindlikkus - teenus peaks olema kergesti ühildatav teiste kasutatavate tehnoloogiatega.

3.3.1 Klientrakenduse tehnoloogia

Klientrakenduse tehnoloogiliste lahenduste valik on tänapäeva veebimaailma äärmiselt kiireloomulist arengut silmas pidades üsnagi keeruline. Tehniliselt tegelevad kliendirakenduse raamistikud ja teegid siiani HTMLi, CSSi ja Javascripti käitamisega, sest need on alustalad, millel kliendipoolne veebimaailm seisab. Siiski on iga vähegi relevantne raamistik või teekide kogum oma filosoofilise ja stilistilise lähenemise poolest erinev. Lähemalt on uuritud järgmisi tänapäeval populaarseid tehnoloogiaid:

- ReactJS - Avatud lähtekoodiga JavaScripti teek mis on loodud Facebooki poolt ning on mõeldud interaktiivsete kasutajaliideste arendamiseks. ReactJS baseerub komponendi põhisel arhitektuuril, mis võimaldab jagada klientrakendust mitmeks erinevaks osaks ning võimaldab seega komponente ka teistes rakendustes taaskasutada [5]. ReactJSi plussidena võib veel välja tuua virtuaalse DOMi (dokumendi objektimudeli) mis tagab suure jõudluse ja efektiivsuse [6].

- AngularJS - Samuti avatud lähtekoodiga JavaScriptil põhinev struktureeritud raamistik mis on loodud Google poolt dünaamiliste veebirakenduste loomiseks [7]. Raamistiku peamiseks eesmärgiks on lihtsustada üheleheküljeliste veebirakenduste arendust. AngularJS paneb rõhku koodi kvaliteedile ja testitavusele, mis on ka üheks peamiseks põhjuseks miks paljud arendajad AngularJSi teistele populaarsetele raamistikele eelistavad [8].

- Vue.js - Progressiivne JavaScripti raamistik kasutajaliideste loomiseks mis on samuti avatud lähtekoodiga [9].

- jQuery - Üks enimkasutatavaid JavaScripti teeke mis on avatud lähtekoodiga, kiire, väike ja teenuste rohke ning aitab lihtsustada ja standardiseerida HTML elementide ja JavaScripti koodi vahelist suhtlust [10].

- Next.js – Reacti raamistik modernsete veebirakenduste ehitamiseks. Raamistik lisab palju võimsaid lisafunktsioone nagu serveripoolne esitamine (*rendering*), automaatne koodi jaotamine, järkjärguline staatiline regenereerimine ja palju muud, mis teevad skaleeruvate ja tootmiseks valmis rakenduste loomise hõlpsamaks.

Kõikidel eelpool mainitud raamistikel ja teekidel on omad eelised ja puudused ning tehniliselt sobiksid antud rakenduse loomiseks nad kõik. Otsuse langetamisel osutusid kõige olulisemateks teguriteks ettevõttes juba kasutusel olevad tehnoloogiad ja veebiarenduse maailmas levivad trendid.

3.3.2 Teenusepoolse keskkonna tehnoloogia

Teenuse ehk serveripoolse keskkonna tehnoloogiapinu valikul tuleb arvestada selle peamiste komponentidega, milleks on:

- Programmeerimiskeel
- Teegid ja raamistikud
- Andmebaas
- Veebiserver
- Operatsioonisüsteem
- Veebimajutus

Järgnevalt on lähemalt uuritud iga komponendi tehnilise lahenduse võimalusi.

3.3.2.1 Programmeerimiskeele valik

Erinevaid serveripoolseid programmeerimiskeeli on palju ning nende populaarsus ajas kõigub pidevalt. Seega on võimatu väita, et mõni keel on tunduvalt parem kui teine - neil kõigil on lihtsalt omad tugevad ja nõrgad küljed. Siinkohal toob töö autor välja praeguse aja mõned populaarseimad:

- Java - Peamised Java eelised: platvormist sõltumatu, objektorienteeritud ja lihtne õppida [11].

- PHP – Skriptimiskeel, mida kasutatakse peamiselt serveripoolsetes lahendustes dünaamiliste veebilehtede loomisel. Toetab objektorienteeritud programmeerimist ja andmebaasiga suhtlemist ning on laialdaselt kasutatud [12].

- C# - Modernne, üldise otstarbega, objektorienteeritud keel, mis on välja töötatud Microsofti poolt nende .Net raamistikus. Seda keelt peetakse küllaltki lihtsasti õpitavaks ja ta baseerub C ja C++ keeltel. Lisaks eelnevalt mainitule on C# ka platvormist sõltumatu, kiire, tüübi-kindel, kergesti skaleeruv, pidevalt täiustatud ning hästi struktureeritud programmeerimiskeel [13].

- Ruby - Dünaamiline, avatud lähtekoodiga programmeerimiskeel, mis on disainitud olema lihtne ja produktiivne. Tal on elegantne süntaks, mida on lihtne ja mugav lugeda ja kirjutada [14]. Peamisteks eelisteks on stabiilsus, paindlikus, lihtsus, kiirus ja objektorienteeritus [15].

- Python - Populaarne, interaktiivne, platvormist sõltumatu, objektorienteeritud, üldise otstarbega, avatud lähtekoodiga programmeerimiskeel mida võib kasutada mitmetel eesmärkidel. Pythonit peetakse lihtsa süntaksiga keeleks, mida on lihtne lugeda ja kirjutada [16].

- JavaScript - Dünaamiline, objektorienteeritud, mitmeotstarbeline, prototüübipõhine programmeerimiskeel, mida kasutatakse tänapäeval nii dünaamiliste kasutajaliideste loomiseks kui ka serveripoolseks rakenduste arenduseks [17].

Kuna Next.JS on raamistik, mis võimaldab ja hõlbustab nii veebirakenduse kliendi- kui ka serveripoolset arendust ning toimib Reactiga sünergias, siis sobib hästi antud rakenduse arenduses seda ära kasutada ning luua ka rakenduse teenusepoolne lahendus JavaScripti kasutades. Lisaks sellele on antud töö autoril JavaScriptiga olnud eelnevalt kõige rohkem kokkupuudet ning ka ettevõttes on palju spetsialiste, kes JavaScripti väga hästi tunnevad ning saavad vajadusel kiiresti aidata.

3.3.2.2 Raamistike ja teekide valik

Selleks, et rakenduse teenusepoolne arendus võimalikult lihtsalt ja kiiresti ellu viia tuleb kindlasti kasutusele võtta ka sobilik raamistik, mis aitaks juba eksisteerivaid tehnoloogiaid ära kasutada ning võimaldaks vältida juba olemasolevate lahenduste nullist arendamist. Siinkohal toob töö autor välja mõned tuntumad teenusepoolsed raamistikud:

Spring – Avatud lähtekoodiga Javal põhinev raamistik. Spring raamistik pakub terviklikku programmeerimise ja konfigureerimise mudelit modernsete Java põhiste rakenduste arenduseks [18].

Ruby on Rails - Dünaamiline veebirakenduste raamistik, ideaalne kiirete rakenduste loomiseks. Loodud David Heinemeier Hansson poolt 2005 aastal [19]. Ruby on Rails rakendused on üldiselt kümme korda kiiremad kui klassikalised Java raamistikud [20]. Programmeerimiskeel: Ruby

Laravel - Lihtne, elegantne ja kergesti loetav. Loodud 2011 aastal Taylor Otwell poolt. Sarnaselt teistele kaasaegsetele raamistikele funktsioneerib ta MVC arhitektuurilisel mudelil. Laravelil on API tugi ja ta sisaldab endas suurt hulka teeke, mis võimaldavad tal kiiresti laieneda. Laravel sisaldab ka palju lisafunktsioone nagu serveri poolne *rendering* ehk esitlus ja "sõltuvuste süstimine" [20]. Programmeerimiskeel: PHP

Django - Usaldusväärne raamistik, mille peamiseks eesmärgiks on lihtsustada keerukate andmebaasipõhiste veebirakenduste arendamine, ehitatud MVC arhitektuuril. Loodud 2005 aastal Django Software Foundation (DSF) poolt. Kindlasti tuleks silmas pidada, et Django on monoliitne, tihedalt ühendatud toode, mis sunnib oma kasutajaid etteantud raamidesse [19]. Programmeerimiskeel: Python

ASP.NET - Tasuta, platvormide ülene, vabavaraline raamistik veebiteenuste ja rakenduste loomiseks kasutades selleks .NETi ja C#. ASP.NET on kiire ja skaleeruv, turvaline, aktiivne kommuun ja tasuta veebimajutus Azure'is [21]. Programmeerimiskeel: C#

Node.js - Populaarne, vabavaraline JavaScripti käituskeskkond, mis võimaldab tarkvara arendajatel luua nii klient- kui ka serveripoolseid veebirakendusi. Loodud aastal 2009 ning on peale seda oma populaarsust järjest kasvatanud. Koosneb sisse ehitatud teekidest mis toimivad kui veebiserver [19]. Programmeerimiskeel: JavaScript

Eelnevatest töös välja toodud tehnoloogiapinu komponentide valikutest tulenevalt võetakse antud rakenduse loomisel kasutusele Node.js. Node.js ainsa valikuna on loodud just JavaScripti rakendustega töötamiseks, sest sisuliselt on ta JavaScripti käituskeskkond ning seega sobib ideaalselt käesoleva rakenduse konteksti. Sarnaselt teistele valikutele lihtsustab Node.js arendaja elu mitmete abistavate meetoditega, millest pikemalt rakenduse arenduse peatükkides juttu tuleb.

3.4 Andmebaasi valik

Andmebaas on struktureeritud teabe või andmete organiseeritud kogum, mida tüüpiliselt hoitakse elektrooniliselt arvutisüsteemis. Andmebaasi kontrollib tavaliselt andmebaaside haldussüsteem (DBMS).

Andmebaasid on alates nende loomisest 1960ndate aastate alguses, märgatavalt arenenud. Navigeerimisandmebaasid nagu hierarhiline andmebaas (mis tugines puulaadsel mudelil ja võimaldas ainult üks-mitmele suhet) ja võrkmudel andmebaas (paindlikum mudel, mis võimaldas ka mitu-mitmele suhet) olid algsed süsteemid, mida kasutati et andmetega manipuleerida. Ehkki oma ülesehituselt lihtsad, olid need varased süsteemid paindumatud. 1980ndatel aastatel muutusid populaarseks relatsioonilised andmebaasid, millele järgnesid objektipõhised andmebaasid 1990ndatel aastatel. Sellele järgnesid NoSQL-i andmebaasid, mis ilmusid vastusena Interneti kasvule ning vajadusele ka struktureerimata andmeid kiiremini töödelda. Täna on andmete kogumise, salvestamise, haldamise ja kasutamise osas tegemas murrangulisi edusamme aga just pilvandmebaasid ja isejuhtivad andmebaasid [22].

Andmebaase on palju erinevaid. Parim andmebaas konkreetse projekti jaoks sõltub sellest, kuidas on projektis andmeid ette nähtud kasutada. Suures plaanis jagunevad andmebaasid kahte tüüpi kategooriasse: Relatsioonilised ja mitte-relatsioonilised (NoSQL) andmebaasid:

- Relatsioonilised andmebaasid - Relatsioonilised andmebaasid muutusid domineerivaks 1980ndatel aastatel. Relatsioonilise andmebaasi andmeüksused on omavahel relatsioonidega (seostega) ühendatud. Neid relatsioone esitatakse põhimõtteliselt tabelitena, kusjuures veergudes kujutatakse andmevälju ja ridades andmeüksusi ehk kirjeid. Relatsiooniline andmebaasitehnoloogia pakub kõige tõhusamalt ja paindlikumalt juurdepääsu struktureeritud teabele. Näited: MySQL, Oracle, Microsoft SQL, MariaDB

- Mitte-relatsioonilised ehk NoSQL andmebaasid - NoSQL ehk mitte-relatsiooniline andmebaas võimaldab struktureerimata ja pool struktureerimata andmeid salvestada ja nendega manipuleerida (erinevalt relatsioonilisest andmebaasist, mis määratleb selgelt, kuidas kõik andmebaasi sisestatud andmed peavad struktureeritud olema). NoSQL andmebaasid muutusid populaarseks, kui veebirakendused muutusid üha tavalisemaks ja keerukamaks. Näited: MongoDB, Redis, Cassandra, Elasticsearch

Erinevat tüüpi mitte-relatsioonilised andmebaasid:

- Dokument-orienteeritud andmebaasid – Talletavad kogu objekti kohta käiva teabe andmebaasis ühes eksemplaris ja iga salvestatud objekt võib teisest erineda. Andmed salvestatakse võti-väärtus paaridena JSON dokumentidesse. Sellised andmebaasid on ülimalt paindlikud. Sellesse kategooriasse kuuluvad mõned kõige kuulsamad NoSQL andmebaasid nagu Couchbase ja MongoDB.

- Võti-väärtus andmebaasid – Seda tüüpi andmebaasi salvestatakse andmed võtme-väärtus paaridena ja võtit kuvatakse vaid üks kord. See on lihtne ja kerge viis andmete salvestamiseks ja neile juurdepääsuks, kuid ka küllaltki ebaefektiivne. Mõned näited on Amazon DynamoDB ja Redis.

- Veeruhoidla põhised andmebaasid - Seda tüüpi võib nimetada ka mitmemõõtmeliseks võtme-väärtuste salvestuseks, kuna see salvestab ja haldab

tohutut andmemahtu tabelites või mitmes veerus, millest iga veerg võib toimida kirjena. See aitab suure hulga andmete skaleerimisel. Märkimisväärsed näited on Scylla, HBase ja Cassandra.

- Graafandmebaasid - Näitavad ühendusi erinevate andmepunktide vahel. Neid kasutatakse enamasti siis, kui on vaja analüüsida erinevat tüüpi andmeid ja nende omavahelist suhet. Need on kujutatud omavahel seotud objektide või sõlmede võrgustikuna. Näiteks on Datastax Enterprise Graph ja Neo4J.

Nagu juba eelnevalt mainitud on andmebaaside tüübid pärast nende esmakordset kasutusele võtmist palju muutunud ja tänapäeval arendatakse pidevalt uusi andmebaase juurde. Igal tüübil, mida modernsed süsteemid tänapäeval kasutatavad on erinevad eelised, mida tasub uurida, arvestades õigeid juurdepääsumustreid, andmete omadusi ja nõudeid. Üks esimesi ja kõige olulisemaid otsuseid uue projektiga alustamisel on projekti vajaduste hindamine ja nõudmistele vastava andmebaasi tüübi leidmine [23].

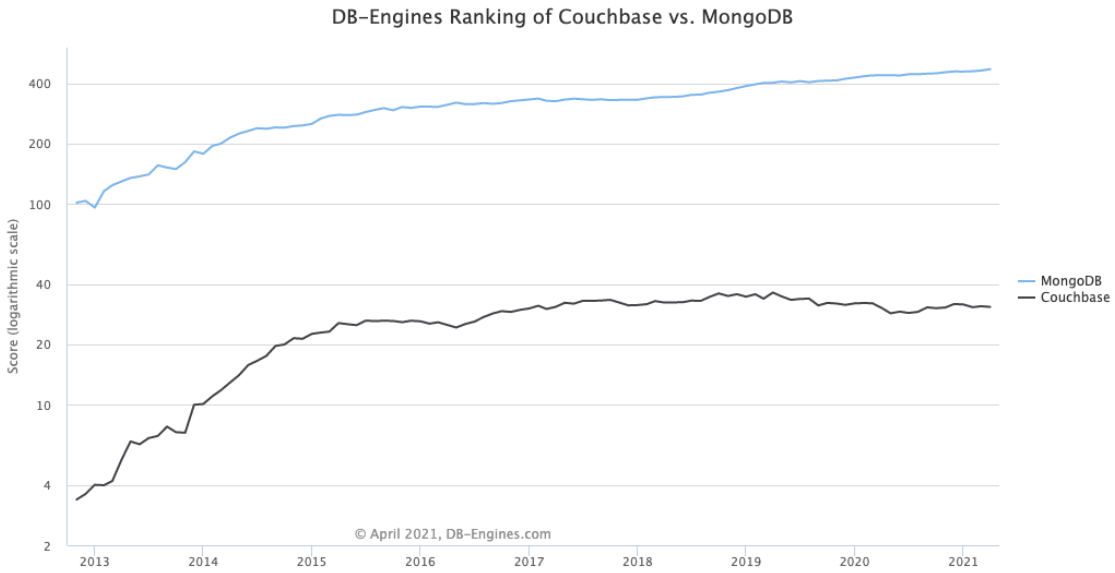
Antud lõputöö autoril puudub eelnev kokkupuude dokument-orienteeritud andmebaasiga töötamisel, kuid kuna antud projekti raames tundub see valik just kõige sobilikum siis otsustati see järele proovida. Seda peamiselt seetõttu, et loodav rakendus ei pea tegelema suurel hulgal väga selgelt struktureeritud andmete tabelitega ning dokumendi kujul andmetega töötamisest piisab täielikult antud projekti nõuete ja vajaduste täitmiseks. Tuntumad dokument-orienteeritud andmebaasid on MongoDB ja Couchbase. Järgnevalt tutvustab töö autor mõlemat võimalikku varianti lähemalt.

MongoDB – Avatud lähtekoodiga dokument-orienteeritud andmebaas, mis on üles ehitatud horisontaalse laiendamise arhitektuurile. 2007. aastal asutatud MongoDB on kasutusel arendajate kogukonnas üle kogu maailma. Selle asemel, et andmeid ridade või veergude kaupa tabelites salvestada (nagu SQL-i andmebaasid), on MongoDB andmebaasi iga rida dokument, mille vormingukeeleks on JSON [24].

Couchbase - Avatud lähtekoodiga, hajutatud, NoSQL-i dokument-orienteeritud andmebaas interaktiivsetele veebirakendustele. Couchbaseil on paindlik andmemudel, ta on hõpsasti laiendatav ning tagab püsivalt kõrge jõudluse.

Üldiselt on MongoDB ja Couchbase oma olemuselt üksteisele väga sarnased, kuid MongoDB on dokument-orienteeritud andmebaasidest kõige populaarsem ning lisaks

sellele on MongoDB ka arendajate kogukonnas eelistatum ja kõrgemalt hinnatud kui Couchbase (Joonis 3 - MongoDB ja Couchbase populaarsuse võrdlus) [25]. Seetõttu osutus ka antud rakenduse andmebaasi valikuks just MongoDB.



Joonis 3 - MongoDB ja Couchbase populaarsuse võrdlus

3.5 Arenduskeskkonna valik

Arenduskeskkond on protseduuride ja tööriistade kogum rakenduse või programmi arendamiseks, testimiseks ja silumiseks. Seda terminit kasutatakse mõnikord sünonüümselt integreeritud arenduskeskkonnaga (IDE), mis on tarkvara arendamise tööriist, mida kasutatakse programmi kirjutamiseks, ehitamiseks, testimiseks ja silumiseks. Samuti pakuvad nad arendajatele ühist kasutajaliidest (UI) erinevate režiimide arendamiseks ja silumiseks [26].

Antud lõputöö autoril on kogemust siiani olnud nii Intellij Idea, Sublime Texti, kui ka Visual Studio Code'iga. Erinevaid integreeritud arenduskeskkondi on küll veel teisigi, kuid antud töö raames uuritakse lähemalt just neid kolme võimalikku varianti:

Intellij Idea – JetBrainsi toodete. See on avatud lähtekoodiga ja seda saavad kolmandad osapooled kasutada IDE-de loomiseks, näiteks Google'i Android Studio. See pakub komponendipõhist platvormidevahelist JVM-põhist rakenduste hosti koos

kõrgetasemelise kasutajaliidese tööriistakomplektiga tööriistakende, puuvaadete ja palju muu loomiseks.

Sublime Text – C++ ja Pythonis kirjutatud Sublime Text on platvormidevaheline lähtekoodiredaktor, millel on Pythoni rakenduste programmeerimisliides (API). Selle koodiredaktori funktsioonide hulka kuuluvad Pythoni-põhine pistikprogrammi API, projektispetsiifilised eelistused ja palju muud.

Visual Studio Code - Visual Studio Code on kõige populaarsem lähtekoodiredaktor ning on kindlustanud arendajate seas populaarseima arenduskeskkonnana kõrgeima positsiooni. See on võimas ja kerge pilvepõhine koodiredaktor, millel on sisse ehitatud JavaScripti, TypeScripti ja Node.js tugi ning millel on rikkalik laienduste ökosüsteem ka teiste keelte jaoks (nt C ++, C #, Java, Python, PHP Go) [27].

Ka antud rakenduse arenduskeskkonnaks valiti Visual Studio Code, kuna seda on väga mugav kasutada ning see aitab arendustööd kiirendada.

3.6 Koodihaldustarkvara valik

Koodihaldustarkvara valiku võimalustest uuritakse lähemalt kolme kõige populaarsemat tasuta varianti:

- GitLab on avatud lähtekoodihoidla ja ühine tarkvaraarendusplatvorm suurte DevOps ja DevSecOps projektide jaoks. GitLab pakub veebipõhist koodi salvestamise asukohta ning väljaannete jälgimise ja CI / CD võimalusi. Hoidla võimaldab majutada erinevaid arendusahelaid ja versioone ning võimaldab kasutajatel eelmiste versioonide koodi kontrollida ja ettenägematute probleemide korral selle juurde tagasi liikuda. GitLab on paljude teiste seas konkurendiks GitHubile. Kuna GitLab on oma versioonikontrolli samuti välja töötatud Giti alusel, toimib see lähtekoodi haldamisel väga sarnaselt. GitLabi järjepideva integreerimise (CI) võimalused võimaldavad arendusmeeskondadel oma koodi loomist ja testimist automatiseerida. GitLab toetab nii avaliku kui ka erasektori arendusi ning on üksikisikutele tasuta. Seevastu mõned konkurendid, näiteks Bitbucket, küsivad privaatses hoidlas üle viie kasutaja eest tasu [28].

- Bitbucket Cloud on meekondadele loodud Giti põhine koodi majutamise ja koostöö tööriist. Bitbucketi oma klassi parimad Jira ja Trello ühendused on loodud selleks, et kogu tarkvarameeskond projekti elluviimiseks kokku viia. Bitbucket pakub meeskondadele ühtset koha, et teha koostööd koodi kontseptsioonist pilveni, ehitada automaatse testimise abil kvaliteetset koodi ja seda kindlalt juurutada [29].

- GitHub on ettevõte, mis pakub pilvepõhist Giti hoidla hostimisteenust. Sisuliselt muudab see üksikisikute ja meeskondade jaoks Giti kasutamise palju lihtsamaks. GitHubi liides on piisavalt kasutajasõbralik et ka algajad koodijad saaksid Giti ära kasutada. Ilma GitHubita nõuab Giti kasutamine üldiselt natuke tehnilisemat teadmist ja käsurea kasutamise oskust. GitHub on isegi nii kasutajasõbralik, et mõned inimesed kasutavad GitHubi isegi muud tüüpi projektide haldamiseks - näiteks raamatute kirjutamiseks. Lisaks saavad kõik registreeruda ja majutada tasuta avalikku koodihoidlat, mis teeb GitHubi avatud lähtekoodiga projektide hulgas eriti populaarseks [30].

"Tasuta privaatsed repod", "Lihtne seadistamine" ja "Kena kasutajaliides ja tööriistad" on võtmetegurid, miks arendajad eelistavad tihti Bitbucketit, kuid "hajutatud versioonikontrollisüsteem", "tõhus hargnemine ja ühendamine" ja "kiire" on peamised põhjused, miks sageli eelistatakse GitHubi [31]. GitHubil on lihtsam ja intuitiivsem liides kui BitBucketil (vastavalt suurele arendajate rühmale, kes on mõlemat platvormi kasutanud). GitHub on eriti hea algajatele, kes soovivad omandada giti ja veebi lähtekoodi *hostimise*. Platvormil on hulgaliselt õppematerjale ja juhendeid, et selle kasutamisega kiiresti alustada ning vajadusel abi leida [32]. Nendel põhjustel sai ka antud rakenduse koodihoidlaks valitud just GitHub.

3.7 Analüüsi kokkuvõte

Tehnoloogiliste lahenduste analüüsist, süsteemile loodud nõuete ja rakenduse arhitektuurist tulenevalt ilmnes, et rakendus ja selle äriloogika peavad olema veebiteenuse kujul, mis kasutab REST API põhimõtteid. Sellele lisandub klientrakendus ehk kasutajaliides, mis suhtleb teenusega API päringute kaudu.

Antud lõputöö raames loodav rakendus saab olema lihtne, üheleheküljeline veebirakendus (SPA). SPA on veebirakendus või veebileht, mis suhtleb veebibrauseriga, kirjutades kasutajale kuvatavat veebilehe vaadet pidevalt dünaamiliselt ümber veebiserverist saadud uute andmetega, selle asemel, et brauser laeks pidevalt uuesti terveid veebilehti. SPA on ideaalne lahendus erakordse ja rikkaliku suhtluse loomiseks kasutaja ja rakenduse vahel. SPA eesmärk on kiiremad üleminekud, mis muudavad veebilehe võimalikult sujuvaks rakenduseks [33].

Põhjusel, et töö autoril on olnud varasemalt kõige rohkem kogemust just JavaScriptiga ning seda kasutatakse ettevõtte siseselt ka just serveripoolsetes lahendustes järjest rohkem siis valis autor rakenduses kasutada programmeerimiskeelena JavaScripti. Lisaks sellele on JavaScript hetkel üks populaarsemaid arenduskeeli ning seda kasutatakse palju [10]. See annab lootust, et hätta jäädes on võimalik kergesti abi leida või küsida.

Järgnevalt toob töö autor välja mõningad eelised, miks nii kliendipoolse kui ka serveri poolse rakenduse arendamiseks JavaScriptil põhinevad tehnoloogiad valida.

- Arendustöö efektiivsus - mitme erineva keelega töötamine on keeruline, eriti algajal arendajal. Seega, kuna lõputöö autoril on JavaScriptiga eelnevalt kõige rohkem kokkupuudet olnud siis leiti, et oleks mõistlik proovida, kuidas töötaks välja lahendus mis baseeruks ainult ühel programmeerimiskeelel.

- Koodi jagamine ja taaskasutus - koodi erinevaid komponente saab kogu rakenduse ulatuses taaskasutada.

- Kiirus ja jõudlus – JavaScripti raamistikud võimaldavad luua väikeseid, kuid suure jõudlusega rakendusi [34].

- Suur hulk tasuta tööriistu - JavaScripti raamistikega kaasneb palju erinevaid vabavaralisi teeke ja mooduleid [35].

Paljuski valitud arenduskeelest tulenevalt otsustati nii serveri, kui ka kliendipoolse rakenduse lahendused baseeruda Next.js raamistikul, kasutades kliendi poolel lisaks Reacti ja serveri poolel Node.jsi. React valiti peamiselt selle populaarsuse tõttu, kuigi lõputöö autoril endal sellega väga palju eelnevat kogemust ei ole. Sellegi poolest kasutatakse seda Paxfulis järjest rohkem.

Andmebaasi valikul osutus sobilikumaks paindliku andmestruktuuriga NoSQL tehnoloogia, mille rakendamine muudab arendamise efektiivsemaks ja muudatustele vastuvõtlikumaks. Nagu juba ka eelnevalt sai mainitud siis antud projekti raames loodava rakenduse puhul sobiks ilmselt mõlemad võimalused, nii MongoDB kui ka Couchbase, kuid kuna mitmetele allikatele tuginedes tundus siiski MongoDB lihtsamini kasutatav, selle kohta on piisavalt abimaterjale internetist kättesaadavad ning ta on oma kategooria tänapäeva üks populaarsemaid valikuid siis osutus valituks just MongoDB [25].

Töö autor otsustas arenduseks kasutada Visual Studio Code'i, kuna oma kogemusele tuginedes on see tänapäeval JavaScripti rakenduste arendusel kõige mugavam kasutada. Ta on kiire, mugav ja autorile juba tuttav tööriist, millega töötamist ei pea hakkama lisaks õppima.

Koodihaldus keskkonnaks valiti GitHub, kuna sellega on autoril eelnevalt lihtsalt kõige rohkem kogemust ja selle õppimiseks ei pea lisaks eraldi aega kulutama. Lisaks sellele kasutatakse seda ka Paxfuli siseselt ja see võimaldab prototüübi loomisel ettevõttes juba kasutusel olevaid tehnoloogiaid ära kasutada.

Vercel – Selleks et *production* ehk lõppkasutajatele kompileeritud kood Node.js käituskeskkonda juurutada võeti töös kasutusele Vercel. Vercel kasutab automaatselt projekti põhiharu (*master branchi)* viimast koodi versiooni. See lihtsustab kogu juurutamise, kompileeritud ja avalikustatud rakenduse testimise protsessi. Ühtlasi on sellel mugav liides turvalisel viisil keskkonna muutujate (environmental variables) seadmiseks, mis salvestatakse krüpteeritult Verceli keskkonda. Lisaks pakuvad nad *production* veebirakenduse minimaalset teenuse silumist Node.js serveri logi reaalajas kuvamise näol.

4 Veebirakenduse arendus

Veebirakenduse arenduse ülevaade jaguneb laias laastus nelja suuremasse peatükki: teenusepoolse rakenduse arendus, klientrakenduse poolne arendus, kirjeldus andmebaasi ülesse seadmisest ning viimasena on lähemalt selgitatud APIde arendust.

Kogu veebirakendus on suures plaanis loodud Next.js najale, kuna see võimaldab tänu oma serveripoolse esitamise (*rendering*) võimalusele kiiresti ja lihtsalt luua Reacti rakendusi ning lisaks sellele pakub Next.js ka väga mugavat failisüsteemi põhist APIde marsruutimise loomist. Automaatselt ülesse seatud API lõpp-punktid võimaldavad mugavalt teha serveripoolseid päringuid Crowdini API vastu. Klientrakenduses ei saa tihti turvalisuse piirangute tõttu selliseid päringuid teha.

4.1 Teenusepoolne arendus

Veebirakenduse *backend* on ehitatud Next.js raamistiku peale ning see omab ühendust andmebaasiga, kus hoitakse kogu kliendipoolse rakenduse äriloogikaga seotud andmeid.

Selleks, et arvutis oleks võimalik ilma brauserita JavaScripti käitada, on vaja paigaldada Node.js (node.js.org). Sellega tuleb kaasa ka NPM ehk *Node Package Manager*, mille abil hallatakse teeke ning jooksutatakse serverit. Need kaks on aluseks sellele, et järgmiste sammudega edasi minna.

Uue Next.js projekti kataloogi loomiseks koos vajalike sõltuvustega tuleb sisestada käsk terminali: npx create-next-app. Sellega lisatakse projekti ka teegid: ’Next’ ’React’ ja ’React-dom’.

Seejärel on baasprojekt olemas - töötav veebiserver näidis lahendusega on loodud ja projekti teegid on kirjas package.json failis (Joonis 4. Ülevaade esialgse projekti teekidest koodinäitena).

```
{
  "name": "loc-service",
  "version": "0.1.0",
  "private": true,

  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start"
  },
  "dependencies": {
    "next": "10.1.3",
    "react": "17.0.2",
    "react-dom": "17.0.2",
  }
}
```

Joonis 4. Ülevaade esialgse projekti teekidest koodinäitena

Esimesed 3 rida kirjeldavad projekti *metadatat* ning selle järgnevad skriptid ja sõltuvused. Skriptid viitavad rakenduse arenduse erinevatele etappidele:

"dev": skript, mis käivitab Next.js arendusserveri. Skripti kasutatakse rakenduse käitamiseks arendusrežiimis. See tähendab, et kood töötab spetsiaalse tõrkeotsingu, kiirlaadimise ja muude funktsioonidega, mis muudavad arendusprotsessi meeldivamaks.

"build": skript, mis kompileerib projekti. Skript loob /.next kataloogi projektist kompileeritud ja optimeeritud versiooni.

"start": skript, mis käivitab Next.js tootmisserveri. Skript käivitab koodi /.next kataloogis kui rakendus on tootmiskeskonnas.

Käsuga „npm run dev“ käivitatakse arendusserver ning aadressil http://localhost:3000/ saab brauseris jälgida juba arenduse käigus loodud veebirakendust [36].

Väga kasulik meetod on ka *Hot-reloding,* mis tähendab seda, et nii kui tehakse muudatus failis siis veebirakendus uueneb automaatselt ja muudatused ilmnevad koheselt ka rakenduses.

Veel üheks oluliseks osaks Next.js juures on *Pages* kataloog. Selles kataloogis asuvaid faile kasutatakse loodavas rakenduses faili nimede alusel veebilehtede marsruutidena. Iga failinimi *Pages* kataloogis vastab URLi viitele brauseris. Näiteks fail mille asukohaks on */pages/articles.js* siis vastab see brauseris aadressile */articles*. Kogu marsruutimise

loogikat haldab Next.js automaatselt. Fail */pages/index.js* on ainuke erand, kuna see toimib avalehena. Lisaks sellele teeb Next.js mugavaks see, et ta toetab ka dünaamiliste marsruutidega lehti. Näiteks kui luua fail nimega *pages/posts/*[id]*.js*, on see juurdepääsetav aadressidel *posts/1, posts/2* jne. See lihtsustab oluliselt arendustööd ning selle arvelt saab arendaja enda aega suunata muu funktsionaalsuse loomisele.

Muude vajalike moodulite lisamine. Osa mooduleid tulevad Next.js poolt automaatselt, osad tuleb lisada ise vastavalt vajadusele. Lisatud moodulid:

"@babel/core": "^7.13.15", - transpileerimise moodul, muudab koodi selliseks mida mõistavad ka vanemad brauserid.

"@babel/plugin-proposal-decorators": "^7.13.15", - eksperimentaalsete dekoraatorite haldamise teek, moderne JavaScript veel ei toeta, tema muudab selle kasutatavaks ilma hoiatusteta.

"axios": "^0.21.1", - HTTP päringute tegemise teek, mis võimaldab lihtsustatud kujul HTTP päringuid teha nii *Frontendis* kui *Backendis.*

"mongodb": "^3.6.6", - Mongo DB draiver - liides, mis paneb Node.js mõistma Mongo DBga suhtlemist.

"next": "10.1.3", - veebirakenduse raamistik.

Peale seda on lisatud moodulitest tulenevalt lihtsustatud komponentide stiilimine (Styled Components) , lihtsustatud HTTP päringute tegemine (Axios), võimalus suhelda MongoDBga (mongodb), eksperimentaalsete dekoraatorite kasutamine (babel).

4.2 Klientrakenduse poolne arendus

Kogu loodud veebirakenduse klientrakenduse staatiline ülesehitus asub failis index.js. Selleks, et klientrakenduse arendusprotsessi kiirendada ja lihtsustada lisati projekti veel järgnevad JavaScripti moodulid ja raamistikud:

- "react": "17.0.2", - UI raamistik, mis töötab sümbioosis Next.jsiga
- "react-dom": "17.0.2", - Reacti pakett, mis lihtsustab DOM'iga töötamist ning kiirendab oluliselt veebilehe kuvamist.

- "@material-ui/core": "^4.11.3", - Reacti üks populaarseim UI elementide teek, mis võimaldab lihtsasti erinevaid UI elemente luua ja stiilida [37].

- "@material-ui/icons": "^4.11.2", - Reacti ikoonide loomiseks ja stiilimiseks mõeldud teek.

- "styled-components": "^5.2.3" – Teek, mis võimaldab JavaScriptis lihtsal viisil CSSi kirjutada. Lubab teha iga klassi komponendid eraldi nii et ei teki ühtegi CSS konflikti. Kasutab sisuliselt sama süntaksit nagu CSS ise, kuigi ta on JavaScript. See võimaldab luua ühe JavaScripti faili kus on stiilid ja loogika samas failis koos selliselt, et konflikte ei teki.

Kuna disaini poolest on loodav klientrakenduse lahendus väga lihtsa struktuuriga, otsustati luua *Single Page Application* (SPA) ehk üheleheküljeline rakendus. Esimese sammuna on klientrakenduse poolt vajalik luua vorm, millest oleks kasutajal lihtne aru saada ja mida oleks kerge täita. Selle kaudu peaks kasutaja sisestama kõik uue tõlkeprojekti loomiseks vajaliku informatsiooni. (Lisa 2 - Klientrakenduse vaade – loo uus tõlkeprojekt).

Vormi vaate loomisel sai kasutatud nii Material UI teegi komponente, kui ka Syled Components teegi funktsioone, mis võimaldasid lihtsasti vormi elemente kujundada. Näiteks võiks siinkohal välja tuua Material UI komponendi *Paper*, mille sai Styled Componentsi meetodi abil uue loodud komponendi sisse mähkida ja seejärel uuele komponendile CSSi kasutades vajalikke stiile lisada (Joonis 5. Koodinäide *Styled Components* teegi kasutusest).

```
const NavigationPaper = styled(Paper)`
    margin-bottom: 42px;
`;
const ContentPaper = styled(Paper)`
    padding: 32px;
`;
```

Joonis 5. Koodinäide *Styled Components* teegi kasutusest

Kui vormi vaade valmis, siis hakati looma funktsionaalsust. Mida teha kasutaja poolt sisestatud andmetega peale seda, kui ta vajutab „Create new project“ nuppu. Kui kasutaja teeb midagi, mille kohta rakendusel on infot vaja, siis tema tegevus kajastub Reacti komponendi *stateis*. Kui kasutaja valib „Create new project“ siis funktsioon, mis tegeleb

stateiga kontrollib üle tingimused ja kui midagi on valesti siis on seal ka osaliselt sisse ehitatud valideerimine. HTML5 ja brauserid teevad valideerimisega tänapäeval head tööd. Juhul kui vorm on ilusti täidetud siis võtab süsteem *State* objektist sisestatud andmed ja koos Axiose mooduli abiga tehakse HTTP POST päring API/Create Project API marsruudi pihta. Sealt võtab asjad üle äriloogika osa.

Teise sammuna loodi klientrakenduse teine vaade, kus kasutajale kuvatakse loetelu loodud projektidest (Lisa 3 - Klientrakenduse põhivaade). Selleks et vastavat infot selles vaates kasutajale kuvada tehakse Axiosega HTTP kaudu GET päring API marsruudi /API/projekts pihta.

Kui projekt on lisatud ja seal on nupp „Get translations links“ siis sellele nupule vajutades sooritatakse HTTP POST päring /API/translate/[id] pihta ning edasise võtab üle äriloogika osa.

Osa päringud toimuvad brauseri ja API vahel (nt. Create project), kuid Next.js võimaldab teha HTTP päringuid ka serveripoolelt. See on hea siis, kui mingil põhjusel on vaja teha päringuid, mis brauseri turvalisuse kaalutlustel (Cors ja muud sellised) on muidu problemaatilised. Näide sellest on GET päring API/projects lõpp-punkti pihta, mis antud projektis asub *getServerSideProps* funktsioonis, et serveri poolelt kuvada dünaamilist sisu.

4.3 Andmebaasi ülesseadmine

Analüüsi tulemusel sai andmebaasi lahenduseks valitud MongoDB ning selle majutamiseks MongoDB Atlas. Andmebaasi ülesseadmine ja projektiga ühendamine toimus järgnevalt kirjeldatud etappide läbi.

Kuna see oli antud lõputöö autori jaoks esimene kokkupuude MongoDB andmebaasiga, siis tuli kõigepealt luua MongoDB Atlasesse konto. MongoDB Atlas võimaldab andmeid hoida pilves ning tänu sellele on erinevatest teenustest ja seadmetest nendele lihtne ligi pääseda. Atlas tegeleb kogu juurutamise keerukusega ja haldamisega valitud pilveteenuse pakkujal (AWS, Azure või Google Cloud) [38]. Antud töö puhul sai pilveteenuse pakkujaks valitud AWS. Atlasesse saab tasuta luua ühe kobara ehk *Cluster*, mille sisse saab teha mitmeid andmebaase. Tasuta kontol on küll mahulised piirangud, kuid antud projekti raames sellest piisab.

Kui konto seadistamine Atlases tehtud, siis järgmise sammuna tuli rakendus andmebaasiga siduda. Kuna Next.jsi server toimib Node.jsi toel, siis sai andmebaasiga ühenduse loomise hõlbustamiseks kasutada MongoDB Nodei draiverit. Selleks tuli projekti lisada vastav moodul käsuga: *npm install mongodb –save*. See lisas projekti MongoDB draiveri ja sisestas vajalikud sõltuvused *package.json* faili. Seejärel tuli lisada MongoDB Atlasest andmebaasi loomisel saadud ühenduse sõne ehk *connection string* koos teiste seadistusega faili *.env.local* (Joonis 6. Koodinäide andmebaasi seadistamisest**Error! Reference source not found.**)

```
MONGODB_URI=mongodb+srv://<USERNAME>:<PASSWORD>@cluster.tdm0q.mongodb.net
MONGODB_DB=LocService

HOSTNAME=localhost
PORT=3000
HOST=http://$HOSTNAME:$PORT
```

Joonis 6. Koodinäide andmebaasi seadistamisest

Järgmiseks sammuks oligi MongoDB päringute koostamine selleks, et saaks esimese tõlkeprojekti luua. Selleks võeti esmalt kasutusele Next.jsi dokumentatsioonis leiduva abistaja „*helper*“ funktsioon, mis sisuliselt toimib konteinerina MongoDB draiveri ümber ning hõlbustab andmebaasiga ühenduse loomist.

Tänu MongoDB dokumendi mudelile saab andmeid salvestada täpselt sellisel kujul nagu rakendus ette näeb. Ainus väli, mille MongoDB ise lisab on unikaalne id, mis on loodud eristamaks dokumenti andmebaasis. Siinkohal lisan näite uue tõlkeprojekti andmebaasi lisamise päringust (Joonis 7. Koodinäide uue projekti andmebaasi lisamisest).

```
const addToDatabase = (data) => {
            const { id, sourceLanguageId, targetLanguageIds, name,
targetLanguages } = data.crowdinData.data.data;
            const { selectedFileTypes, repoLocation, targetFilesLocation } =
data.requestData;

            const databaseEntry = {
                id,
                sourceLanguageId,
                targetLanguageIds,
                name,
                targetLanguages,
                selectedFileTypes,
                repoLocation,
                targetFilesLocation,
            };

            try {
                const insertObject = async (databaseEntry) => {
                    const { db } = await connectToDatabase();
                    const result = await
db.collection("Projects").insertOne(databaseEntry);

                    console.log(result);
                };

                if (databaseEntry) {
                    insertObject(databaseEntry);
                }

                return {
                    success: true,
                };
            } catch (err) {
                return {
                    success: false,
                    error: err,
                };
            }
        };
```

Joonis 7. Koodinäide uue projekti andmebaasi lisamisest

Siinjuures tuleks välja tuua, et ainsad kohad veebirakenduses, mis otse andmebaasiga suhtlevad on API meetodid. Klientrakenduse ja andmebaasi ning teiste teenustega suhtlemise eest vastutavadki API lõpp-punktid, mida klientrakendus on mõeldud

kasutama. See tähendab, et klientrakendus vastutab visuaalse representatsiooni ja andmete kogumise eest, kuid äriloogika on jäetud eraldiseisva üksuse hooleks.

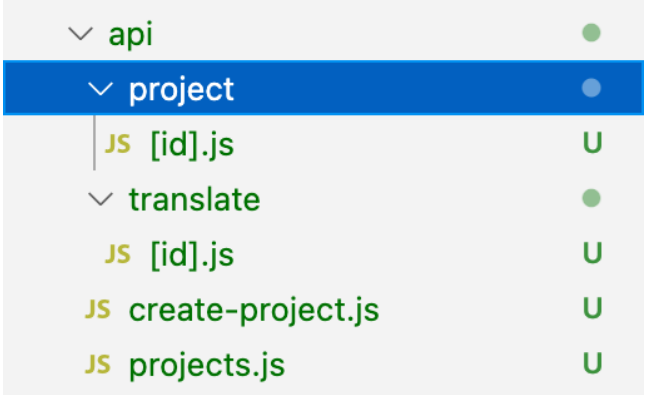
4.4 APIde arendus

Next.js'is on sisse ehitatud failisüsteemi põhine marsruutimine (*routing)*. Kõik failid kaustas ´pages/api´ on kaardistatud ´/api/*´-ga ja neid käsitletakse lehe asemel API lõpp-punktina.

API-marsruudi toimimiseks tuleb eksportida vaikimisi funktsioon (päringu käitleja).

Erinevate HTTP-meetodite käitlemiseks API-marsruudil saab oma päringukäsitluses kasutada ´req.method'meetodit. Käesoleva projekti raames kasutame põhiliselt HTTP GET ja POST meetodeid, sest enamasti tegeleme andmete saatmise ja vastu võtmisega.

API üldine ehitus järgib REST põhimõtteid. Näiteks *„api/project/[id]"* ja *„api/translate/[id]"* lõpp-punktid võimaldavad klientrakendusel dünaamiliselt andmeid pärida ja saata. Siinkohal toon näite REST API dünaamilistest lõpp-punktidest (Joonis 8. Näide REST API dünaamilistest API lõpp-punktidest projektis).



Joonis 8. Näide REST API dünaamilistest API lõpp-punktidest projektis

Kandilised sulud koos [id] parameetriga võimaldavad tänu Next.jsi rakenduse loogikale äärmiselt lihtsalt sedasorti lõpp-punkte luua ja ühtlasi päringu sisu päringukäitluse funktsioonides edasi kasutada.

```
import { connectToDatabase } from "../../../util/mongodb";
import { ObjectID } from "mongodb";

export default async function projectHandler(req, res) {
    const {
        query: { id },
        method,
    } = req;

    switch (method) {
        case "GET":
            try {
                const { db } = await connectToDatabase();
                const result = await db.collection("Projects").findOne({ _id:
new ObjectID(id) });

                res.status(200).json(result);
            } catch (err) {
                res.status(err).json({});
            }

            break;
        case "PUT":
            const { files } = req.body;

            try {
                const updatedProject = {
                    $set: {
                        files: files,
                    },
                };

                const { db } = await connectToDatabase();
                const result = await db.collection("Projects").updateOne({
_id: new ObjectID(id) }, updatedProject);

                res.status(200).json(result);
            } catch (err) {
                res.status(err).json({});
            }

            break;
        default:
            res.setHeader("Allow", ["GET", "PUT"]);
            res.status(405).end(`Method ${method} Not Allowed`);
    }
```

Joonis 9. Koodinäide GET meetodi API marsruudist, et pärida info ühe konkreetse projekti kohta

5 Tulemuste analüüs

Antud lõputöö eesmärgi kirjeldamiseks toodi välja töö skoop funktsionaalsete ja mittefunktsionaalsete nõuete määramisega.

Arenduse käigus saadi valmis rakenduse toimiv prototüüp, mis vastab suures osas esitatud nõuetele. Tegemist on veebirakendusega, mille kaudu on kasutajatel võimalik uus tõlkeprojekt ülesse seada ilma Crowdini otseselt ise kasutamata (Lisa 2 - Klientrakenduse vaade – loo uus tõlkeprojekt). Crowdiniga suhtlemise eest vastutab rakenduse- API ja äriloogikapoolne osa.

Kui projekt on edukalt loodud, on kasutajal võimalus rakenduse esilehel saada ülevaade loodud projektidest ning vastava projekti tõlked alla laadida. Esialgses töö skoobis on küll ettenähtud tõlgete Crowdinist otse sihtkohta eksportimine, kuid kahjuks antud lõputöö tulemusena valminud prototüüpi see funktsionaalsus ei mahtunud. Lisaks sellele on hetkel veel teostamata üksikute staatiliste tõlkeprojektide vahendamise funktsioon. Selle arendamiseks on küll juba suures osas funktsionaalsust olemas, kuid ajalistel kaalutlustel on see osa veel lõpetamata.

Antud lõputöös kasutatud tehnoloogiad osutusid äärmiselt õigustatuks. Eelnevalt oli juba teada, et Paxfulis kasutatakse klientrakenduste arenduses peamiselt Reacti teeke, kuid töö käigus selgus lisaks, et ettevõttesiseselt liigutakse ka Next.js raamistikule. See omakorda lihtsustab kindlasti tuleviks nii loodava rakenduse edasiarendust kui ka teiste ettevõtte siseste tehnoloogiatega integreerimist.

Rakenduse lähtekood ja töötava rakenduse veebiaadress on avalikult kättesaadavad (Lisa 4 – Rakenduse versioonihaldus ja veebiaadress)

Autor õppis töö käigus väga suures mahus enda jaoks uusi tehnoloogiaid ja arendamise võimalusi. Töö autori jaoks oli väga huvitav lugeda ja analüüsida JavaScripti ja sellega kaasnevate tööriistade jõudsat arengut. Töö käigus omandatud oskused seavad kindlasti tugeva alustala loodud rakendust ettevõttes edasi arendada.

6 Kokkuvõte

Peamiseks eesmärgiks oli antud diplomitöö käigus leida lahendus probleemile, kus uute tõlkeprojektide ühendamine ettevõtte tõlkehaldussüsteemi on keeruline ja väärtuslikku ressurssi nõudev protsess.

Töö käigus analüüsiti vajamineva süsteemi funktsionaalseid ja mittefunktsionaalseid nõudeid, uuriti põhjalikult võimalikke tehnilisi lahendusi ning loodi ühte võimalikku lahendust välja pakkudes süsteemist prototüüp. Tegemist on lihtsa üheleheküljelise veebirakendusega, mis suhtleb API'de abil tõlkehaldussüsteemiga ja vastab kõikidele skoobis esitatud tootejuhi-poolsetele funktsionaalsetele nõuetele. Kahjuks ei jõudnud veel lõpule projektijuhi-poolsete funktsionaalsete nõuete arendus, kuigi nende elluviimiseks on kõik eeldused loodud. Kahjuks tuli ajapuuduse tõttu siiski hetkel nõudmistes järeleandmisi teha.

Tuleviku edasiarendustes võiks kindlasti lõpuni viia projektijuhi-poolsete funktsionaalsete nõuete arenduse, nendele võiks lisada ka kõikide töös nõuete määramise osas juba välja toodud tulevikus planeeritud lisafunktsionaalsused.

Paxfuli töötajad näevad senini valmis saanud rakenduse prototüübist lähtudes loodavas rakenduses suurt potentsiaali ning autor usub, et töö on kindlasti praktilise väärtusega ning tulevikus palju kasutatav. Kuna autor teab omast kogemusest, et Crowdin on tõlkehaldussüsteemina kasutusel veel mitmetes Eesti idufirmades, siis ei saa välistada ka võimalust, et loodud rakendust saavad ka teised ettevõtted kunagi ära kasutada.

Kasutatud kirjandus

[1] M. M. Group, „Internet World Stats,“ 2020. [Võrgumaterjal]. Available: https://www.internetworldstats.com/stats7.htm. [Kasutatud 24.04.2021].

[2] Crowdin, „Crowdin API v2,“ 2021. [Võrgumaterjal]. Available: https://support.crowdin.com/api/v2/#section/Introduction/Rate-Limits. [Kasutatud 02.02.2021].

[3] Paxful, „Paxful,“ 2021. [Võrgumaterjal]. Available: https://paxful.com/. [Kasutatud 03.02.2021].

[4] G. B. a. V. V., „RubyGarage,“ jaanuar 2020. [Võrgumaterjal]. Available: https://rubygarage.org/blog/technology-stack-for-web-development. [Kasutatud 15.03.2021].

[5] Facebook, „React,“ 2021. [Võrgumaterjal]. Available: https://reactjs.org/. [Kasutatud 22.04.2021].

[6] K. Shah, „ReactJS - Introduction, Advantages And Disadvantages,“ 7 aprill 2021. [Võrgumaterjal]. Available: https://www.c-sharpcorner.com/article/react-introduction-advantages-and-disadvantages/. [Kasutatud 15.03.2021].

[7] „What Is AngularJS?,“ 2020. [Võrgumaterjal]. Available: https://docs.angularjs.org/guide/introduction. [Kasutatud 15.03.2021].

[8] S. Mehta, „8 Benefits of Using AngularJS for Web App Development,“ 21 august 2020. [Võrgumaterjal]. Available: https://stackify.com/8-benefits-of-using-angularjs-for-web-app-development/. [Kasutatud 15.03.2021].

[9] E. You, „What is Vue.js?,“ 2021. [Võrgumaterjal]. Available: vuejs.org/v2/guide/. [Kasutatud 21.03.2021].

[10] „Developer Survey Results 2019,“ Stack Overflow, 2019. [Võrgumaterjal]. Available: https://insights.stackoverflow.com/survey/2019. [Kasutatud 21.03.2021].

[11] IBM, „Advantages of Java,“ 2020. [Võrgumaterjal]. Available: https://www.ibm.com/docs/en/aix/7.2?topic=monitoring-advantages-java. [Kasutatud 18.02.2021].

[12] w3schools, „PHP Tutorial,“ 2021. [Võrgumaterjal]. Available: https://www.w3schools.com/php/default.asp. [Kasutatud 15 03 2021].

[13] CodeXoxo, „What Is C# Language, Advantages & Features Of C# Language,“ 30 august 2019. [Võrgumaterjal]. Available: https://www.codexoxo.com/advantages-c-sharp-language/. [Kasutatud 21.03.2021].

[14] R. community, „About Ruby,“ 2021. [Võrgumaterjal]. Available: http://www.ruby-lang.org/en/about/. [Kasutatud 24.03.2021].

[15] R. Community, „What are the advantages of Ruby,“ 2021. [Võrgumaterjal]. Available: ruby-lang.co/what-are-the-advantages-of-ruby/. [Kasutatud 24.03.2021].

[16] Opensource.com, „What is Python?,“ 2021. [Võrgumaterjal]. Available: https://opensource.com/resources/python. [Kasutatud 26.03.2021].

[17] Mozilla, „About JavaScript,“ MDN Web Docs, 29 märts 2021. [Võrgumaterjal]. Available: developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Kasutatud 16.03.2021].

[18] „Spring Framework,“ VMware, Inc, 2021. [Võrgumaterjal]. Available: https://spring.io/projects/spring-framework. [Kasutatud 05.04.2021].

[19] K. Oponowicz, „What Backend Technology Should I Choose for My Web Application?,“ 2021. [Võrgumaterjal]. Available: ordergroup.co/en/blog/what-backend-technology-should-i-choose-my-web-application/. [Kasutatud 05.02.2021].

[20] J. Mistry, „10 Popular Web Frameworks for Web App Development in 2021,“ Monocubed, 18 märts 2021. [Võrgumaterjal]. Available: https://www.monocubed.com/10-most-popular-web-frameworks/. [Kasutatud 06.02.2021].

[21] Microsoft, „ASP.NET,“ 2021. [Võrgumaterjal]. Available: dotnet.microsoft.com/apps/aspnet. [Kasutatud 25.03.2021].

[22] Oracle, „What is a Database,“ 2021. [Võrgumaterjal]. Available: www.oracle.com/database/what-is-database/#autonomous. [Kasutatud 16.03.2021].

[23] Prisma, „Introduction to databases,“ [Võrgumaterjal]. Available: https://www.prisma.io/dataguide/intro/comparing-database-types. [Kasutatud 06.03.2021].

[24] MongoDB, „Why and When to Use MongoDB,“ 2021. [Võrgumaterjal]. Available: https://www.mongodb.com/why-use-mongodb. [Kasutatud 15.03.2021].

[25] DB-engines, „DB-Engines Ranking - Trend of Couchbase vs. MongoDB Popularity,“ DB-engines, aprill 2021. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking_trend/system/Couchbase%3BMongoDB. [Kasutatud 17.03.2021].

[26] Techopedia, „Development Environment,“ 2021. [Võrgumaterjal]. Available: https://www.techopedia.com/definition/16376/development-environment. [Kasutatud 21.04.2021].

[27] A. Choudhury, „Top 10 Development Environments Used In 2019,“ 27 november 2019. [Võrgumaterjal]. Available: https://analyticsindiamag.com/top-10-development-environments-used-in-2019/. [Kasutatud 15.04.2021].

[28] TechTarget, „GitLab,“ oktoober 2020. [Võrgumaterjal]. Available: https://whatis.techtarget.com/definition/GitLab. [Kasutatud 16.04.2021].

[29] Atlassian, „A brief overview of Bitbucket,“ 2021. [Võrgumaterjal]. Available: https://bitbucket.org/product/guides/getting-started/overview. [Kasutatud 17.04.2021].

[30] „What Is GitHub? A Beginner’s Introduction to GitHub,“ Kinsta Inc, 11 märts 2021. [Võrgumaterjal]. Available: https://kinsta.com/knowledgebase/what-is-github/. [Kasutatud 17.04.2021].

[31] „Bitbucket vs Git,“ Stackshare, Inc, aprill 2021. [Võrgumaterjal]. Available: https://stackshare.io/stackups/bitbucket-vs-git. [Kasutatud 17.04.2021].

[32] „BitBucket vs GitHub — The Complete Review,“ Codegiant, 18 juuli 2020. [Võrgumaterjal]. Available: https://blog.codegiant.io/bitbucket-vs-github-5af0c766b776. [Kasutatud 17.04.2021].

[33] T. Grabski, „Benefits of Next JS for Building Websites and Apps,“ Pagepro, 9 detsember 2020. [Võrgumaterjal]. Available: https://pagepro.co/blog/benefits-of-using-next-js-for-building-websites-and-apps/. [Kasutatud 03.03.2021].

[34] Mozilla, „Express/Node introduction,“ MDN Web Docs, 17 veebruar 2021. [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. [Kasutatud 03.03.2021].

[35] E. team, „Express middleware,“ [Võrgumaterjal]. Available: http://expressjs.com/en/resources/middleware.html. [Kasutatud 15.03.2021].

[36] „Getting Started with Next.js,“ Vercel, Inc., 2021. [Võrgumaterjal]. Available: https://nextjs.org/docs/getting-started. [Kasutatud 12.03.2021].

[37] „MATERIAL-UI,“ MATERIAL-UI, 2021. [Võrgumaterjal]. Available: https://material-ui.com/ . [Kasutatud 28.04.2021].

[38] „MongoDB Atlas,“ MongoDB, 2021. [Võrgumaterjal]. Available: https://docs.atlas.mongodb.com/. [Kasutatud 27.03.2021].

[39] Codecademy, „What is REST?,“ 2021. [Võrgumaterjal]. Available: https://www.codecademy.com/articles/what-is-rest. [Kasutatud 12.04.2021].

[40] R. Vatwani, „What is REST API?,“ Zestard Technologies Pvt Ltd., 25 veebruar 2020. [Võrgumaterjal]. Available: https://www.zestard.com/blog/rest-api-benefits/. [Kasutatud 21.04.2021].

[41] M. Milošević, „MongoDB vs Couchbase - part two,“ SmartCat, 9 jaanuar 2017. [Võrgumaterjal]. Available: https://blog.smartcat.io/2017/mongodb-vs-couchbase-part-two/. [Kasutatud 04.27.2021].

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

Mina, Liis Talsi

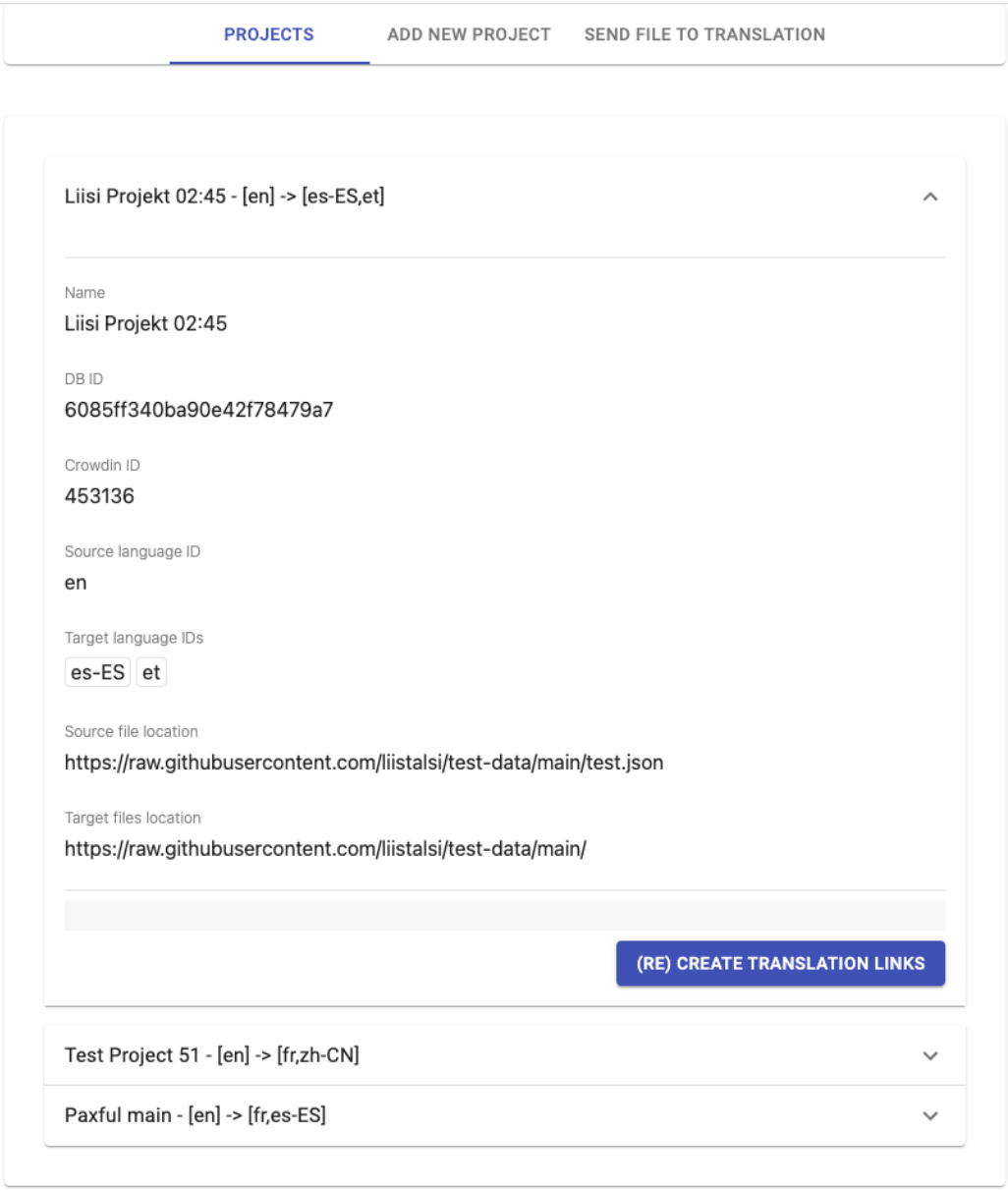
1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tõlkeprojektide haldusteenuse loomine Paxfuli näitel“, mille juhendaja on Meelis Antoi.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse 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.

15.05.2021

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 vaade – loo uus tõlkeprojekt

Lisa 3 – Klientrakenduse põhivaade



Lisa 4 – Rakenduse versioonihaldus ja veebiaadress

Versioonihaldus: https://github.com/liistalsi/LocService

Veebiaadress: https://locservice.vercel.app/