Tartu Ülikool
Arvutiteaduse instituut
Informaatika õppekava

Pärt Veidenberg

Tõlkerakendus mitmekeelsuse haldamiseks

Bakalaureusetöö (9 EAP)

Juhendaja: Vambola Leping
Juhendaja: Reiko Randoja

Tartu 2021

Tõlkerakendus mitmekeelsuse haldamiseks

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärgiks on veebipõhise rakenduse loomine mitmekeelsuse haldamiseks. Töö sisaldab tuntumate tõlkerakenduste/tõlketeekide analüüsi, veebirakendust ja kasutatuid tehnoloogiaid ning rakenduse ülesehituse kirjeldust.

Võtmesõnad:

Veebirakendus, tõlkevõtmed, lokaliseerimine, *JavaScript*, *React.js*, *Node.js*, *GraphQL*

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Translation application for multi-language support

Abstract:

The aim of this Bachelor's thesis is to develop a web application for multi-language support. The thesis contains an analysis of popular localization applications/localization libraries, localization application itself, technologies used and process of developing a web application

Keywords:

Web application, translation keys, localization, *JavaScript*, *React.js*, *Node.js*, *GraphQL*

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Lokaliseerimist aetakse tihti segamini tõlkimisega, aga tegelikult on mõistete tähendused erinevad. Tõlkimine on lokaliseerimise üks osa, aga lokaliseerimine ise on palju ulatuslikum[1]. Lokaliseerimine on kogu protsess, et muuta olemasolev toode kindla regiooni turu jaoks sobivaks, näiteks kuupäeva ja aadressi formaadi muutmine ning mõõtühikute ja rahaühikute teisendamine. Seejuures tõlkimine on protsess, mille käigus muudetakse tekst ühest keelest teise.

Käesoleva bakalaureusetöö eesmärgiks on analüüsida olemasolevaid tõlkerakendusi ja luua ise veebipõhine rakendus teiste veebi- ja mobiilirakenduste tõlgete haldamiseks. Kolmandas peatükis asuvast analüüsist selgub, et olemasolevad tõlkerakendused on väiksemate alustavate projektide jaoks kallid ja võivad sisaldada liigseid funktsionaalsusi. Vastupidiselt võivad vajalikud funktsionaalsused olla puudu. Bakalaureusetöö on jaotatud kolmeks peatükiks. Esimeses peatükis räägitakse kasutatud tehnoloogiatest ja nende valimise põhjustest. Teises peatükis analüüsitakse olemasolevaid tõlkerakendusi. Kolmandas peatükis tutvustatakse veebirakenduse ülesehitust ja tööpõhimõtet.

1 Kasutatud tehnoloogiad

Selles peatükis loetletakse tehnoloogiad, mida tehtud projektis kasutatakse. Räägitakse kasutusele võtmise põhjustest ja kirjeldatakse antud tehnoloogiaid lühidalt.

1.1 Node.js

Node.js on avatud lähtekoodiga (*open-source*) platvormist sõltumatu (*cross-platform*) arendustarkvara, mida kasutatakse serverite ja võrgundus rakenduste arendamiseks. *Node.js'i* rakendus töötab ühe protsessina ja uut lõime ei looda iga päringu kohta. Sisend-väljund operatsioonide nagu võrgust lugemise, andmebaasiga suhtlemise või failisüsteemiga toimetamise ajal ei blokeerita lõime ära ja sellega ei raisata protsessori tööaega ootamise peale. Vastavaid operatsioone jätkatakse, kui neile saabub oodatud vastus. Sellise tööpõhimõttega suudab *Node.js* hallata tuhandeid samaaegseid ühendusi ühes serveris ilma lõimesid üle koormamata.

Node.js üheks suurimaks eeliseks on töötamine *JavaScript* programmeerimiskeelel. Miljonid eessüsteemi arendajad, kes kirjutavad *JavaScript'i* saavad ka kirjutada serveri poolset koodi ning selle jaoks pole vajadust õppida täiesti uut programmeerimiskeelt [2]. Kuigi *Node.js* sobib paljude rakenduste jaoks, ei sobi ta keeruliste arvutusülesannete tegemiseks nagu *Fibonacci* jada arvutamiseks. Mahuka arvutusülesande teostamisel suunatakse kogu protsessori saadaolev töömaht seda ülesannet täitma ja alles pärast selle ülesande lahendamist tegeletakse järjekorras olevate päringutega. Seetõttu tekib rakenduses viivitus ning just selle põhjuse tõttu ei soovitata *Node.js'i* kasutada raskete arvutusülesannete jaoks [3].

Node.js'i kogukond on aktiivne ja pidevalt kasvav. Tänu sellele on võimalik valida paljude teekide vahel, mida on võimalik installida *NPM* (*Node Package Manager*) abil. Nende teekide kasutamine on soovitatav, sest see vähendab ridade arvu koodis ja võimalikke tekkivaid vigu. Kuigi paljud teegid on head, siis leidub ka selliseid, mida enam ei uuendata ja nende kasutamisel tekivad koodis vead.

Node.js valiti, sest sama programmeerimiskeele kasutamine nii eessüsteemis, kui ka tagasüsteemis kiirendab arendust vähendades uue keele õppimiseks kuluvat aega. Selles projektis kasutatakse *Node.js* teeke serveri, kui ka eessüsteemi arendamiseks. Lisaks sellele on *Node.js* dokumentatsioon hästi kirjutatud.

1.2 PostgreSQL

PostgreSQL on võimas avatud lähtekoodiga relatsiooniline andmebaasisüsteem, mis kasutab *SQL* programmeerimiskeelt ja on sellele lisanud veel palju lisavõimalusi. *PostgreSQL*'i ajalugu ulatub aastasse 1986, kui seda hakati arendama *POSTGRES* projekti raames California ülikoolis Berkeleys. Aktiivne arendus on kestnud enam kui 30 aastat. Lisaks sellele on *PostgreSQL* teeninud tugeva maine enda tõestatud arhitektuuri, usaldatavuse, andmetervikluse, laialdaste võimaluste ja laiendatavuse tõttu. *PostgreSQL* töötab kõikidel tuntumatel operatsioonisüsteemidel - *Linux*, *macOS*, *Windows*, *Solaris*, *BSD* [4].

Sellesse projekti valiti *PostgreSQL* andmebaas tema populaarsuse, tasuta kättesaadavuse ning *TypeORM'i* ja *GraphQL*'i toe tõttu.

1.3 TypeORM

TypeORM on objekt-relatsioonvastendus raamistik, mis kasutab töötamiseks *Node.js* ning on kirjutatud *TypeScript'is*. Objekt-relatsioonvastendus on tööriist, mis vastendab olemid andmebaasi tabelitega ja suudab luua tabelite vastavad omavahelised suhted. *TypeORM* lihtsustab arendusprotsessi automatiseerides olemist tabelisse ja tabelist olemisse toimuvaid teisendusi. Lisaks sellele on *TypeORM'iga* võimalik luua andmebaasi tabelid vastavalt defineeritud olemitele ning teha operatsioone nende olemite objektidega nagu sisestada, uuendada ja kustutada objekte andmebaasis [5].

TypeORM'i objektidega tehtavad operatsioonid kiirendavad ja lihtsustavad arendusprotsessi ning vähendavad koodi. *TypeORM* on kirjutatud *TypeScript'is,* mis tähendab, et taaskord ei pea õppima uut programmeerimiskeelt ja kogu arendus on tüübitud. Objekti muutmisega muudab *TypeORM* vastavalt ka andmebaasi tabelit ning sellepärast on *TypeORM* selles projektis.

1.4 React

React on avatud lähtekoodiga *JavaScript'i* teek, mis on mõeldud eessüsteemi arendamiseks ja selle abil on võimalik ehitada keerulisi kasutajaliideseid. *React'is* on mugav luua taaskasutatavaid komponente. Komponentide taaskasutamine tagab koodi mitte kordamise ning kogu komponendi loogika on ühes kohas, seetõttu ei pea tegema muudatusi mitmes kohas. Lisaks sellele vähendab see vigade arvu koodis ja kiirendab arendustööd. *React'is* andmete põhiliseks liigutamiseks kasutatakse parameetreid ja olekut. Parameetreid saavad komponendid edasi anda nende alamkomponentidele. Kui olek muutub, laetakse uuesti ainult

see komponent, milles muutus toimus ning arendajatel pole vaja oleku muutumisega kaasnevaid muudatusi hallata [6].

React'i kasutatakse projektis tema võimekuse tõttu laadida uuesti ainult komponente ja vaateid, mis muutuvad ilma, et arendaja peaks kõike kontrollima. Komponentide loomisega hoitakse kood korras ning on võimalik liigutada andmeid mugavalt ühest kohast teise. *React'is* on võimalik kasutada ka *TypeScript'i*, mis on soovituslik ning on ka projektis kasutatud.

1.5 TypeScript

TypeScript on avatud lähtekoodiga keel, mis täiustab ja tüübib *JavaScript'i*. See on üks maailma populaarseim tööriist tüüpide lisamiseks ja defineerimiseks. Tüüpide kasutamine on täiesti valikuline, aga tüüpide defineerimine aitab ennetada vigu enne koodi käivitamist. Tüübi vead on *JavaScript'i* kasutavates projektides kõige sagedasemad. *TypeScript'i* kasutamine projektis vähendab silumise jaoks vajaminevat aega ning muudab koodi turvalisemaks ja lühemaks [7].

TypeScript'i kasutatakse antud projektis, et oleks selgelt näha millist tüüpi parameetreid funktsioonidesse on vaja sisse anda ja millist tüüpi andmeid funktsioon tagastab. *TypeScript'i* kasutamisega välditakse andmete liigutamisega seotud segadusi ja vigu.

1.6 GraphQL Nexus

GraphQL Nexus on deklaratiivne, "*code-first*"põhimõttel ja tugevalt tüübitud *GraphQL* skeemi konstruktsioon. *GraphQL Nexus* on osa *Nexuse* raamistikust, aga seda saab kasutatada ka eraldi seisva osana. Kui "*schema-first*"põhimõtte puhul kirjutatakse skeem enne ja pärast resolver (tegevus mis tehakse päringu või mutatsiooni välja kutsumisel), siis *GraphQL Nexus'*es kirjutatakse päringu või mutatsiooni definitsioon ja resolver ühte kohta ning mõlemad on samas programmeerimiskeeles *JavaScript*/*TypeScript* [8]. See võimaldab vähendada vigu ja muuta koodi loetavamaks. Eriti oluline on see suurte projektide puhul. *GraphQL Nexus* loodi mõeldes *JavaScripti*/*TypeScripti* peale, mis kombineerib *TypeScript'*i tööpõhimõtte, tingimuslikud tüübid ja tüüpide mestimised, et automaatselt luua tüüpide katvus.

GraphQL Nexus't kasutatakse projektis tema võimekuse poolest genereerida päringutest ja mutatsioonidest skeemi fail, muutes arenduse protsessi kiiremaks ja lihtsamaks. Lisaks sellele on tal *TypeScript'i* tugi, mis tüübib skeemi ning see tagab sobivuse juba projektis olemasolevate tehnoloogiatega.

1.7 GraphQL Code Generator

GraphQL Code Generator on tööriist, mis võimaldab *GraphQL* skeemist genereerida *TypeScript'i* tüübid (vt Joonis 1.1), et eessüsteem teaks, mis tüüpi andmeid päringud ja mutatsioonid tagastavad. Selline tüübi teadlikkus kiirendab ja lihtsustab arendust, sest nii oskavad programmeerimiskeskkonnad soovitada arendajale objektis olevaid andmeid, mida tavaliselt ei osata soovitada [9].

GraphQL Code Generator'it kasutatakse selles projekti, et eessüsteem teaks tagasüsteemist tulevate andmete tüüpe ning, et neid poleks vaja ise kirjutada. Samuti on antud tööriistal võimekus genereerida tüübid vastavate päringutega ja mutatsioonidega seotud funktsioonide jaoks.

2 Olemasolevate tõlkerakenduste/tõlketeekide analüüs

Selles peatükis analüüsitakse kolme erinevat tõlkerakendust/tõlketeeki. Nende leidmiseks on kasutatud veebilehte *G2.com*. *G2* on veebileht, kust firmad saavad leida, hinnata ja hallata tehnoloogiaid, et muuta enda töö kiiremaks ja mugavamaks [10]. Analüüsitavad tõlkerakendused/tõlketeegid on valitud selle järgi, mis tundusid idee ja tööpõhimõtte järgi olevat kõige sarnasemad praegusele arendatavale tõlkerakendusele ning olid populaarsemate hulgas [11].

2.1 i18next

i18next on internatsionaliseerimis raamistik, mis on kirjutatud *JavaScript'is*. Seda on mugav integreerida eessüsteemi raamistikke nagu *React*, *AngularJS*, *Vue.js* ja veel paljudesse teistesse [12][13]. Järgnevalt kirjeldatud integratsioon on tehtud *React* raamistiku, sest seda raamistiku kasutatakse ka arendatavas tõlkerakenduses. Teiste raamistike näited on nähtaval dokumentatsioonis ja on põhimõttelt samad. *i18next* raamistiku plussideks on tõlkevõtmetel põhinev tõlkimine (*t('Welcome to React')* on antud juhul määratud tõlkevõti *i18next'is* millele antakse *translation.json* failis vastav väärtus, vt Joonis 2.1) ja nende kokku koondamine *i18next'i* poolt genereeritud tõlkefaili (vt Joonis 2.2). Antud raamistik on tasuta kättesaadav. *i18next'i* miinusteks on graafilise kasutajaliidese puudumine. Graafilise kasutajaliidese puudumine tähendab, et tõlkijate kutsumine projekti ei ole võimalik, sest *i18next* on teek, mis lisatakse projekti ja projektile tõlkijatel ligipääsu tavaliselt pole. Selline lahendus muudab tekstide tõlkimise keerukamaks, sest nüüd tuleb vastavate tõlgete saamiseks tõlkijale saata vastava keele tõlkefail (antud juhul translations.json) ja pärast seda saadab tõlkija tõlgitud faili tagasi. Lõpuks peab arendaja tõlkijalt saadud faili lisama käsitsi tagasi projekti. Mida suuremaks keelte arv projektis kasvab, seda kiiremini muutub tõlkimisprotsess tüütuks ja aeganõudvaks.

```
1  import React from 'react';
2
3  // the hook
4  import { useTranslation } from 'react-i18next';
5
6  function MyComponent () {
7    const { t, i18n } = useTranslation();
8    return <h1>{t('Welcome to React')}</h1>
9  }
```

Joonis 2.1 *i18next* tõlkevõti "Welcome to React" [14].

i18next ülesehitus hakkab vastavate teekide alla laadimisest (vt Joonis 2.3). Järgnevalt tuleb teek konfigureerida, tehes *index.js* faili kõrvale *i18n.js* faili (vt Joonis 2.4) ning pärast seda importida *i18n.js* fail *index.js* faili (vt Joonis 2.5). Järgmiseks tuleb luua *translation.json* fail, kus seatakse tõlkevõtmed vastavusse nende tõlkega (vt Joonis 2.2). Edasi on arendajal valik, kas jätkata tõlkevõtmete lisamist *translation.json* faili käsitsi või lisada juurde *i18next-scanner* teek pärast mille konfigureerimist käiakse läbi kood ja lisatakse leitud tõlkevõtmed automaatselt translations.json faili [15]. Viimase teegi lisamine on vägagi soovitatud. Pärast seda on integratsiooni protsess lõppenud.

```
{
  "Welcome to React": "Welcome to React and react-i18next"
}
```

Joonis 2.2 *translations.json* fail [14].

```
npm install react-i18next i18next --save
```

Joonis 2.3 *i18next* teekide allalaadimine [14].

```
import i18n from "i18next";
import { reactI18nextModule } from "react-i18next";

import translationEN from '../public/locales/en/translation.json';

// the translations
const resources = {
  en: {
    translation: translationEN
  }
};

i18n
  .use(reactI18nextModule) // passes i18n down to react-i18next
  .init({
    resources,
    lng: "en",

    keySeparator: false, // we do not use keys in form messages.welcome

    interpolation: {
      escapeValue: false // react already safes from xss
    }
  });

export default i18n;
```

Joonis 2.4 *i18n.js* konfiguratsiooni fail [14].

```
import React, { Component } from "react";
import ReactDOM from "react-dom";
import './i18n';
import App from './App';

// append app to dom
ReactDOM.render(
  <App />,
  document.getElementById("root")
);
```

Joonis 2.5 *i18.js* faili import *index.js* faili [14].

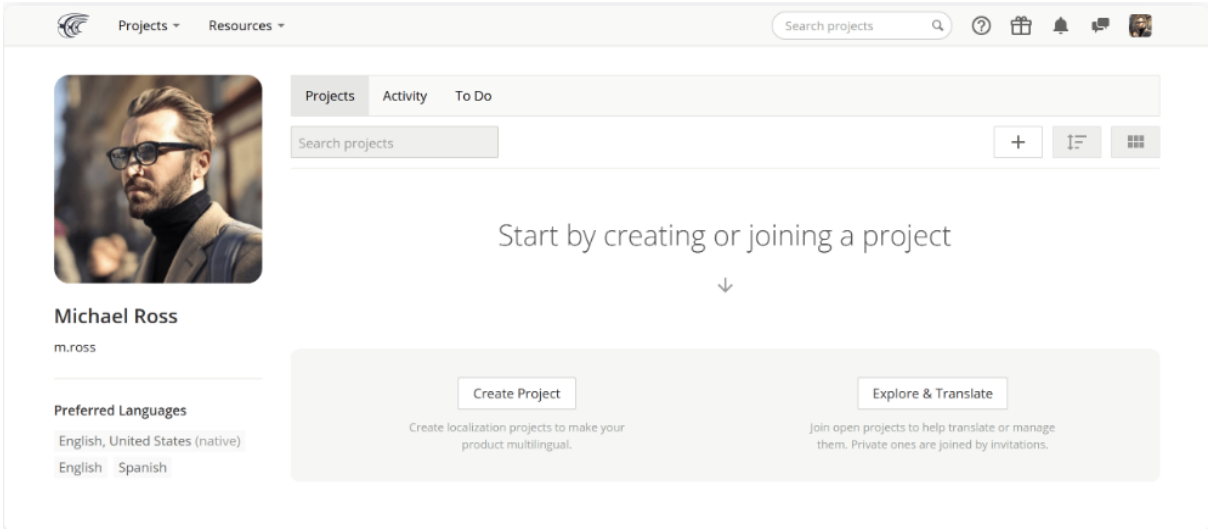
2.2 Crowdin

Crowdin'i pakette on kokku viis ning tasuta prooviperioodid puuduvad (vt Joonis 2.6). Esmapilgul vaadates on näha, et *Crowdin* pakub ka tasuta paketti, täpsemal uurimisel selguvad aga põhjused, miks see on tasuta. Tasuta paketiga on võimalik teha ainult üks privaatne projekt ning ainult avalikku projekti on võimalik kutsuda enda tõlkijaid. Kui projekti avalikkus ja võimalus kõikidel teistel seda otsida ja vaadata pole probleem, siis need väljatoodud punktid kõige suuremad miinused ei ole. Küll tuleb, aga arendajal projektiga tegeleda üksinda, sest *Managers included* ei ole saadaval tasuta paketis ja isegi ka *Pro* paketis, mis tähendab, et kedagi teist sellesse projekti seda haldama ja tõlkevõtmeid lisama ei ole võimalik kutsuda. Seda võib pidada tasuta paketi suurimaks miinuseks, ning on ka üks põhjustest hetkel arendatava projekti tegemiseks. Kui tasuta ja *Pro* pakett välja jätta ja valida kohe *Team* pakett, siis on *Crowdin* hea valik, et muuta enda rakendus mitmekeelseks [16].

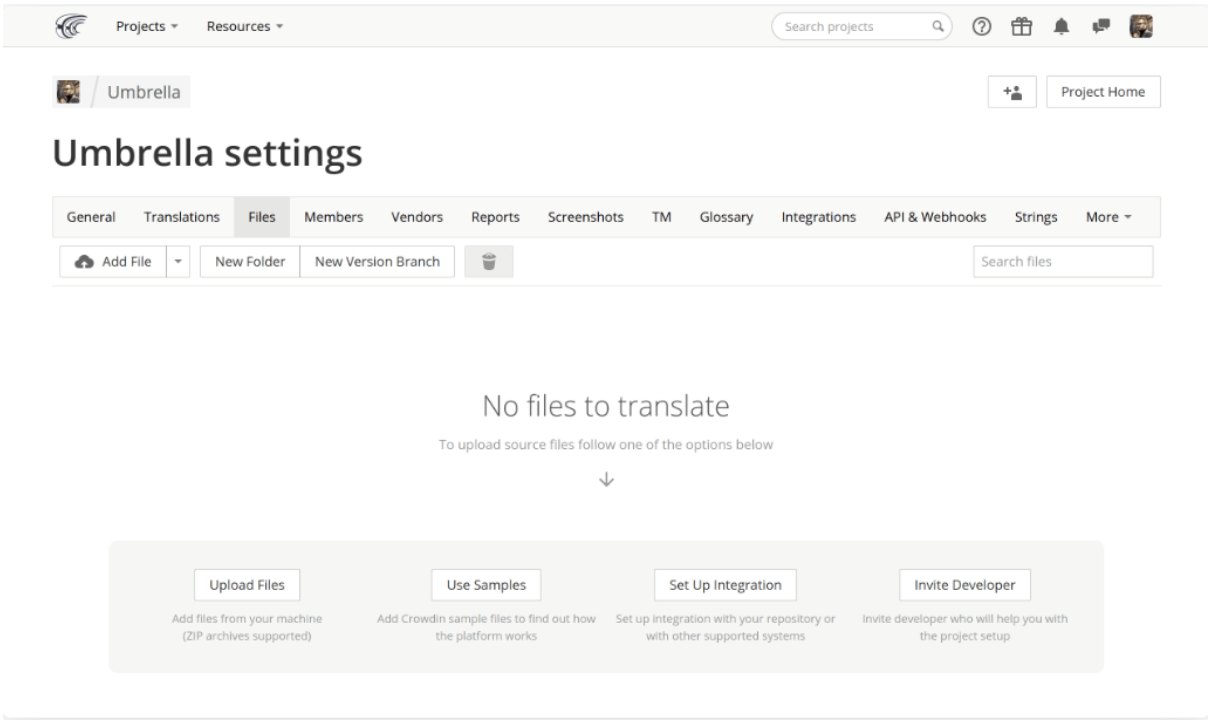
Crowdin'i ülesehitus algab kasutaja loomisega ja sisselogimisega. Seejärel on kasutajal võimalik teha projekt (vt Joonis 2.7). Projektis tuleb määrata keeled, millesse tahetakse tõlkida ning tuleb määrata ka lähtekeel. Erinevalt arendatava tõlkerakendusega ei toimu *Crowdin'is* tõlkevõtmete ja tõlgete lisamist. Ette tuleb anda tõlgete fail, mis peaks olema määratud lähtekeeles (vt Joonis 2.8) ning seetõttu on *Crowdin'i* hea kasutada koos *i18next'iga*, mis genereerib ise tõlkefaili. Faili formaadi tuvastab *Crowdin* ise. Etteantud faili põhjal loob *Crowdin* ise vastavate valitud keelte jaoks failid, kus tõlkijad saavad hakata olemasolevat teksti tõlkima ja arendajad vastavad failid alla laadida ja projekti lisada (vt Joonis 2.9).

	Free	Pro	Team	Team+	Business
	$0 / mo	from $40 / mo	from $140 / mo	from $450 / mo	from $1500 / mo
THE BASICS	CALCULATE ↓	CALCULATE ↓	CALCULATE ↓	CALCULATE ↓	CALCULATE ↓
Hosted words	Unlimited	Starting at 60 000	Starting at 150 000	Starting at 500 000	Starting at 2 500 000
Translators	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited
Public projects	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited
Private projects	1	3	Unlimited	Unlimited	Unlimited
Managers included	×	×	5	10	40

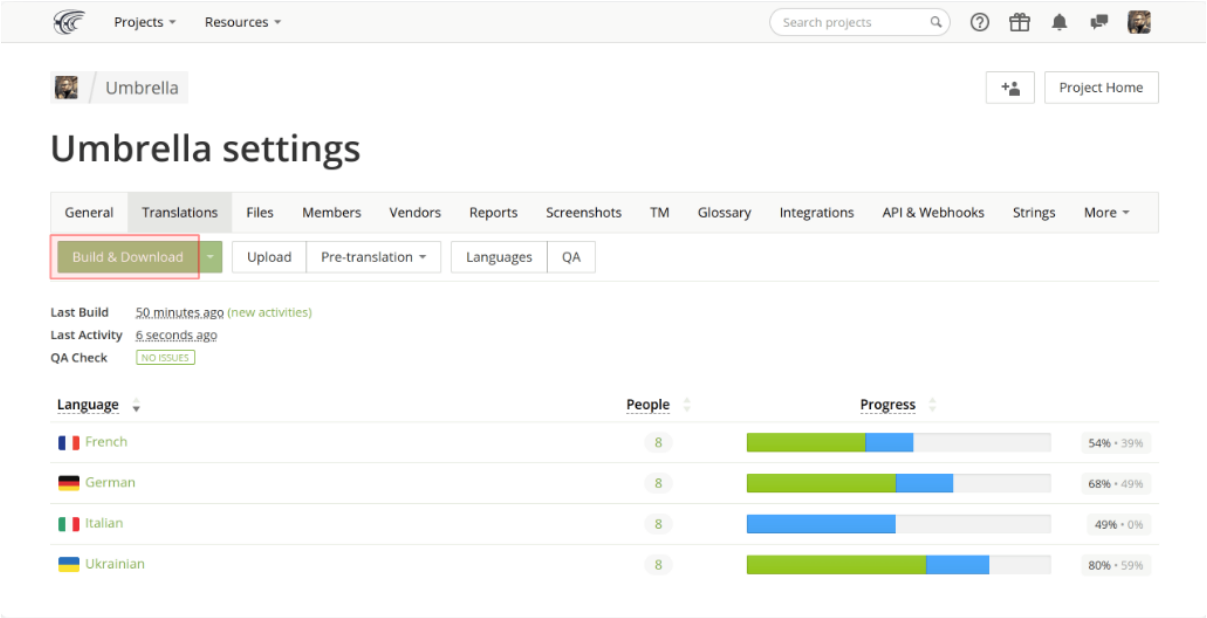
Joonis 2.6 *Crowdin'i* hinnakiri [17].



Joonis 2.7 *Crowdin'i* projekti loomine [18].



Joonis 2.8 *Crowdin'i* faili üleslaadimise vaade [19].



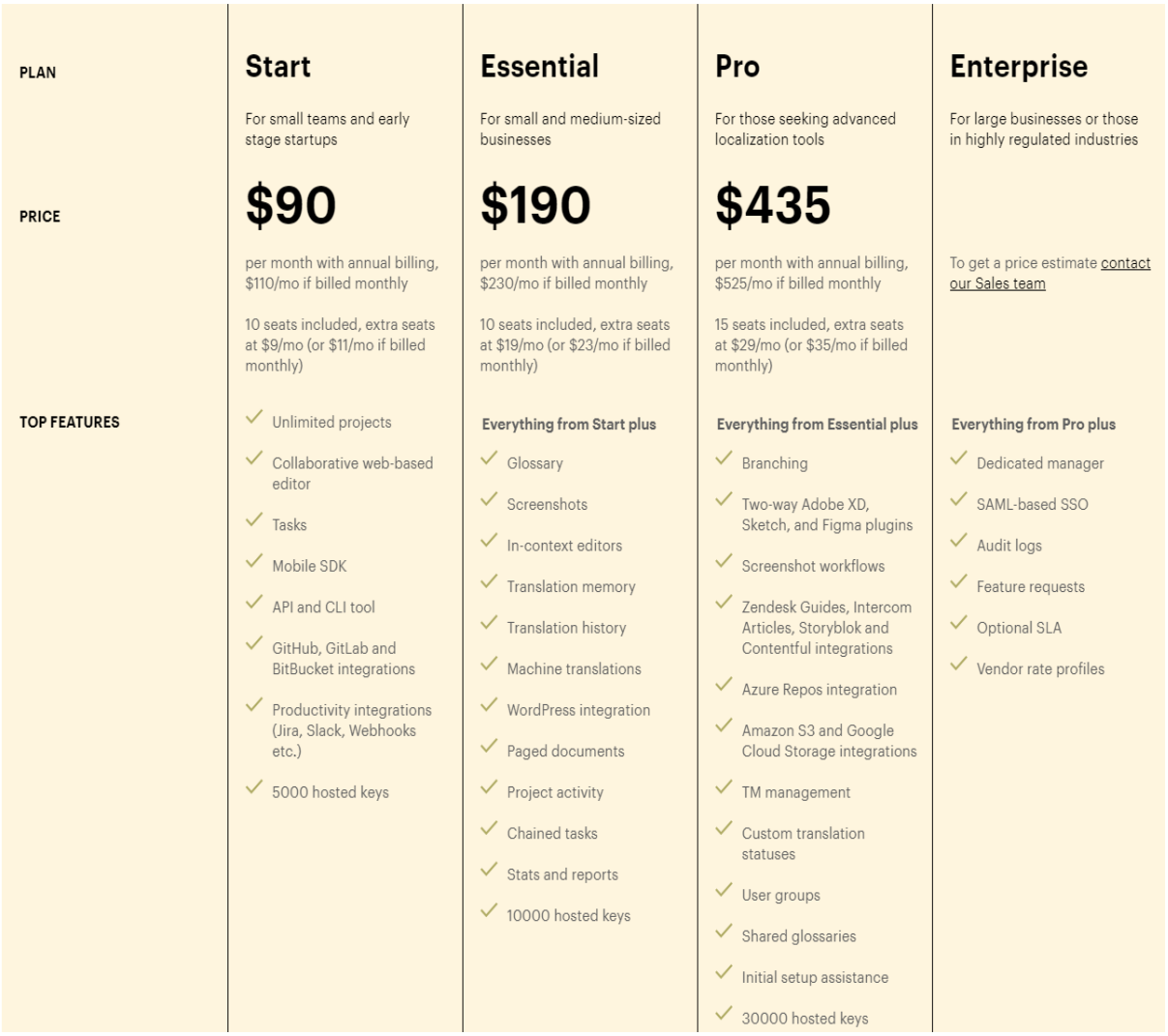
Joonis 2.9 *Crowdin'i* tõlgete allalaadimise vaade [20].

2.3 Lokalise

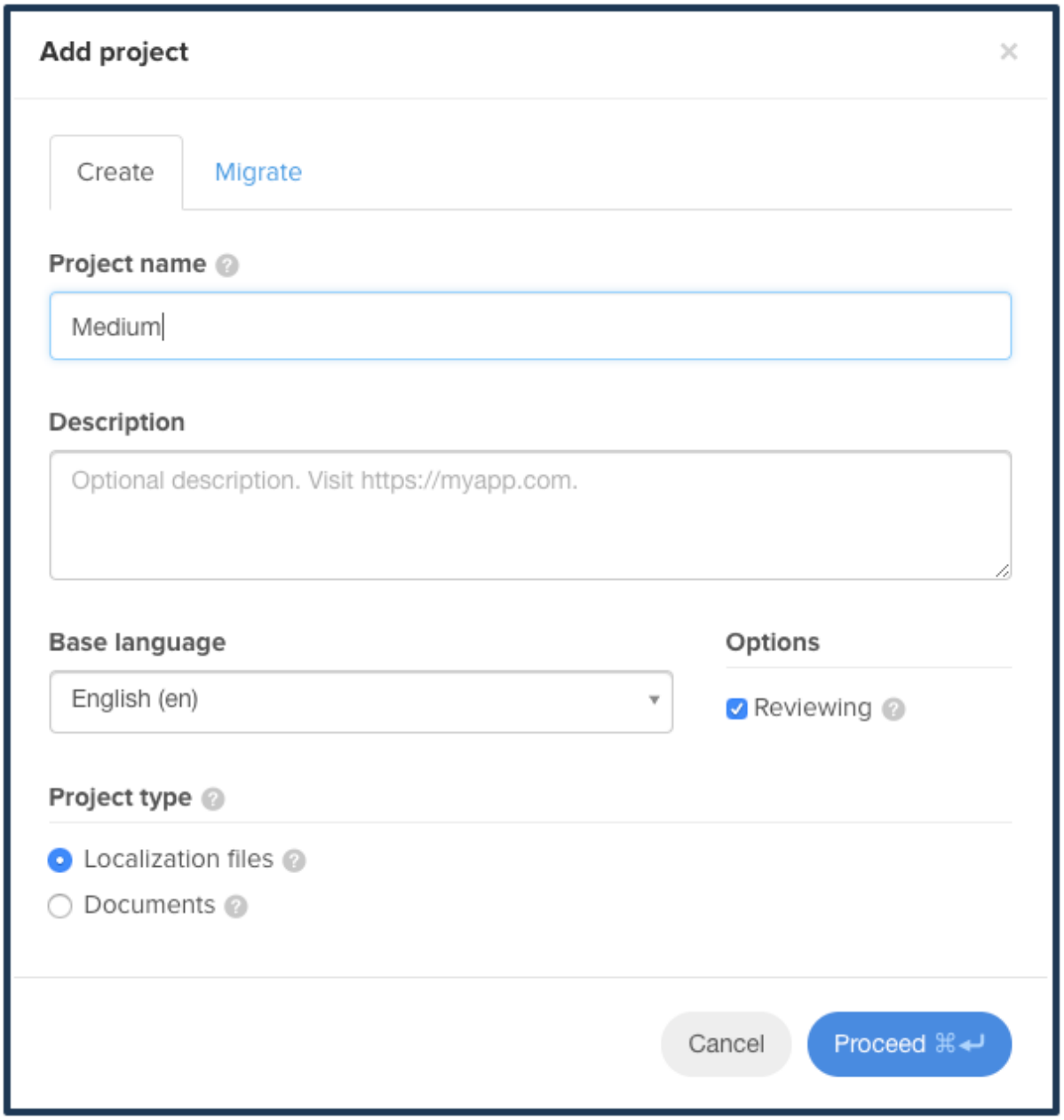
Lokalise'il on võimalik valida nelja paketi vahel. Iga paketi tasuta prooviaeg on 14 päeva, pärast mida tuleb hakata maksma hinda vastavalt valitud paketile (vt Joonis 2.10). Vaadates hinna tabelit, on selgelt näha, et väiksema projekti jaoks on *90$* kuus maksta päris suur summa, eriti kui rakendus võib olla mõned üksikud vaated. Hind on *Lokalise'i* suurim miinus. *Lokalise'i* plussid on tema intuitiivne kasutajaliides ja võimalus tõlkevõtmetele juurde lisada silte, millisesse valdkonda võti kuulub. Lisaks sellele saab tõlkevõtmetele veel juurde lisada pilte kindlast vaatest, et tõlkijatel ja arendajatel oleks hea ülevaade toimuvast. Projekti omanik saab kutsuda arendajaid ja tõlkijaid vastavate projektidega liituma. See tähendab, et võtmete lisamine ja teksti tõlkimine toimub *Lokalise'i* keskkonnas ehk tavaliselt toimuv tõlkefailide saatmine jääb ära.

Lokalise on oma ülesehituselt kõige sarnasem hetkel arendatava tõlkerakendusega[21]. Kui kasutaja on tehtud ja sisse logitud, siis on võimalik luua projekt (vt Joonis 2.11). Pärast sisselogimist on võimalik lisada keeli ja tõlkevõtmeid (vt Joonis 2.12, Joonis 2.13). Keelte ja tõlkevõtmete lisamise tulemusena tekib tabel, kuhu on võimalik lisada vastavaid tõlkeid (vt Joonis 2.14). *Lokalise* kasutab *REST API't* ehk tõlgete saamiseks enda mitmekeelsust vajavasse projekti tuleb teha päring *Lokalise'i API* pihta, millele antakse kaasa

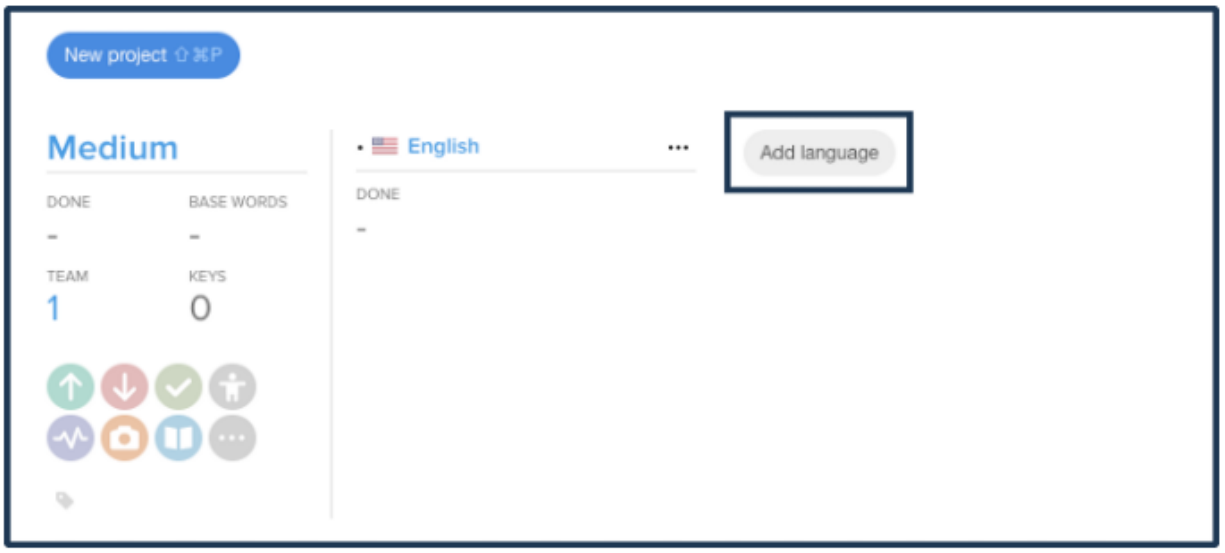
vastava projekti *API* võti. Sellel protsessi lihtsustamiseks on Lokalise teinud teegid *php*, *Ruby*, *Node.js'i*, *Golang'i* ja *Python'i* jaoks [22]. Kõik andmed tagastatakse *JSON* kujul pärast, mida saab arendaja neid enda vajadustele kasutama hakata.



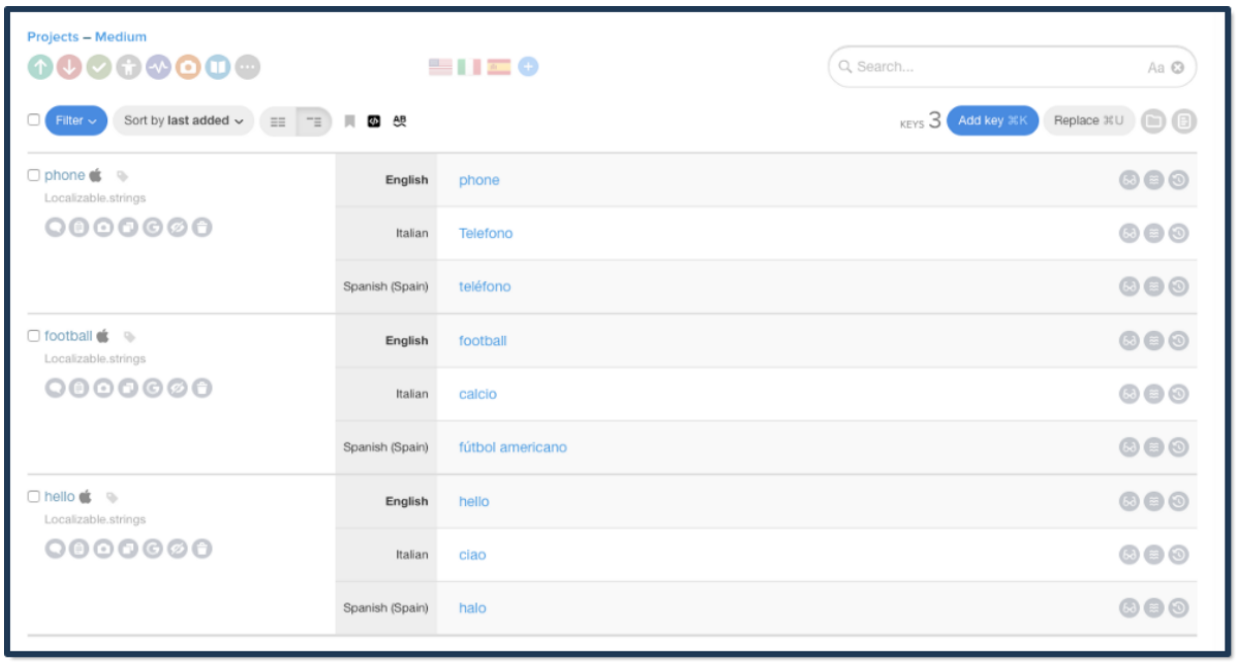
Joonis 2.10 *Lokalise'i* hinnakiri [23].



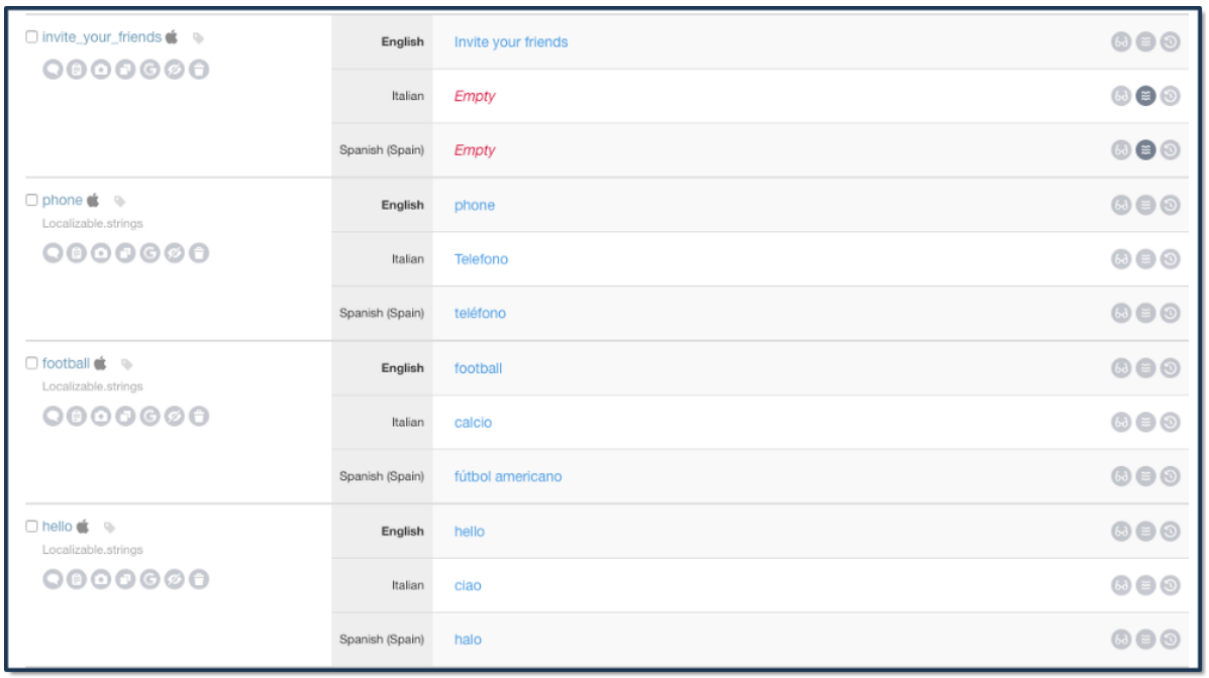
Joonis 2.11 *Lokalise'i* projekti lisamine [21].



Joonis 2.12 *Lokalise'i* keele lisamine [21].



Joonis 2.13 *Lokalise'i* tõlkevõtme lisamine [21].



Joonis 2.14 *Lokalise'is* tõlgete muutmine [21].

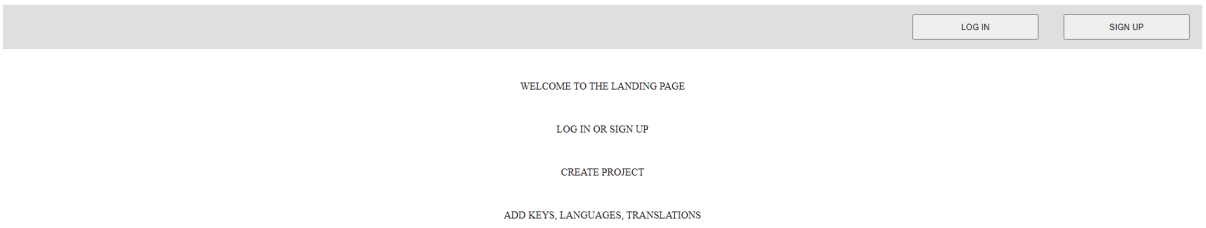
3 Veebirakenduse ülesehitus

Veebirakendus on ülesehitatud kolmeks osaks, mis kokku moodustavad töötava terviku - eessüsteem, tagasüsteem ja andmebaas (vt Lisa Ⅰ). Selles peatükis kirjeldatakse nende osade tööd.

3.1 Eessüsteem

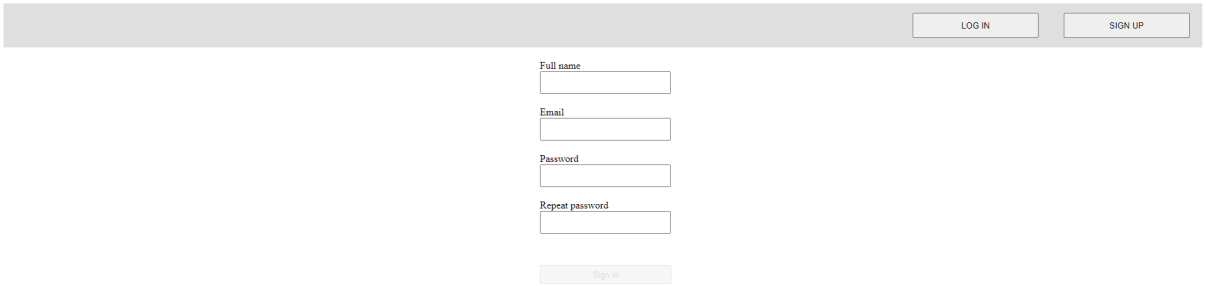
Eessüsteemi pooles on kasutatud *Node.js*, *Reacti*, *TypeScripti* ja *GraphQL*'i.

Kasutajale nähtavad rakenduse osad on jaotatud vaadeteks ning nende voog algab kasutaja saabumisest avalehele (vt Joonis 3.1). Avalehel on kirjas täiendav info antud veebirakenduse ja selle kasutamise kohta.



Joonis 3.1 Rakenduse avaleht.

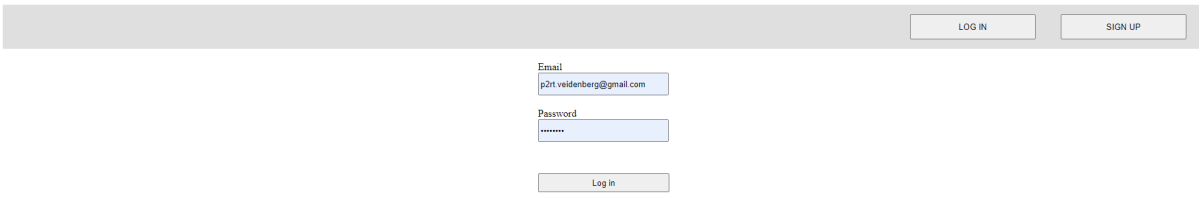
Avalehe päisel on kaks nuppu - "*Log In*" ja "*Sign Up*". Vajutades nuppu "*Sign Up*" avaneb uue kasutaja tegemise vaade. Selles vaates kasutaja tegemiseks on ära vaja täita nelja väljaga vorm - nimi, email, salasõna, salasõna kordus ja vajutada vormis olevat nuppu "*Sign up*" (vt Joonis 3.2).



Joonis 3.2 Kasutaja tegemise vaade.

Kui konto loomine õnnestub logitakse kasutaja sisse ja suunatakse projektide vaatesse (vt Joonis 3.4).

Vajutades avalehe päises nuppu "*Log In*" avaneb sisselogimise vaade (vt Joonis 3.3), mille vormis tuleb ära täita emaili ja salasõna väli ning vajutada nuppu "*Log In*". Eduka sisselogimise puhul suunatakse kasutaja projektide vaatesse (vt Joonis 3.4).

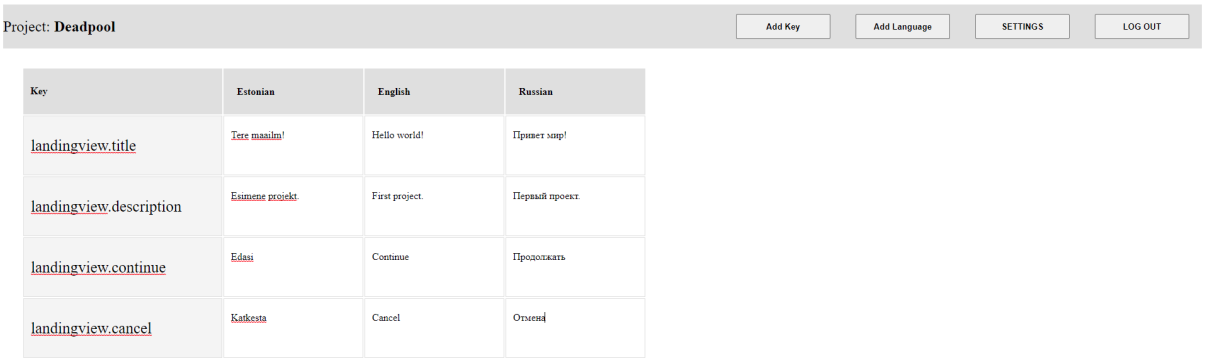


Joonis 3.3 Kasutaja sisselogimise vaade.

Joonis 3.4 Projektide vaade.

Projektide vaate päises on võimalik seanss (*session*) katkestada, vajutades nuppu "*Log out*".

Projektide vaate kehas on loend antud kasutajale võimalikest projektidest. Projekti ümbrisele (*container*) vajutades suunatakse kasutaja vastava projekti vaatesse (vt Joonis 3.5).

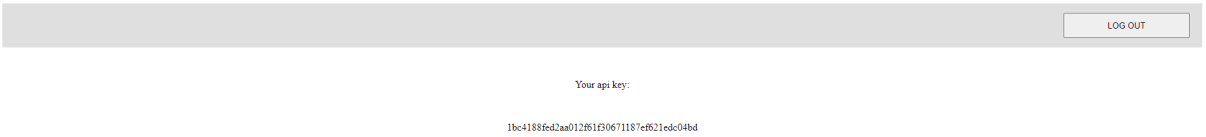


Joonis 3.5 Projekti vaade.

Projekti vaates näidatakse kasutajale tabelit, mille ridade esimesteks elementideks on tõlkevõtmed ja veergude esimesteks elementideks on keeled. Tabeli sisu iga väärtus vastab kindlale keelele ja tõlkevõtmele. Igale tabeli lahtrile on võimalik klikkida, et muuta tema väärtust.

Projekti vaate päises on nupud "*Add key*", "*Add language*", "*Settings*", "*Log out*".

“*Add key*” lisab tabelisse juurde uue rea. “*Add language*” lisab tabelisse uue veeru. “*Settings*” nupp suunab kasutaja seadete vaatesse (vt Joonis 3.6) ja “*Log out*” lõpetab kasutaja seansi.



Joonis 3.6 Seadete vaade.

Seadete vaate päises on nupp “*Log out*”, mis lõpetab kasutaja seansi ja suunab ta avalehele.

Töö kirjutamise hetkel on seadete vaate kehas nähtav ainult *API* võti, et tehtud tõlkeid oleks võimalik pärida antud tõlkerakendust kasutavasse projekti.

3.2 Tagasüsteem

Tagasüsteemi saab jaotada kolmeks suuremaks osaks - olemiteks, rakendusliideseks ja serveriks. Olemid on *TypeORM'i* elemendid, millega defineeritakse andmebaasi tabeli objektid, nendes sisalduvad väljad ja relatsioonid (vt Joonis 3.7)[24]. Antud joonisel on toodud näitena olem “Projekt”, millel on enda unikaalne universaalne identifikaator ja projekti nimi. Antud olemis on veel defineeritud neli relatsiooni, mis jaotuvad oma olemuselt kaheks - “*OneToMany*” ja “*ManyToMany*”. “*OneToMany*” tähendab, et olem A sisaldab ainult ühte olemi B isendit ja “*ManyToMany*” tähendab, et olem A sisaldab mitut olemi B isendit ning olem B sisaldab endas mitut olemi A isendit [25].

```
import { BaseEntity, Column, Entity, OneToMany, ManyToMany, PrimaryGeneratedColumn } from "typeorm";
import { LanguageEntity } from "./LanguageEntity";
import { TagEntity } from "./TagEntity";
import { TranslationkeyEntity } from "./TranslationkeyEntity";
import { UserEntity } from "./UserEntity";

@Entity("project")
export class ProjectEntity extends BaseEntity {
  @PrimaryGeneratedColumn("uuid")
  readonly id!: string;

  @Column({ type: "text" })
  projectName!: string;

  @Column({ type: "text" })
  apiKey!: string

  @OneToMany(() => LanguageEntity, (language) => language.project)
  languages: LanguageEntity[];

  @OneToMany(() => TagEntity, (tag) => tag.project)
  tags: TagEntity[];

  @OneToMany(() => TranslationkeyEntity, (translationkey) => translationkey.project)
  translationkeys: TranslationkeyEntity[];

  @ManyToMany(() => UserEntity, (user) => user.projects)
  users: UserEntity[];
```

Joonis 3.7 Projekti olem.

Kui kõik olemid ja nende vahelised relatsioonid on valmis, jooksutati migreerimise jaoks tehtud skripti, mis olemite põhjal defineeritud objektidest genereerib andmebaasi tabelid.

Rakendusliidese tegemiseks on kasutatud *GraphQL Nexus't*, mida kirjutatakse *JavaScriptis*/*TypeScriptis*. Kasutame näitena projekti lisamise mutatsiooni (vt Joonis 3.8). *GraphQL Nexusus'ega* defineerime ära "*addProject*" nimelise mutatsiooni objekti, mille tagastus tüübiks on "*Project*". "*Project*" on spetsiaalselt kohandatud tüüp (vt Joonis 3.9) millel on väljad "*id*", "*projectName*", "*apiKey*" ja "*translations*". Mutatsioon saab sisse ühe argumendi - "*projectName*", mis on sõne tüüpi. Väljal "authorize" kontrollitakse, kas kasutajal on üldse õigust antud mutatsiooni teha. Kõige viimaseks väljaks on resolver kus defineeritakse, mida mutatsioon peaks tegema ja tagastama. Projekti lisamine on kõigest üks operatsioon. Vajalik on veel projekti muuta, kustutada ja kõiki projekte pärida ning kõikide nende kohta on vaja teha eraldi vastavalt mutatsioonid ja päringud. Kõikidest päringutest ja mutatsioonidest genereerib *Nexus* ise skeemi faili ja tüübib selle ära (vt Joonis 3.10), mida kasutatakse hiljem eessüsteemis.

Serveri tegemiseks on kasutatud "*apollo-server-express*" teeki (vt Joonis 3.11) [26]. Serveri käivitamisel defineeritakse seansi vahetarkvara (*session middleware*) ja küpsise parserit (*cookie parser*).

```
import { mutationField, stringArg } from "@nexus/schema";
import { UnauthorizedError } from "../../../validate";
import { ProjectEntity } from "../../entities/ProjectEntity";

export default mutationField("addProject", {
  type: "Project",
  args: {
    projectName: stringArg(),
  },
  description: "Adds new project",
  authorize: (_parent, _args, context) => context.isLoggedIn(),
  resolve: async (_parent, args, { viewer }) => {
    if (!viewer) {
      throw new UnauthorizedError();
    }

    // check if project name already exists
    if (viewer.projects.some((project) => project.projectName === args.projectName)) {
      throw new Error("Project name already exists");
    }

    const project = ProjectEntity.create({
      projectName: args.projectName,
    });

    viewer.projects.push(project);
    await viewer.save();

    return project;
  },
});
```

Joonis 3.8 Mutatsioon *addProject*.

```ts
import { objectType } from "@nexus/schema";
import { In } from "typeorm";
import { TranslationEntity } from "../entities/TranslationEntity";
import { TranslationkeyEntity } from "../entities/TranslationkeyEntity";
import { getRootTypingImport } from "../services/getRootTypingImport";

export default objectType({
  name: "Project",
  rootTyping: getRootTypingImport("ProjectEntity"),
  definition(t) {
    t.id("id");
    t.string("projectName");
    t.string("apiKey");
    t.list.field("translations", {
      type: "Translation",
      resolve: async (project, _args, _context) => {
        const translationKeyIdEntities = await TranslationkeyEntity.find({
          where: {
            projectId: project.id,
          },
        });

        const translationKeyIds = translationKeyIdEntities.map((translationKeyIdEntity) => translationKeyIdEntity.id);

        return TranslationEntity.find({
          where: {
            translationkeyId: In(translationKeyIds),
          },
        });
      },
    });
  },
});
```

Joonis 3.9 Defineeritud tüüp *Project.*

```
### This file was generated by Nexus Schema
### Do not make changes to this file directly

type Language {
  id: ID!
  languageName: String!
}

type Mutation {
  """
  Adds new language
  """
  addLanguage(languageName: String!, projectId: String!): Language!

  """
  Adds new project
  """
  addProject(projectName: String!): Project!

  """
  Adds new translation
  """
  addTranslation(languageId: String!, translationkeyId: String!, translationValue: String!): Translation!

  """
  Adds new translationkey
  """
  addTranslationkey(projectId: String!, translationKeyName: String!): Translationkey!

  """
  Attempts to log user in
  """
  login(
    """
    Email address
    """
    email: String!

    """
```

Joonis 3.10 Osa *Nexus Schema* genereeritud skeemi failist.

```
// setup graphql server
const apolloServer = setupGraphqlServer({
  path: graphqlPath,
  schema,
  playgroundEnabled: config.graphql.playground,
  introspection: config.graphql.introspection,
  debug: config.graphql.debug,
});

// get the express application
const app = express();

// trust the proxy
app.enable("trust proxy");

// apply urlencoded body parser
app.use(bodyParser.urlencoded({ extended: false, limit: "50MB" }));
app.use(bodyParser.json());

// setup cookies support used by sessions
app.use(cookieParser());

// setup session support (used by authentication etc)
app.use(sessionMiddleware);

// initialize session middleware
app.use((request, _response, next) => {
  // should not happen
  if (!request.session) {
    throw new Error("Session support is not properly configured");
  }

  next();
});
```

Joonis 3.11 "*apollo-server-express" teegiga loodud server.*

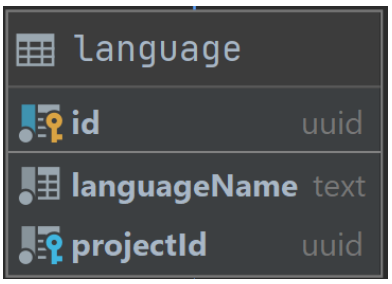
3.3 Andmebaas

Selles projektis on kasutatud *PostgreSQL* andmebaasi. See koosneb seitsmest põhitabelist ja mõnest väiksemast abi tabelist, mis ühendavad põhitabeleid. Põhitabeliteks on

"*language*", "*project*", "tag", "*translation*", "*translationkey*", "*user*", "*session*".

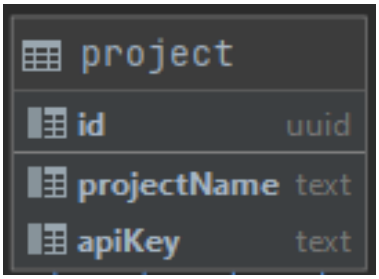
Kõikide tabelite esimeseks väljaks on tema tabelile vastav universaalne unikaalne identifikaator (*uuid*) ja kõikide tabelite väljade tüübid on nähtaval piltidel.

"*language*" - tabel keelte jaoks, mille väljadeks on keele nimi ja projekti identifikaator (*id*), millesse keel kuulub.



Joonis 3.12 Keele tabel.

"*project*"- tabel projekti jaoks, mille väljadeks on projekti nimi ja *API* võti.

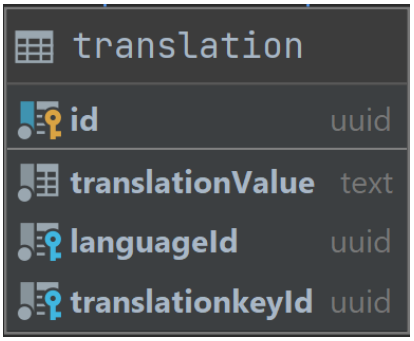


Joonis 3.13 Projekti tabel.

"*tag*" - tabel sildi jaoks, mille väljadeks on sildi nimi ja projekti identifikaator, millesse silt kuulub.

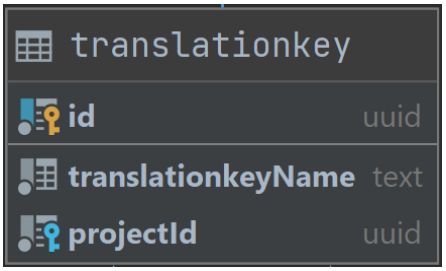
Joonis 3.14 Sildi tabel.

"*translation*" - tabel tõlke jaoks, mille väljadeks on tõlke väärtus ning keele- ja tõlkevõtme identifikaator, millega tõlke väärtus on seotud.



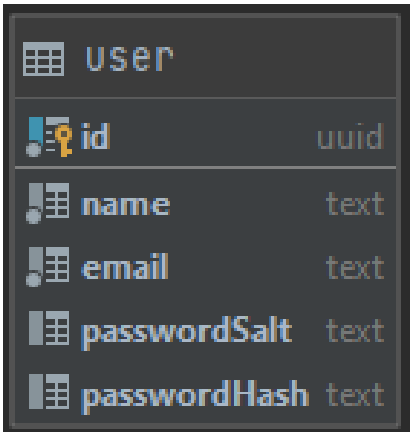
Joonis 3.15 Tõlke tabel.

"*translationkey*" - tabel tõlkevõtme jaoks, mille väljadeks on tõlkevõtme nimi ja projekti identifikaator, millesse tõlkevõti kuulub.



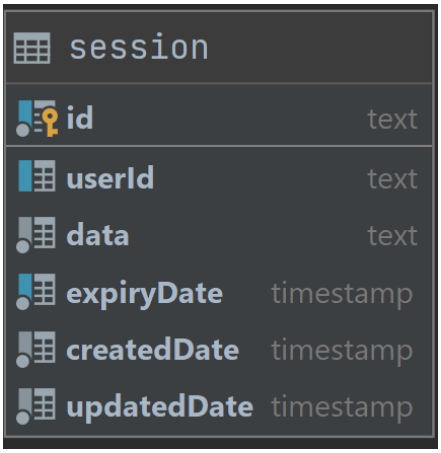
Joonis 3.16 Tõlkevõtme tabel.

"*user*" - table kasutaja jaoks, mille väljadeks on nimi, e-mail, salasõna sool ja salasõna räsi.



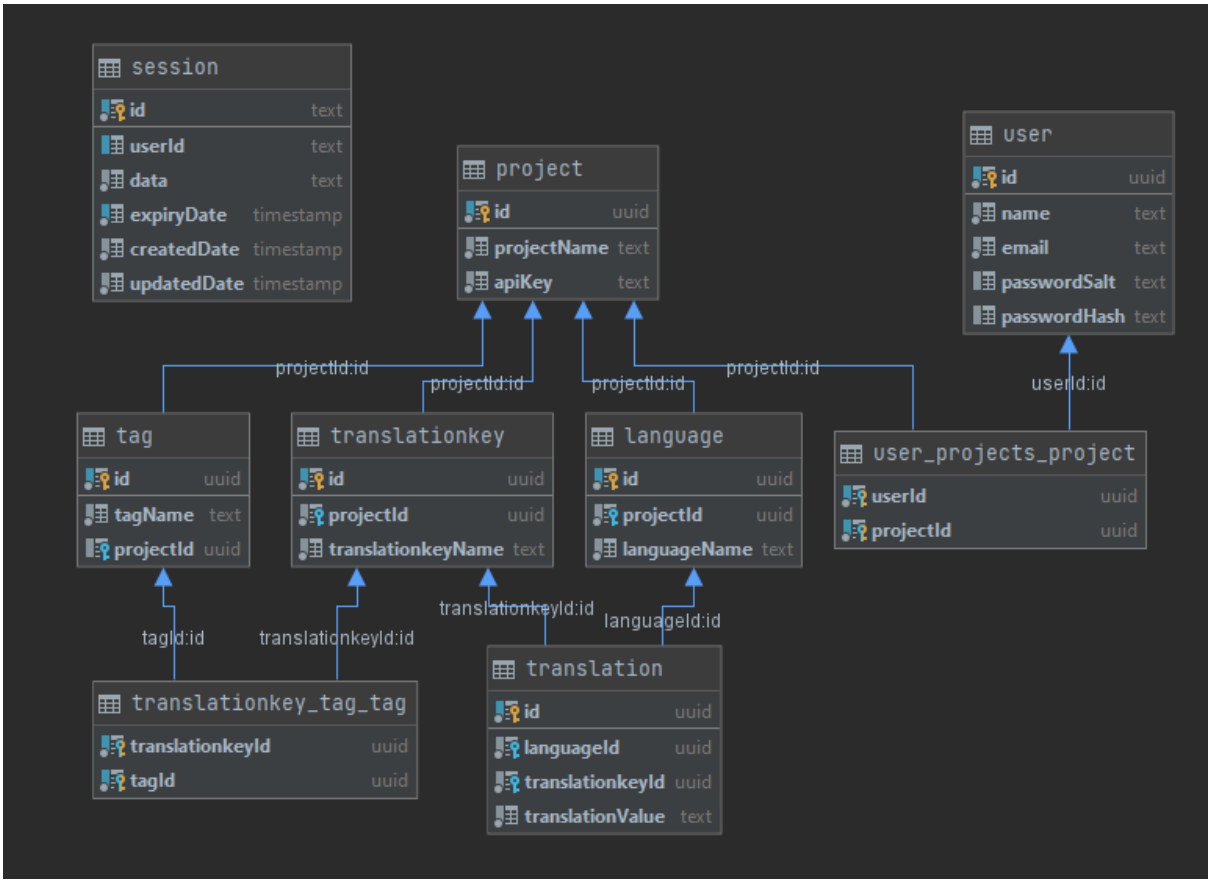
Joonis 3.17 Kasutaja tabel.

"*session*" - tabel seansi jaoks, mille väljadeks on kasutaja identifikaator, lisaandmed, aegumise kuupäev, loomise kuupäev ja muutmise kuupäev.



Joonis 3.18 Seansi tabel.

Kogu andmebaas ja tema relatsioonid -



Joonis 3.19 Andmemudel.

Edasiarendus

Lõputöö raames loodud rakendust on võimalik arendada edasi. Töö kirjutamise hetkel pole veel rakendus avalikult kõigile kättesaadav, seega on see üks edasi arendamise punkt. Lisaks sellele on andmebaasi lisatud sildid, mida eessüsteemi veel lisatud pole. Samuti saab paremaks teha tekkinud vigade kuvamist kasutajale, et oleks selgelt aru saada, mis läks valesti. Seadete vaatesse tuleb lisada veel uue salasõna tegemine, kasutajatel rollide andmine ja kasutajate kutsumine projektidesse. Veel peaks uurima tõlketabeli koormustaluvust ja võimalusel optimeerima päringuid või andmevorme.

Kokkuvõte

Käesoleva lõputöö eesmärgiks oli luua tõlkerakendus mitmekeelsuse haldamiseks ja analüüsida sarnaseid olemasolevaid rakendusi. Rakendus on suunatud arendajatele, kellel on vaja enda veebi- või mobiilirakendus muuta tasuta mitmekeelseks. Analüüsi käigus selgusid põhjused uue rakenduse loomiseks. Ühtlasi on ka välja toodud rakenduse loomiseks kasutatud tehnoloogiad ja kirjeldatud kliendi, serveri ja andmebaasi osade tööpõhimõtet. Lõputöö eesmärk täideti tõlgete haldamise veebirakenduse loomisega ning sellele järgneb edasine arendus.

Viidatud kirjandus

[1] *What Is Localization, And When Do You Need It?* Scott Ludwigsen, 2018.

Saadaval: https://blog.languageline.com/what-is-localization. [Kasutatud 2. mai 2021]

[2] *Node.js Definition*, James Denman, 201.

Saadaval:

https://whatis.techtarget.com/definition/Nodejs. [Kasutatud 2. mai 2021]

[3] *When You Should and Shouldn't Use Node.js for Your Project*, Juan Cruz Martinez, 2020. Saadaval: https://livecodestream.dev/post/when-you-should-and-should-not-use-nodejs-for-your-project [Kasutatud 2. mai 2021]

[4] *What is PostgreSQL?*

Saadaval: https://www.postgresql.org/about/. [Kasutatud 3. mai 2021]

[5] *TypeORM - Introduction.*

Saadaval:

https://www.tutorialspoint.com/typeorm/typeorm_quick_guide.htm. [Kasutatud 3. mai 2021]

[6] *What And Why React.js*, Nitin Pandit, 2021.

Saadaval:

https://www.c-sharpcorner.com/article/what-and-why-reactjs/. [Kasutatud 4. mai 2021]

[7] *Why You Should Choose TypeScript Over JavaScript*, Gints Dreimanis, Olga Bolgurtseva, 2020. Saadaval: https://serokell.io/blog/why-typescript#what-is-typescript%3F. [Kasutatud 3. mai 2021]

[8] *Introducing GraphQL Nexus*, Tim Griesser, 2019.

Saadaval: https://www.prisma.io/blog/introducing-graphql-nexus-code-first-graphql-server-development-ll6s1yy5cxl5. [Kasutatud 4. mai 2021]

[9] *What is GraphQL Code Generator?*

Saadaval: https://www.graphql-code-generator.com/docs/getting-started/index. [Kasutatud 4. mai 2021]

[10] *Information about G2.com.*

Saadaval: https://culture.g2.com/about. [Kasutatud 4. mai 2021]

[11] *The Top 5 Software Localization Tools*.

Saadaval: Highest Rated Software Localization Tools | G2. [Kasutatud 2. mai 2021]

[12] *i18next introduction*.

Saadaval: https://www.i18next.com/overview/introduction. [Kasutatud 4. mai 2021]

[13] *i18next supported frameworks*.

Saadaval: https://www.i18next.com/overview/supported-frameworks. [Kasutatud 4. mai 2021]

[14] *react-i18next Quick start.*

Saadaval: https://react.i18next.com/guides/quick-start. [Kasutatud 5. mai 2021]

[15] *i18next-scanner teek*.

Saadaval: http://i18next.github.io/i18next-scanner/. [Kasutatud 3. mai 2021]

[16] *Crowdin Introduction*.

Saadaval: https://support.crowdin.com/crowdin-intro/. [Kasutatud 3. mai 2021]

[17] *Crowdin pricing.*

https://crowdin.com/pricing#annual. [Kasutatud 5. mai 2021]

[18] *Create Crowdin project.*

https://support.crowdin.com/creating-project/. [Kasutatud 5. mai 2021]

[19] *Crowdin* tõlkefaili üleslaadimine

https://support.crowdin.com/uploading-files/. [Kasutatud 5. mai 2021]

[20] *Crowdin* tõlkefaili allalaadimine

https://support.crowdin.com/downloading-translations/. [Kasutatud 5. mai 2021]

[21] *Lokalise flow*.

Saadaval: https://medium.com/@ryanisnhp/save-your-time-with-lokalise-a-translation-platform-eac3d2f740cc. [Kasutatud 3. mai 2021]

[22] *Lokalise teegid*.

Saadaval: https://app.lokalise.com/api2docs/curl/. [Kasutatud 3. mai 2021]

[23] *Lokalise pricing*.

Saadaval: https://lokalise.com/pricing. [Kasutatud 3. mai 2021]

[24] *TypeORM Entity*.

Saadaval: https://www.tutorialspoint.com/typeorm/typeorm_entity.htm. [Kasutatud 3. mai 2021]

[25] *TypeORM relations*.

Saadaval: https://www.tutorialspoint.com/typeorm/typeorm_relations.htm. [Kasutatud 3. mai 2021]

[26] *Apollo* server.

Saadaval: https://www.npmjs.com/package/apollo-server-express. [Kasutatud 3. mai 2021]

Lisad

I . Programmi lähtekood

"Tõlkerakendus mitmekeelsuse haldamiseks" lähtekoodi repositoorium – viimati uuendatud 27.04.2021

https://github.com/PVeidenberg/translationkeys/tree/origin/development

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Pärt Veidenberg**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   Tõlkerakendus mitmekeelsuse haldamiseks,

   mille juhendajad on Vambola Leping ja Reiko Randoja,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

   1. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

   2. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

   3. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Pärt Veidenberg

16.04.2021