APLIKACIJA ZA UPRAVLJANJE PRIJEVODIMA UNUTAR KODA DRUGIH APLIKACIJA

Peklarić, Kris

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:884142

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-27**

ALGEBRA SVEUČILIŠTE

Repository / Repozitorij:

Algebra Univerity - Repository of Algebra Univerity

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Aplikacija za upravljanje prijevodima unutar koda drugih aplikacija

Kris Peklarić

Zagreb, Veljača 2023.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, Veljača 2023.

Predgovor

Želim se zahvaliti svome mentoru Bojanu Fulanoviću na pomoći i strpljenju tijekom cijelog procesa uključenog u izradi završnog rada, svim profesorima na Visokom učilištu Algebra na prenesenom znanju i iskustvima koji su me naučili inženjerski pristupati rješavanju problema te time olakšali daljnji put.

Posebno se želim zahvaliti svojoj obitelji i prijateljima na podršci tijekom studiranja i života.

A Gradišćanska 24, HR-10000 Zagreb
T (01) 2222 182
E student@algebra.hr
www.algebra.hr

ALGEBRA

Temeljem članka 8. Pravilnika o završnom radu i završnom ispitu na preddiplomskom studiju Visokog učilišta Algebra sačinjena je ova

Potvrda o dodjeli završnog rada

kojom se potvrđuje da student Kris Peklarić, JMBAG 0203011897, OIB 91504477321 u šk. godini 2021./2022., studij: Primjenjeno računarstvo - Preddiplomski studij, smjer: Programsko inženjerstvo, od strane povjerenstva za provedbu završnog ispita, dana 23.02.2022. godine, ima odobrenu izradu završnog rada

s temom: **Aplikacija za upravljanje prijevodima unutar klijentskih biblioteka za izradu sučelja web aplikacija**

i sažetkom rada: Ideja ovog rada je izrada desktop aplikacije koja se bavi upravljanjem prijevodima unutar koda. Aplikacija učitava određeni projekt u memoriju tako da prolazi kroz sve datoteke te pronalazi rezervirana mjesta za prijevode. Korisnik može preko sučelja aplikacije dodavati nove prijevode ili mijenjati postojeće. Svaki projekt sadrži postavke projekta poput svih jezika koje projekt podržava. Ne prevedene riječi moguće je automatski prevesti koristeći Google Translate API.

Mentor je: Bojan Fulanović.

Odobrenjem završnog rada studentu je omogućen upis kolegija "Izrada završnog projekta/Praksa" te je sukladno članku 8. Pravilnika o završnom radu i završnom ispitu dužan najkasnije do početka nastave ljetnog semestra u sljedećoj školskoj godini, uspješno obraniti završni rad uspješnim polaganjem završnog ispita.

U protivnom student može zatražiti novog mentora/icu i temu te ponovo upisati kolegij "Izrada završnog projekta/Praksa" budući da rad koji nije predan i obranjen na završnom ispitu u roku određenom Pravilnikom završnom radu i završnom ispitu prestaje vrijediti. Izrada novog završnog rada se izvodi sukladno rokovima određenima za školsku godinu u kojoj je studentu određen novi mentor/ica i dodijeljen novi završni rad.

Potpis studenta:	Potpis mentora:	Potpis predsjednika povjerenstva:

Ova potvrda izdaje se u 4 (četiri) primjerka od kojih 3 (tri) idu kao prilog završnom radu.

Sažetak

Ideja ovog rada je izrada desktop aplikacije koja se bavi upravljanjem prijevodima unutar koda kako bi olakšala svakodnevne situacije sa kojima se programeri susreću te im uštedjela vrijeme koje se troši na taj posao. Aplikacija učitava određeni projekt u memoriju tako da prolazi kroz sve datoteke te pronalazi rezervirana mjesta za prijevode. Nakon učitavanja prijevoda korisnik može preko sučelja aplikacije dodavati nove prijevode ili mijenjati postojeće. Svaki projekt sadrži postavke projekta poput svih jezika koje projekt podržava kako bi aplikacija znala raditi sa projektom. Kako bi korisnik mogao brzo pristupiti projektima te mijenjati postavke nadodan je ekran koji se bavi isključivo upravljanjem projektima. Radi brzog dodavanja novih prijevoda zbog podrške za što više jezika, dodana je podrška za Google Translate API pomoću kojega korisnik može automatski prevoditi nepoznate riječi.

Ključne riječi: Electron, React.js, Google Translate API, desktop aplikacija, upravljanje prijevodima.

Summary

The idea for this work is to a create desktop application that will manage with translations inside code so it can make it easier for developers in everyday situations and it saves them time that they would usually spend on managing translations. The application loads projects into memory by going through all files and searching for translations on them. After all translations were loaded the user can easily manage it through application's interface. Every project contains settings, so application knows which languages it supports. For quick access and project management, there is an interface specially intended for it. Also, there is support for Google Translate API so users can quickly make new translations into languages that they currently don’t support.

Key words: Electron, React.js, Google Translate API, desktop application, translation management.

Sadržaj

1. Uvod 3

2. Vrste jezičnih i kulturoloških podrški 4

3. Arhitektura aplikacije 5

4. Razvoj aplikacije 12

4.1. Struktura projekta 12

4.1.1. Opis 12

4.1.2. Upravljanje projektima 14

4.1.3. Uređivač prijevoda 17

4.2. Korištene tehnologije 21

4.2.1. JavaScript 21

4.2.2. TypeScript 21

4.2.3. Node.js 22

4.2.4. Electron 23

4.2.5. React.js 24

4.2.6. HTML 25

4.2.7. CSS/SCSS 26

4.2.8. PrimeReact 27

4.2.9. Redux 27

4.2.10. React Hook Form 28

4.2.11. Create React App 28

4.2.12. Vue.js 28

4.3. Dodavanje korisnikovog projekta 29

4.4. Učitavanje projekta u memoriju 37

4.4.1. Prikaz stabla datoteka i mapa 38
4.4.2. Prikaz odabrane datoteke u uređivaču prijevoda 44
4.5. Prikaz svih upotreba za određeni identifikator 48
4.6. Upravljanje prijevodima 50
4.7. Spremanje svih promjena 52
4.8. Prevođenje pomoću Google Translate API 55
4.9. Izvoz u CSV formatu 60
Zaključak 62
Popis kratica 63
Popis slika 64
Popis kodova 65
Literatura 67

1. Uvod

Današnje web aplikacije koje koriste moderne JavaScript programske okvire imaju načine na koje rješavaju problem prijevoda. Nažalost održavanje prijevoda programerima oduzima puno vremena koje bi moglo biti usmjereno na druge stvari, čime bi samo zadovoljstvo programera naraslo. Također postoji ogroman broj web aplikacija koje nisu lokalizirane, a postoji potreba za lokalizacijom i dodavanjem novih prijevoda kako bi se mogle plasirati na nova tržišta.

Kod trenutnog unosa prijevoda unutar koda programer je obavezan raditi provjere postojanja identifikatora vezanih za te prijevode i ovisno po potrebi ih dodavati ili ažurirati. Time nepotrebno troši vrijeme koje bi moglo biti bolje iskorišteno za puno važnije stvari poput većeg fokusa na samom kodu.

Cilj ovoga rada je izrada aplikacije koja će omogućiti pojednostavljenje načina upravljanja prijevodima i smanjiti broj mogućih grešaka u kodu vezanih za prijevode. Fokus rada aplikacije će biti isključivo na podržavanju internacionalizacije (skraćeno *i18n*) usmjerene prema modernim JavaScript programskim okvirima. S obzirom da internacionalizacija obuhvaća mnogo toga (ostavljanje prostora unutar korisničkih sučelja za lakše prevođenje u jezike koji zahtijevaju više znakova, razvoj korištenjem alata koji podržavaju internacionalne znakove i mnoge druge), aplikacija će se isključivo fokusirati na prijevode. Korisnici će imati mogućnost postojeće prijevode automatski prevoditi u sve preostale jezike koje su zadali za taj projekt. Prijevodi će se moći izvesti u CSV formatu kako bi se olakšala suradnja sa prevoditeljima. Aplikacija je izrađena Electron tehnologijom i fokusirati će se na podršku za Vue.js programski okvir pomoću kojeg će se kreirati testni projekt. Korisničko sučelje koristit će React.js tehnologiju.

2. Vrste jezičnih i kulturoloških podrški

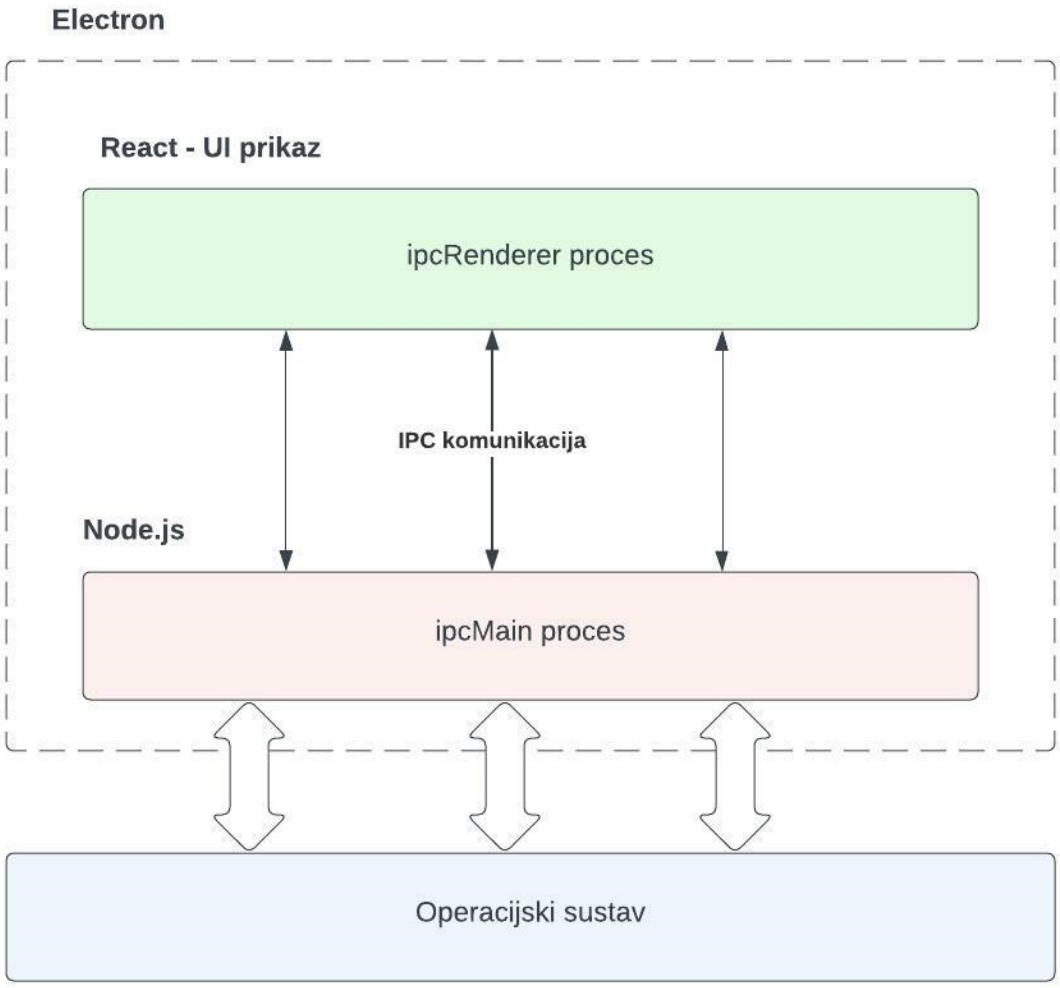
Postoje različite države, a sa njima i različiti jezici. Neke države govore istim jezikom, ali su određene riječi, način izražavanja i valuta drugačiji. Na osnovu toga možemo reći da je kultura tih država različita. Za lakši način razvoja programa (engl. software) koji je prilagođen različitim kulturama i jezicima osmišljen je proces koji se zove Internacionalizacija (engl. *internationalization*).

Internacionalizacija je osmišljavanje i razvoj proizvoda, aplikacija ili dokumenata koji omogućavaju lakšu lokalizaciju za ciljanu skupinu koja se razlikuje u kulturi, regiji ili jeziku (https://www.w3.org/International/questions/qa-i18n). Za internacionalizaciju se još koristi kratica *i18n* koja je nastala od prvog i zadnjeg slova u riječi *internationalization* te broja 18. koji označava broj slova između prvog i zadnjeg slova. Za razliku od internacionalizacije, lokalizacija (engl. *localization*) je proces prilagodbe internacionaliziranog programa za specifični jezik ili regiju. Ona se bavi prijevodom tekstova te korištenjem specifično lokalnih komponenti kao što je na primjer prikaz valuta u različitim lokalizacijama. Kratica za lokalizaciju je *L10n* te je također nastala na isti način od prvog i zadnjeg slova te broja 10 koji označava broj slova između od riječi *localization*. Pojam globalizacija se koristi unutar određenih kompanija za korištenje internacionalizacije i lokalizacije.

U ovom radu problem kojim se aplikacija bavi je fokusiran na lokalizaciju i prijevode. Kako bi aplikacija radila sa prijevodima, zamišljeno je da se prijevodi za svaku lokalizaciju nalaze u posebnoj datoteci koja je u JSON formatu. Svaki ključ unutar datoteke predstavlja identifikator za taj prijevod te treba biti isti u različitim datotekama da bi se odnosio na isti prijevod. U slučajevima kada ključ kao vrijednost sadrži objekt sa novim prijevodima, aplikacija to također podržava tako da dodjeljuje prefiks sa ključem identifikatora unutar kojega se nalazi te točkom nakon koje dolazi ključ tog prijevoda. Za nove dublje razine aplikacija se ponaša na isti način tako da nadodaje prijašnje vrijednosti ključeva sve do najviše razine. U slučajevima kada nadodamo prijevod za zadani jezik dok svi ostali nedostaju te se ne nalaze u lokalizacijskim datotekama za druge jezike, aplikacija omogućuje automatsko prevođenje kako bi olakšala dodavanje prijevoda koji nedostaju. Ime datoteke treba sadržavati prva dva znaka koja predstavljaju određenu kulturu za koju su ti prijevodi namijenjeni. Fokus aplikacije je na lokalizaciji i upravljanju prijevodima.

3. Arhitektura aplikacije

Aplikacija je zamišljena kao desktop aplikacija koja se može koristiti bez internetske konekcije. Za dodatnu značajku automatskog prevođenja je ipak potrebna internetska konekcija. Rađena je koristeći *Electron programski okvir* (engl. *framework*) te se sastoji od dva dijela. Prvi dio je sam **Electron** koji koristi *Node.js* tehnologiju te služi za pristup i obradu podataka pohranjenih na računalu korisnika dok je drugi dio **UI sučelj**e koje koristi *React.js* tehnologiju te služi za sam prikaz (Slika 3.1).



Slika 3.1 Prikaz komunikacije između Node.js i React.js dijela u aplikaciji

Electron i *React* međusobno komuniciraju pomoću interprocesne komunikacije (engl. *Inter process communication*, skraćeno IPC). Procesi komuniciraju tako da prosljeđuju poruke

unutar korisnički definiranih kanala. Sastoje se od dva modula u kojem se mogu definirati. Prvi je **ipcMain** koji se definira i koristi unutar **Electron** dijela aplikacije te **ipcRenderer** unutar **React** dijela. Poruke nemaju obavezan smjer u kojem mogu putovati tako da mogu putovati u svim smjerovima [1].

U glavnoj Electron datoteci uvozimo Electron objekte (Kôd 3.1), konfiguraciju prozora koju smo definirali te Node.js pomoćne metode za upravljanje putanjom datoteke koje želimo učitati u aplikaciju

```
import windowConfig from './config/windowConfig';
import { app, BrowserWindow, ipcMain } from 'electron';
import * as path from 'path';
import * as url from 'url';
```

Kôd 3.1 Uvezeni dijelovi korišteni unutar glavne Electron datoteke

Nakon toga definiramo varijablu za glavni prozor na kojoj ćemo kasnije kreirati novi prozor (Kôd 3.2), varijablu za provjeru da li se pokreće aplikacija tijekom razvoja te omogućavamo vanjski pristup Electron metodama za slanje poruka (**ipcMain**) te Electron instanci (**app**).

```
let mainWindow: BrowserWindow | null;
export { ipcMain, app }
const isDevelopment = process.env.NODE_ENV === 'development'
```

Kôd 3.2 Definiranje varijabli unutar glavne Electron datoteke

Zatim definiramo funkciju koja će kreirati novi prozor iz konfiguracije koju smo definirali (Kôd 3.3) te ovisno o modu u kojem je aplikacija pokrenuta učitati datoteku ili postaviti URL.

```
function createWindow() {
      mainWindow = new BrowserWindow({
            ...windowConfig['overview'],
            webPreferences: {
              preload: path.join(__dirname, 'preload.js'),
            },
      });

      if (isDevelopment) {
       mainWindow.loadURL("http://localhost:3000#/overview");
      } else {
```

```
        mainWindow.loadURL(
              url.format({
               pathname: path.join(__dirname, '../index.html'),
               protocol: 'file:',
               slashes: true
              })
           );
        }
}
```

Kôd 3.3 Funkcija za kreiranje prozora

Na kraju kada je *Electron* aplikacija spremna koristimo tu funkciju za kreiranje prozora te se on na kraju prikazuje korisniku (Kôd 3.4). Također dodajemo kod za gašenje aplikacije kada korisnik izađe iz aplikacije ili makne prozor. Na kraju postavljamo sve slušače (engl. *listenere*[1]) koje su na *ipcMain* dijelu aplikacije tako da ih uvezemo korištenjem `require()` metode.

```
app.whenReady().then(async () => {
      await createWindow();
});

app.on('window-all-closed', function () {
      app.quit();
});

require('./handlers/index');
```

Kôd 3.4 Postavljanje prozora kada je aplikacija spremna

Konfiguracija prozora za oba ekrana koja koristimo u aplikaciji (Kôd 3.5):

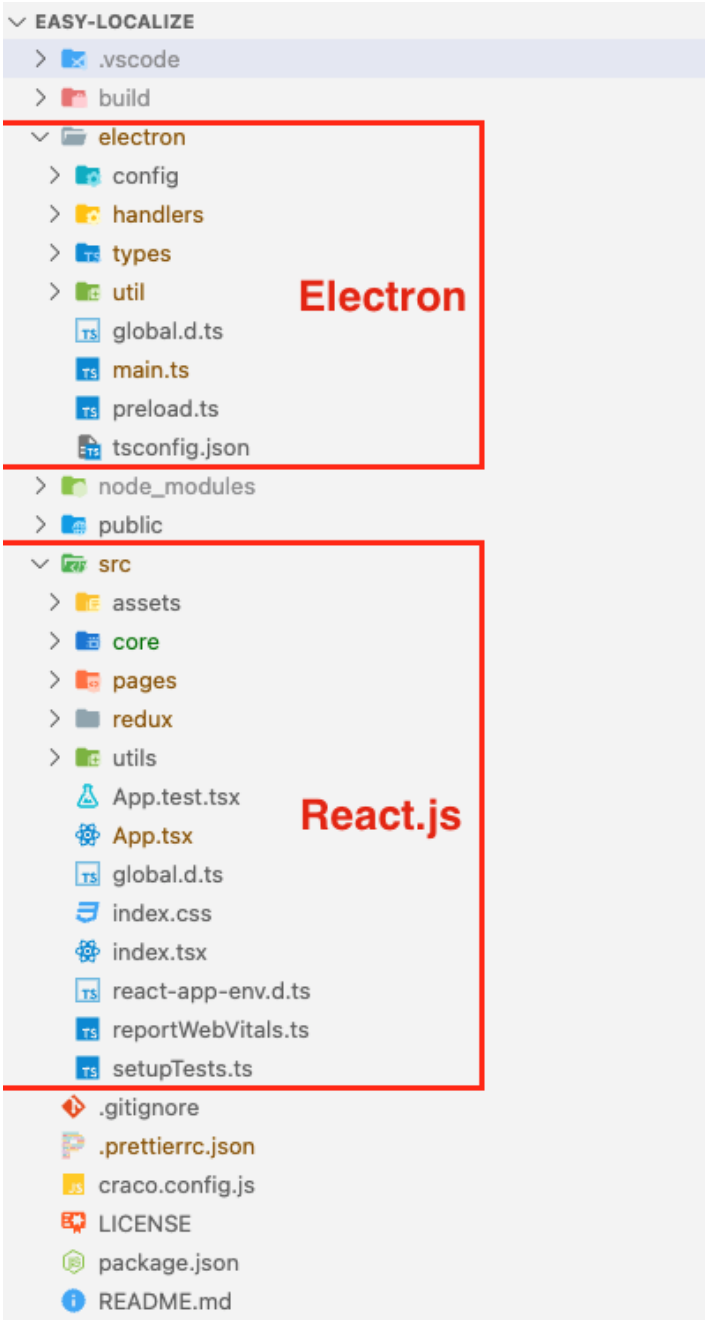
```
const windowConfig: any = {
      overview: {
            title: 'Overview',
            width: 880,
            minWidth: 880,
            maxWidth: 880,
            height: 600,
            minHeight: 600,
```

[1] Listener je metoda koja sluša određene događaje te u ovisnosti izvršava proslijeđene funkcije

```
        maxHeight: 600,
        animation: true,
        center: true,
        maximum: false,
    },
    editor: {
        title: 'Editor',
        width: 1600,
        minWidth: 1280,
        maxWidth: 1980,
        height: 840,
        minHeight: 820,
        maxHeight: 900,
        animation: true,
        maximum: true,
    },}; export default windowConfig;
```

Kôd 3.5 Konfiguracija prozora

Electron dio ima pristup datotekama i mapama korisnika dok **React** dio nema (Slika 3.2). Svaki dio za sebe je izoliran zbog sigurnosti pristupa korisnikovih podataka te je zato potrebna interprocesna komunikacija. Electron je radni okvir za kreiranje desktop aplikacija koristeći web tehnologije kao JavaScript, HTML te CSS. Omogućuje kreiranje *više-*



Slika 3.2 Podjela koda unutar aplikacije

platformskih aplikacija iz jedne baze izvornog koda koja koristi JavaScript te je zato vrlo popularan.

Komunikacija između React.js dijela i Electron dijela se odvija pomoću interprocesne komunikacije. U komunikaciji postoje dva dijela, jedan je glavni proces (**ipcMain**) koji je odgovoran za primanje i obradu *Node.js* dijela te dio koji se nalazi u prikazu i iz kojega se pozivaju poruke. U *window* objektu React.js dijela aplikacije eksponiran (engl. *exposed*) je Electron objekt koji u sebi sadržava metode za primanje i slanje poruka (`send`, `on`, `removeAllListeners`) (Kôd 3.6).

```
import { contextBridge, ipcRenderer } from 'electron';

contextBridge.exposeInMainWorld('electron', {
      send: (channel: string, args: any[]) =>
ipcRenderer.send(channel, args),
      on: (channel: string, callback: (event:
Electron.IpcRendererEvent, ...args: any[]) => void) =>
ipcRenderer.on(channel, callback),
      removeAllListeners: (channel: string) =>
ipcRenderer.removeAllListeners(channel),
});
```

Kôd 3.6 Omogućavanje metoda unutar renderer procesa

Da bi poruke mogli primati u glavnom procesu moramo definirati na što će se poruke pozivati (naziv događaja) i dodijeliti funkcije koje će dalje obrađivati te poruke (engl. *handlers*). To se definira na glavnom procesu koji na sebi ima metodu `on`. Također ukoliko želimo poslati podatke ili poruke natrag u React.js onda možemo koristiti metodu `send()` definiranu na argumentu `sender` [3].

```
import { BrowserWindow, screen } from 'electron';
import { ipcMain } from '../main';
import windowConfig from '../config/windowConfig';

function setWindow({ sender }: Electron.IpcMainEvent, config:
{ window: string, projectId: number }) {
 ...
    sender.send('resize-window-return', {
      projectId: config.projectId
      });
}
```

```
export const windowOpenEditor = ipcMain.on('window-open-
editor', setWindow)
```

Kôd 3.7 Slanje događaja za postavljanje prozora

Poruke se iz prikaza šalju na način da pozovemo na globalno definiranom Electron objektu koji se nalazi u window objektu metodu `send()` (Kôd 3.7). Kao argumente metoda `send()` prima naziv događaja koji je obavezan te kao drugi argument objekt koji želimo poslati koji nije obavezan. Važno je da naziv događaja bude isti.

```
window.electron('naziv-događaja', {items: []});
```

Ako želimo dohvatiti poruku iz Electron dijela onda moramo definirati slušajuću metodu te funkciju koja će se izvršavati nakon što taj event bude pozvan (engl. *callback*[2]) (Kôd 3.8). Slušajuću metodu definiramo unutar *useEffect React hooka* koji se izvršava prilikom montiranja komponente u dokument (engl. *Document Object Model*, skraćeno DOM[3]). Nakon micanja komponente iz dokumenta, slušajuća funkcija (engl. listener) se čisti.

```
useEffect(() => {
      if (window.electron) {
                  window.electron.on('naziv-događaja-
                  pozvanog-iz-main-procesa', (event:
                  any, args: any) => {});
            window.electron.send('settings-load');
      }
      return () => {
window.electron.removeAllListeners(
      'naziv-događaja-pozvanog-iz-main-procesa'
            );
      };
}, []);
```

Kôd 3.8 Primanje događaja unutar React.js dijela

[2] Callback je funkcija koja se izvodi nakon što određeni uvjet bude zadovoljen.
[3] DOM je programsko sučelje za web dokumente koje je stablastog oblika.

4. Razvoj aplikacije

Unutar ovog poglavlja bit će opisana struktura projekta, tehnologije koje su korištenje te značajke koje aplikacija ima te na koji način su ostvarene.

4.1. Struktura projekta

Projekt se sastoji od dva dijela. Prvi je dio za upravljanje projektima dok je drugi dio za uređivanje prijevoda unutar koda drugih aplikacija (engl. *editor*).

4.1.1. Opis

Upravljanje projektima je bilo nužno za napraviti iz razloga što je bilo potrebno jedno mjesto gdje će korisnik moći mijenjati način na koji je projekt konfiguriran te imati brzi pristup prošlim projektima bez potrebe za ponovni unos cijele konfiguracije. Drugi dio projekta se odnosi na uređivač prijevoda u kojem je učitan projekt te korisnik ima opcije upravljanja prijevodima. Svaki dio je odijeljen svojim ekranom te u istom trenutku može biti prikazan samo jedan prikaz. Sam projekt je napravljen kao Electron aplikacija za radnu površinu iz razloga sto se htjelo omogućiti rad na svim okruženjima, bio je potreban pristup mapama i datotekama te je sam razvoj bio puno lakši uz pomoć korištenja postojećih web tehnologija za sam prikaz. Jedan od razloga je i bila zaštita koda korisnika zbog čega nije bilo moguće imati klasičnu web aplikaciju jer bismo time otvorili mogućnost za krađom koda korisnika.

U kodu (Kôd 4.1) možemo vidjeti glavnu React.js datoteku u kojoj se nalaze svi ekrani. Datoteka se sastoji od uvezenih biblioteka i React komponenti te funkcije koja vraća JSX kod koji je odgovoran za prikaz. Sav **JSX**[4] kod se nalazi u `Provider` komponenti od *Redux biblioteke* (engl. *library*) kojoj je proslijeđen uvezeni `store` sa zadanom konfiguracijama te raznim akcijama koje je moguće vršiti nad njima. Time smo omogućili da sve komponente imaju pristup globalnom stanju aplikacije te ga također mogu mijenjati. Unutar `Provider` komponente se nalazi `Router` komponenta koja je odgovorna za odabir komponente koje želimo prikazati za odabranu putanju. Unutar nje se nalazi `Switch` komponenta unutar koje

[4] JSX je ekstenzija JavaScript jezika koja služi za lakše oblikovanje HTML elemenata pomoću koda

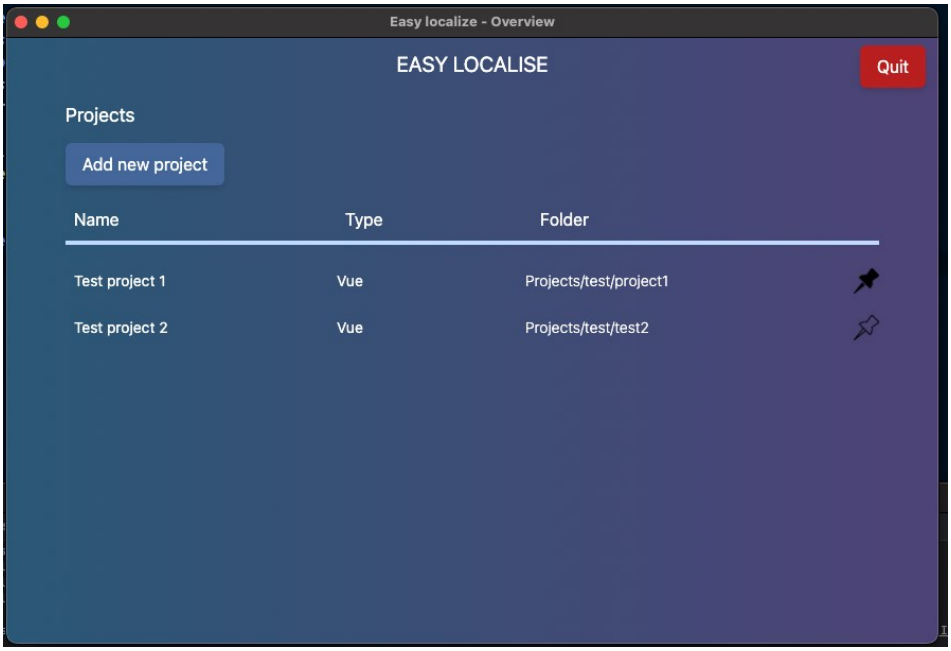
je više *Route* komponenti. `Switch` komponenta označava da se može prikazati samo jedna komponenta ovisno u putanji koju korisnik koristi. Svaka `Route` komponenta sastoji se od *path* svojstva koji označava putanju za koju će ta komponenta biti aktivna te se unutar nje nalazi komponenta koja će se u tom slučaju prikazivati. U ovoj aplikaciji imamo samo dvije komponente koje se korisniku prikazuju te se one aktiviraju unutar *Electron* koda.

```
import { Provider } from 'react-redux';
import { HashRouter as Router, Switch, Route } from 'react-
router-dom';
import Editor from './pages/editor/Editor';
import Overview from './pages/overview/Overview';
import store from './redux/store';

function App() {
      return (
            <Provider store={store}>
                  <Router>
                        <Switch>
                              <Route path="/overview">
                                    <Overview />
                              </Route>
                              <Route path="/editor">
                                    <Editor />
                              </Route>
                              <Route path="/">
                                    <Overview />
                              </Route>
                        </Switch>
                  </Router>
            </Provider>
      );
}

export default App;
```

Kôd 4.1 Organizacija React komponenti u projektu

4.1.2. Upravljanje projektima

Prilikom pokretanja aplikacije korisniku se prvo pojavljuje ekran za upravljanje projektima (Slika 4.1 Prikaz početnog ekrana aplikacije). Na tom ekranu korisnik može odabrati projekt, raditi promijene konfiguracije aplikacije i projekta, pričvrstiti projekt radi lakšeg odabira, prikaz svih projekata koje je do sada koristio te brisanje.



Slika 4.1 Prikaz početnog ekrana aplikacije

Svaki projekt ima svoju konfiguraciju pomoću koje se aplikacija drugačije ponaša. Prilikom dodavanja novog projekta korisnik mora podesiti konfiguraciju za taj projekt. Ta konfiguracija uključuje odabir tehnologije koju projekt koristi, zadanog jezika, mape u kojoj se nalazi projekt, mape u kojoj se nalaze svi prijevodi te odabir liste jezika koju korisnik želi podržavati.

Za dokaz koncepta ovog projekta odabrana je tehnologija **Vue.js** tako da korisnik može odabrati za sada samo ovaj tip projekta. Prilikom dodavanja novog projekta sva polja za konfiguraciju su obavezna. Zadani jezik je obavezan zbog načina na koji aplikacija radi. Aplikacija na temelju zadanog jezika uspoređuje ključeve sa ključevima datoteka od drugih jezika te na osnovu njih zaključuje da li postoje prijevodi za te jezike. Korisnik također mora

odabrati mapu za jezike iz razloga sto je svaki projekt drugačije strukturiran. Radi pojednostavljenja ovog rada odabran je **JSON** format te datoteke za prijevode imaju naziv u formatu koda **alpha2**[5] pojedinog jezika. Svaka datoteka za prijevode je organizirana na način da se sastoji od ključeva (identifikatora) i prijevoda. Svaka datoteka za prijevode treba sadržavati sve ključeve koje ima datoteka sa zadanim jezikom. Ukoliko ne sadrži ključ sa prijevodom smatra se da prijevod za taj jezik ne postoji. Sama konfiguracija se sprema i učitava u korisničku „app data“ mapu čija lokacija ovisi o operacijskom sustavu koji korisnik koristi.

Prikaz se sastoji od `Overview` komponente (Kôd 4.2) koja u sebi sadrži `ProjectList` komponentu, naslova te gumba za izlaz iz aplikacije.

```
import ProjectList from './components/ProjectList';
import './Overview.scss';
function Overview() {
	function quitApplication() {
		window.close();
	}
return (
	<main className="overview bg-gradient-to-r dark">
		<h1 className="pt-3 dark:text-white text-center text-xl
		uppercase">
			Easy localise
		</h1>
		<section
		className="container mx-auto mt-5">
			<h2 className="dark:text-white text-lg mb-3">
				Projects
			</h2>
			<ProjectList />
		</section>
		<button
			className="absolute top-2 right-3 px-4 py-2
		dark:text-white bg-red-700 hover:bg-red-600 rounded-md
		shadow-md hover:shadow-lg"
			onClick={quitApplication}>
				Quit
```

[5] Alpha2 je prikaz kodova država pomoću dva znaka

```
            </button>
      </main>
      );
}
export default Overview;
```

Kôd 4.2 Komponenta početnog ekrana

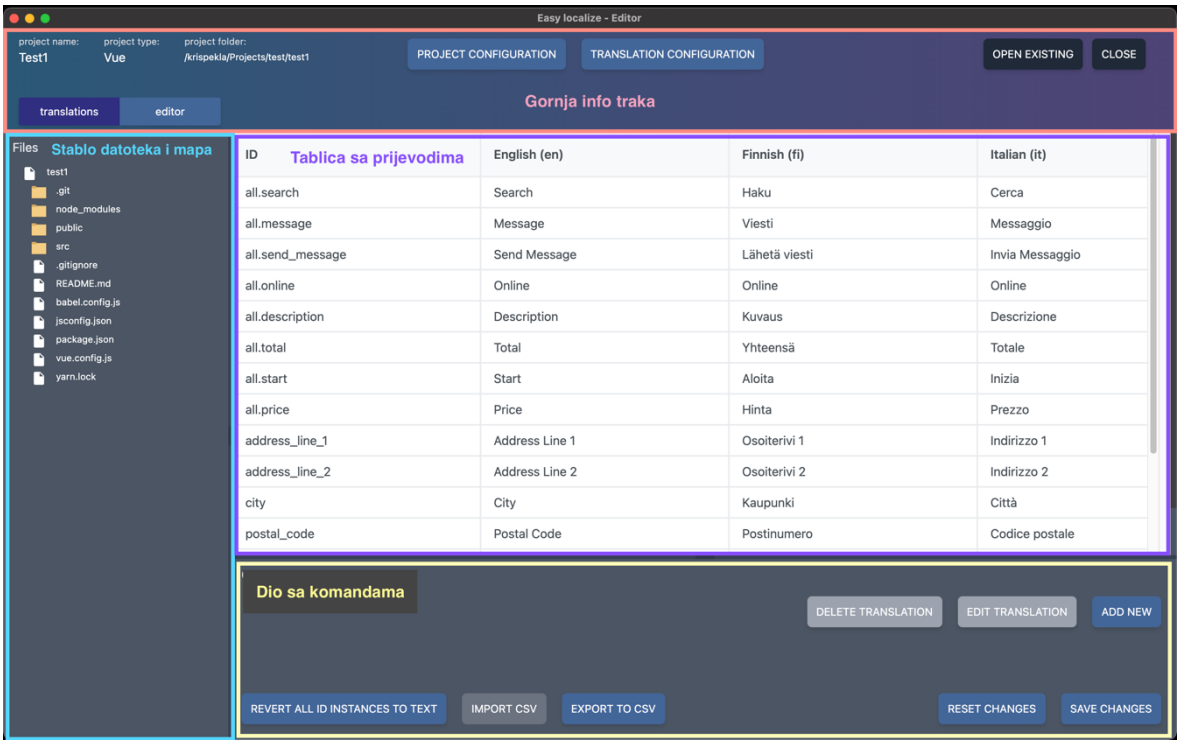
Unutar `ProjectList` komponente (Kôd 4.3) nalazi se lista projekata, kontekstni meni, dijalog za dodavanje novog projekta ili editiranje postojećeg te gumb za dodavanje novog projekta.

```
function ProjectList({ history }: any | HashRouter) {
      . . .
      return (
            <>
                  <button
                        onClick={(e) => setDisplayDialog(true)}
                        className="px-4 py-2 dark:text-white bg-
      queenBlue hover:bg-queenBlueHover rounded-md shadow-md
      hover:shadow-lg">
                        Add new project
                  </button>
                  <ProjectDialog
                        project={currentItem}
                        projectIndex={projectIndex}
                        type={projectDialogType}
                        displayDialog={displayDialog}
                        setDisplayDialog={setDisplayDialog}
                        clearDefault={clearDefaultDialogType}
                  />
                  <div
                  className="flex flex-row mt-5 dark:text-white">
                        <span className="w-4/12 pl-2">Name</span>
                        <span className="w-3/12 pl-2">Type</span>
                        <span className="w-5/12 ">Folder</span>
                  </div>
                  <hr
                  className="bg-blue-200 h-1 border-0 mt-2 mb-3" />
                  <ContextMenu
                        model={contextMenuItems}
```

```
                        ref={contextMenuRef}
                        onHide={onItemContextMenuClickHide} />
                    {
                        projects.map((project: Project, key:
                    number) => projectItemRenderer(project, key))
                    }
                    {
                        projects.length < 1 &&
                        <span className="text-gray-200 ml-
                    3">Project list is empty!</span>
                    }
                </>
        );
    }
    export default withRouter(ProjectList);
```

Kôd 4.3 Komponenta za upravljanje projektima

4.1.3. Uređivač prijevoda



Slika 4.2 Prikaz različitih sekcija editora

Nakon što je korisnik dodao projekt te ga odabrao na prvom prozoru, otvara se novi prozor preko cijelog ekrana koji sadrži razne sekcije vezane uz taj projekt te prijevode (Slika 4.2

Prikaz različitih sekcija editora). Sastoji se od gornje trake koja sadrži opće informacije o projektu te gumb za konfiguraciju, prikaza stabla datoteka i mapa, tablice sa prijevodima, uređivača (engl. editora) pojedine datoteke te komandnog dijela koji sadrži gumbe za upravljanje prijevodima.

Komponenta `Editor` (Kôd 4.4) u sebi sadrži `Splitter` i `SplitterPanel` komponente [4] koje služe za razdvajanje sekcija te time nadodaju mogućnost proširivanja, smanjivanja određenih sekcija po želji korisnika. Unutar `Editor` komponente nalaze se svi važni dijelovi koji su prikazani na slici. Možemo ih razdvojiti na tri sekcije. Prva je info traka koja u sebi sadržava komponente `ProjectInfo`, `ProjectCommands` te gumb za prebacivanje između tablice sa prijevodima i prikaza pojedine datoteke. `ProjectInfo` komponenta je zadužena za prikaz osnovnih informacija o projektu poput imena projekta, tipa i putanje na kojoj se nalazi. `ProjectCommands` sadrži gumbe za otvaranje dijaloga za izmjenu postavki projekta te gumb za izlaz iz projekta. Druga sekcija se sastoji od komponenta `FileTree`, `TranslationEditor` i `FileEditor`. `FileTree` komponenta je zadužena za prikaz stabla datoteka i mapa. `TranslationEditor` sadrži tablicu sa svim prijevodima te dijalog za prijevode. `FileEditor` je zadužen za prikaz odabrane datoteke u `FileTree` komponenti te označavanje prijevoda pronađenih unutar datoteke drugom bojom te omogućavanje odabira toga prijevoda i mijenjanje. Treća sekcija se sastoji od komponenti sa komandama za manipuliranje odabranih prijevoda unutar druge sekcije. Čine ju komponente `FileEditorCommand` i `TranslationEditorCommand` te se prikazuju ovisno o trenutno aktivnoj komponenti u drugoj sekciji.

```
import { Splitter, SplitterPanel } from 'primereact/splitter';
function Editor({ history }: any | HashRouterProps) {
      const [modeToggle, setModeToggle] = useState('translations');
      const currentProject: Project = useAppSelector(
            (state) =>
      state.settings.projects[state.settings.currentProject]
      );
      return (
            <div className="editor flex flex-col dark pt-2">
                  {currentProject && (
                  <header
                  className="flex flex-col justify-between px-5">
```

```
                  <div className="flex justify-between">
                        <ProjectInfo />
                        <ProjectCommands />
                  </div>
                  <div className="pb-2">
                        <SelectButton
                              className="ml-19"
                              value={modeToggle}
                              options={toggleModeOptions}
                              unselectable={false}
                              onChange={
                               (e) => setModeToggle(e.value)
                              }
                        />
                  </div>
            </header>
)}
{currentProject && (
<main className="">
            <Splitter
            className="min-w-full min-h-full" gutterSize={5}>
                  <SplitterPanel
                  className="filetree-splitter">
                        <FileTree />
                  </SplitterPanel>
                  <SplitterPanel className="splitter-editor">
                        <Splitter
                         layout="vertical" gutterSize={5}>
                              <SplitterPanel size={70}>
                                    <div
                              className={modeToggle ===
                              'editor' ? 'hidden' : 'block'}>
                                    {<TranslationEditor />}
                                    </div>
                                    <div
                              className={modeToggle ===
                              'editor' ? 'block' : 'hidden'}>
                                    {<FileEditor />}
                                    </div>
                              </SplitterPanel>
```

```
                                    <SplitterPanel size={30}>
                                          {
                                    modeToggle === 'editor' ?
                                    <FileEditorCommand /> :
                                    <TranslationEditorCommand />
                                          }
                                    </SplitterPanel>
                              </Splitter>
                        </SplitterPanel>
                  </Splitter>
            </main>
            )}
      </div>
      );
}
export default withRouter(Editor);
```

Kôd 4.4 Editor komponenta

Tablica sa prijevodima i uređivač se ne prikazuju istovremeno nego je potrebno koristiti gumb za prebacivanje (engl. *toggle*) iz razloga što nema dovoljno prostora na ekranu za prikaz obje komponente. Ovisno o odabiru tablice i uređivača, gumbi u komandnom dijelu se mijenjaju. Prozor ne mora nužno biti preko cijelog ekrana nego se može smanjivati ovisno o korisnikovim željama. Nadodane su minimalne dimenzije koje prozor može imati kako bi aplikacija dobro izgledala te bi se vidjeli svi dijelovi uređivača. Klikom gumba za konfiguraciju u gornjoj traci korisniku se otvara dijalog koji je isti kao i dijalog koji se koristi prilikom editiranja, dodavanja projekta. Korisnik unutar dijaloga može vidjeti te mijenjati postavke projekta. Na gornjoj desnoj strani nalazi se gumb za zatvaranje projekta nakon kojeg korisnik ponovno vidi prozor za upravljanje projektima.

4.2. Korištene tehnologije

Tehnologije koje su se koristile unutar aplikacije vezane su uz web svijet tako da su većinom vezane za JavaScript tehnologiju.

4.2.1. JavaScript

JavaScript je jednostavan, interpretirani skriptni jezik nastao 1995. godine kao način za dodavanje interakcije unutar preglednika. Nalazi se unutar svih preglednika te je od tada evoluirao te omogućio stvaranje modernih web aplikacija. Ima slično ime kao programski jezik *Java*, ali programski jezik kao takav nema nikakve sličnosti sa *Javom*. U početku se koristio kao način za animaciju, interakciju unutar web stranica, ali je sa vremenom evoluirao te omogućio široku primjenu uključujući kreiranje složenih sustava. Najveća beneficija ove tehnologije je njena široka primjena te jednostavnost pisanja koda. S druge strane jednostavnost je i najveća mana jer otežava *debugiranje* te ispod sebe sadrži mnoštvo kompleksnosti koje također smanjuju performanse.

4.2.2. TypeScript

TypeScript je JavaScript jezik koji je proširen sa sustavom za statičke tipove. To znači da je svaki TypeScript program ujedno i JavaScript program. TypeScript datoteke koriste *„.ts“* nastavke dok JavaScript **„.js“** . Koristi istu sintaksu kao i *JavaScript* te nadodaje novu čime je prelazak postojećih *JavaScript* aplikacija na TypeScript vrlo jednostavan. Ako želimo izvršiti TypeScript kod potrebno ga je prvo prevesti u *JavaScript*. Prevoditelj tijekom prevađanja radi razne provjere uključujući za statičke tipove čime smanjujemo mogućnost za greškama u kodu prilikom izvršenja. Statički tipovi su zbog toga najveća prednost prilikom korištenja ovog jezika. Varijabla koja je deklarirana ne može mijenjati tip te može koristiti samo definirane tipove. Jedna od prednosti zbog ovog načina pisanja koda su razni alati koje programerima vrlo rano otkrivaju greške te im time pomažu u produktivnosti. Neke ostale prednosti su lakše izmjene koda, bogatije iskustvo programera unutar razvojnog okruženja, prelazak novih postojećih JavaScript programera na TypeScript može ići postepeno, čitljivost koda i lakše čitljiva programska sučelja tijekom korištenja tuđeg koda. Najveća mana TypeScript-a je što nema prave statičke tipove jer se na kraju prevađa u JavaScript tako da tijekom izvršenja programa ti statički tipovi ne postoje te se ne mogu

izvršiti potrebne provjere. Još jedna mana je što programeri moraju pisati više koda što usporava cijeli proces. Zbog toga je za male projekte preporučljivo držati se što jednostavnijeg te koristiti JavaScript dok za srednje i veće projekte na kojima sudjeluje više programera koristi TypeScript-*a* dolaze do izražaja. Neke ostale mane su pokretanje TypeScript-a unutar preglednika ima dodatan korak za prevađanje.

4.2.3. Node.js

Node.js je poslužiteljska platforma koja koristi *Chrome-ov*, ***V8 JavaScript runtime engine***. Sam pogon (engl. *engine*) napisan je u programskom jeziku *C++* te on izvršava JavaScript kod. Razvio ga je 2009. godine *Ryan Dahl* zajedno sa svojim kolegama te je to projekt otvorenog koda. Jezik u kojem se pišu Node.js programi je **JavaScript**. Glavni problem koji je riješio je bio problem konkurentnosti(mogućnost izvršenja više zahtjeva istovremeno) tadašnjih web servera koji nisu omogućavali više od 10.000 istovremenih zahtjeva. Među-platformski[6] je tako da se može koristiti na gotovo svim operacijskim sustavima. Asinkron je te je baziran na događajima (engl. *event driven*). Najvažnija značajka je to što su svi događaji ne blokirajući (engl. *non-blocking*) tj. dva različita događaja koja se izvršavaju nisu ni na koji način ovisna jedna o drugom u odnosu na resurse poslužitelja nego su neovisna. Programer je pišući JavaScript koji se izvršava unutar preglednika limitiran određenim sučeljima kojima nema pristup zbog sigurnosti samog korisnika. Za razliku od preglednika koji je limitiran određenim sučeljima, *Node.js* ima dodatna sučelja kojima je unutar koda omogućen puno veći pristup samom računalu na kojemu se izvršava čime ima puno više mogućnosti. Pomoću *Node.js* sučelja programer ima pristup mapama i datotekama (engl. *file system*) operacijskog sustava na kojemu se kod izvršava. To je i jedan od glavnih razloga zašto ovaj projekt koristi ovu tehnologiju. Neke ostale značajke ove platforme je bolja produktivnost programera jer se za poslužiteljske i klijentske poslužitelje koristi isti programski jezik (JavaScript), postoji ogromni broj besplatnih *NPM paketa*, ima dobre performansa te ukoliko su potrebne još veće performanse moguće je pisati dodatke u *C++* jeziku. Mane su mu smanjene performanse za teške računalne zadatke, mogućnost sigurnosnih propusta u drugim *NPM paketima*, održavanje koda može biti komplicirano ukoliko nije dobro strukturirano.

[6] Među-platformski znači da se kod može izvršavati na više različitih operacijskih sustava.

4.2.4. Electron

Electron je projekt otvorenog koda koji omogućuje kreiranje aplikacija za radnu površinu. Kreirao ga je inženjer *Cheng Zhao* koji je tada radio za *Github*. Na početku se zvao *Atom shell* jer je nastao za potrebe *Atom editora* da bi kasnije bio preimenovan u *Electron*. Velika prednost *Electrona* naspram ostalih sučelja za izgradnju aplikacija za radnu površinu je što omogućuje korištenje web tehnologija za kreiranje aplikacija koje se ponašaju kao klasične aplikacije za radnu površinu te ima podršku za više platformi. Programeri mogu kreirati grafička sučelja te pristupati raznim sučeljima operacijskog sustava uključujući datotečnom sustavu za razliku od internetskih preglednika koji imaju ograničen pristup zbog sigurnosti korisnika. *Electron* se sastoji od **Node.js** i **Chromium content modula**. *Chromium* je internetski preglednik, projekt otvorenog koda koji se koristi u preglednicima kao *Chrome*, *Microsoft Edge*, *Opera* te drugi.

Content modul je modul unutar *Chromiuma* koji je zadužen za prikazivanje sadržaja unutar preglednika. On sadrži ono najosnovnije za prikaz HTML, CSS i JavaScript datoteka. U *Electron* aplikaciji postoje dvije vrsta procesa. Postoji glavni proces te proces za renderiranje kojih može biti više. Glavni proces ima različite uloge od kojih su neke pokretanje i gašenje aplikacije, definiranje i prikaz prozora, kreiranje i uništavanje procesa za renderiranje, pristup određenim sučeljima operacijskog sustava kao datotečni sustav, prikaz dijaloga za datoteke te mnogim drugima. Procesi za renderiranje služe primarno za prikaz te sa druge strane su vrlo ograničeni jer koriste web tehnologije koje možemo naći u internetskim preglednicima. Za bilo kakav pristup nekom sučelju operacijskog sustava proces za renderiranje mora komunicirati za glavnim procesom. Interprocesna komunikacija se sastoji od dva dijela **ipcMain** i **ipcRenderer**. Ta dva dijela međusobno komuniciraju korisnički definiranim kanalima te mogu komunicirati u oba smjera. Najpoznatije *Electron* aplikacije su *Visual Studio Code*, *Slack*, *Atom*, *Microsoft Teams*, *Skype*, *WhatsApp Desktop* te mnoge druge. S obzirom da *Electron* koristi *Node.js,* jezik koji se koristi za pisanje *Electron* aplikacija je *JavaScript*. To puno olakšava programerima kreiranje aplikacija jer ne moraju učiti dodatne tehnologije da bi kreirali aplikacije za radnu površinu nego se mogu fokusirati na samu *Electron* tehnologiju.

4.2.5. React.js

React.js je *JavaScript* biblioteka koja služi za izgradnju korisničkih sučelja. Prototip (*FaxJS*[7]) je nastao 2011. godine u kompaniji *Facebook* iz potrebe kompanije za sustavom koji će olakšati dodavanje novog koda te upravljanje postojećim. 2012. godine *Facebook* je kupio kompaniju Instagram koja je htjela koristiti tu tehnologiju te je 2013. godine *Facebook* osmislio naziv *React* te prebacio u projekt otvorenog koda. Stabilna verzija je stigla 2015. godine te su od tada mnoge velike kompanije počele prihvaćati i koristiti ovu tehnologiju. Neke od njih su osim *Facebooka* i *Instagrama*, *Uber*, *Airbnb*, *Reddit*, *Netflix*, *Dropbox*, *Discord*, *Reddit* te mnoge druge kompanije [2].

Prije popularnosti *React.js-a* web stranice i aplikacije su se renderirale na poslužiteljima te je korisnik zatim dobio gotove HTML, CSS i JavaScript datoteke. JavaScript je sadržavao kod za određenu interakciju, ali za bilo kakvu veću radnju korisnika, preglednik je morao dohvaćati nove datoteke sa poslužitelja koje je zatim preglednik iz početka iscrtavao. Time je korisničko iskustvo bilo limitirano na način da nije bilo moguće napraviti kompliciranije sučelje koje je koristilo nekakvu standardnu biblioteku i bilo razumljivo svim programerima. React biblioteka to je riješila na način da se sav kod više nije *renderirao* na serveru nego je korisnik jednom dohvatio React biblioteku i kod koji je kostur aplikacije te bi zatim ovisno o interakciji dohvaćao manje dijelove koje su mu bile potrebne ovisno o interakciji korisnika sa aplikacijom (slike, videa, dijelove koda). Korisnik bi također dohvatio samo jednu HTML datoteku koja bi bila povezana sa JavaScriptom te je JavaScript kod ovisno o željenom prikazu dodavao i micao HTML elemente. Prije popularnosti React.js-a programeri su morali manualno manipulirati DOM-om što je kreiralo mnoštvo prepreka. Sa pojavom *React.js-a* te sličnih biblioteka one su dodale mehanizme za manipulaciju DOM-om tako da se programeri nisu morali fokusirati na manualno dodavanje i micanje HTML elemenata jer je to za njih obavljala React.js biblioteka. To je i uvelike olakšalo programerima kreiranje web aplikacija jer su se mogli fokusirati na viši nivo apstrakcije te su web aplikacije postale jako popularne.

[7] FaxJS je prva verzija React-a koja se koristila interno unutar Facebook kompanije

React aplikacije se sastoje od **komponenti**. Komponenta predstavlja najmanju jedinicu koda koji predstavlja dio prikaza u DOM-u. Komponente se nalaze jedna unutar ili pokraj druge komponente pa možemo reći da su ***composables***. Svaka komponenta u sebi sadrži *render* funkciju koja je zadužena za prikaz html elemenata te komponente.

Glavna karakteristika *React.js* biblioteke je ***virtualni DOM***. *Document Object Model* (DOM)[10] je sučelje za web dokumente koje je strukturirano na način da omogućuje programskim jezicima interakciju sa tim dokumentom koji predstavlja web stranicu. Ima određenu strukturu i sadržaj koji je propisan standardom te kojeg se mora držati. Stablastog je oblika te se sastoji od mnoštva čvorova (engl. *node*). Manipulacija DOM-om u pregledniku je vrlo spora te je prije pojave modernih JavaScript preglednika bilo puno teže pisati dobar i čitljiv kod koji je mogao raditi složene tranzicije, animacije te manipulacije. *React.js* ima virtualni DOM koji umjesto da direktno radi promjene elemenata na pravom DOM-u, kalkulira sve promjene u memoriji virtualnog DOM-a te zatim gleda razliku između pravog i virtualnog DOM-a te radi minimalne promjene na pravom DOM-u. Time je React.js riješio problem sporosti manipuliranja pravim DOM-om te stvorio lakši način za programere da kreiraju složeni grafički prikaz.

4.2.6. HTML

HTML je kratica od *HyperText Markup Language* te služi za stvaranje HTML dokumenata. HTML jezik oblikuje strukturu i sadržaj web stranica. Preglednici učitavaju HTML dokumente u svoj pogon (engl. *engine*) te ih zatim obrađuju zajedno sa drugim tehnologijama (CSS, JavaScript) te korisniku prikazuju stranicu. Nastao je 1990. godine od strane *Tim Berners Lee*, a prva verzija objavljena je 1993. godine. Od tada pa sve do verzije 4.0 objavljene u prosincu 1997. godine bilo je različitih promjena da bi se od verzije HTML 4.0 specifikacija ustalila te ostala vrlo slična ako ne i ista današnjoj. Verzija HTML 4.0 je prva verzija koja je bila namijenjena za upotrebu u cijelom svijetu jer je podržavala sve internacionalne karaktere (nešto slično današnjem *unicode*[8] standardu), podržavala je umetanje CSS-a, skriptiranje, forme, tablice te pristupačnost. U prosincu 1999. godine izlazi verzija 4.01 sa nekim malim promjenama više kao nadopuna koja je riješila do tada sve

[8] Unicode je standard za prikaz znakova neovisno o jeziku.

uočene probleme. Verzija HTML 5 izlazi 2014. godine te nadodaje nove semantičke elemente ("section", "header", "footer", "template" i mnoge druge).

HTML dokument se sastoji od HTML elemenata. Svaki element se sastoji od otvorene i zatvorene oznake (engl. tag). Svaka pojedina oznaka započinje znakom "<" zatim nazivom HTML elementa te završava znakom ">". Zatvorene oznake također sadrže kosu crtu prije naziva elementa. Između otvorene i zatvorene oznake može se nalaziti tekst sadržaja ili drugi elementi. Svaka otvorena oznaka može u sebi sadržavati atribute koji definiraju određena svojstva tog elementa. Svaki HTML dokument mora sadržavati HTML verziju koja koristi *("<!DOCTYPE>"* za *HTML 5*) te *html element* unutar kojega se obavezno mora nalaziti ***head*** te ***body*** *element*. Unutar *head elementa* se nalaze i stavljaju podaci o tom dokumentu te uvoz drugih skripti i stilova. Unutar *body elementa* se stavljaju elementi koji definiraju strukturu i sadržaj dokumenta koji će se korisniku prikazati.

4.2.7. CSS/SCSS

CSS (*Cascading Style Sheets*) je jezik koji određuje kako će izgledati prikaz HTML dokumenata. On određuje kako će se HTML *elementi* prikazivati, izgledati unutar preglednika. Zajedno sa HTML i JavaScript tehnologijom čini osnovnu bazu koja je standardizirana te se nalazi unutar svih preglednika. Do sada je bilo tri verzije. Prva službena verzija izašla je 1996. godine te je bila ograničena brojem stilskih svojstava. Druga verzija izašla je 1998. godine te je nadodala puno novih svojstava. Treća verzija izašla je 1999. godine te je omogućila da se stilska pravila mogu podijeliti u posebne module, podršku za još novih stilskih pravila, animacije i korištenje specijalnih fontova (kao na primjer *Google Fonts*). Stilovi se mogu definirati unutar HTML dokumenta sa *style* oznakom ili u posebnoj datoteci. Sintaksa joj je vrlo jednostavna te se brzo uči. Sastoji se selektora te deklaracijskog bloka. Selektor definira određeni dio na kojeg će se definirani stilovi odnositi. Mogu biti svi elementi određenog tipa HTML *elementa*. Zatim mogu biti definirani po atributu, identifikatoru (*id*), klasi (*class*) te pseudo svojstvu. Može se koristiti više selektora za definiranje istog deklaracijskog bloka. Unutar deklaracijskog bloka nalazi se lista stilskih svojstva koji se primjenjuju za selektor. Stilska svojstva mogu biti jedino definirana pravila te su obično engleske riječi. Stilsko svojstvo se sastoji od naziva svojstva i vrijednosti koju

želimo primijeniti na selektor. Nakon naziva svojstva se obično stavlja dvotočka (:) te zatim ide vrijednost i na kraju vrijednosti točka sa zarezom (;).

Ponekad je selektor zahtijevao da naziv bude jako dug. Više tako definiranih selektora dovodio je do toga da kod bude teže čitljiv. Zbog toga je nastao skriptni jezik SCSS. **SCSS** jezik je prošireni CSS jezik. On se interpretira i kompajlira u CSS jezik. Olakšava pisanje složenih selektora, korištenje varijabla za vrijednosti svojstva te mnoga druga svojstva. Glavna razlika u sintaksi je što pomoću SCSS jezika možemo pisati selektore unutar deklaracijskog bloka čime je puno jednostavnije čitati i pisati kod.

4.2.8. PrimeReact

PrimeReact je biblioteka *React komponenti* te je projekt otvorenog koda. Koristi se u velikim korporacijama kao *Ebay*, *Intel*, *Mercedes-Benz*, *Audi* te mnogi drugi. Jedan od razloga zašto se koristi u ovom projektu je zato što je biblioteka dobro održavana zahvaljujući velikim korporacijama koje ju sponzoriraju. Korištenjem ove biblioteke značajno je olakšano kreiranje polja za unos podataka, kontekstnih izbornika te drugih komponenti.

4.2.9. Redux

Redux je biblioteka za upravljanje stanjem aplikacije. Postala je standardna tehnologija koja se koristi unutar *React aplikacija*. Za razliku od pojedinih komponenti koje su međusobno isprepletene i povezane, *Redux* sadrži **store** koji je centralno mjesto gdje su podaci koji se koriste za prikaz. Za razliku od komponenti gdje komunikacija mora slijediti određena pravila, komunikacija između *Redux store-a* te svake komponente je direktna. Time je značajno pojednostavljeno upravljanje podacima za prikaz te otkrivanje grešaka. Komunikacija između komponente i *store-a* se odvija na poseban način koji je definiran *Reduxom*. Sastoji se od *store-a*, akcija (actions) te *reducer-a*. Store je mjesto gdje su definirani podaci čije vrijednosti označavaju trenutno stanje unutar aplikacije. Akcija je jednostavan JavaScript objekt koji označava naziv za pozivanje *reducera*. *Reducer* je *JavaScript funkcija* koja prima trenutno stanje unutar aplikacije te radi određene promjene na njemu.

4.2.10. React Hook Form

React Hook Form je biblioteka za izgradnju formi, sučelja i validacija. Sadrži mehanizam za validacije koje su jednostavne za upotrebu. Lako se integrira sa ostalim bibliotekama te je napravljena u *TypeScriptu*.

4.2.11. Create React App

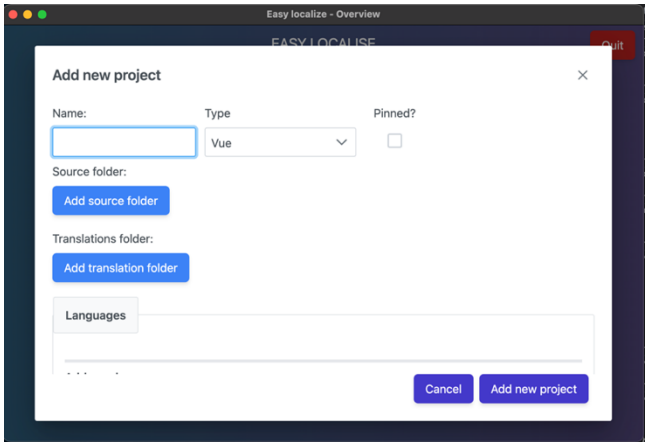
Create React App (skraćeno CRA) je predefinirani predložak (engl. template) koji postavlja okruženje za razvoj *React.js* aplikacija. Namjena mu je da olakšava programerima konfiguriranje i korištenje zadnjih *JavaScript* značajki unutar *React.js* projekta. Koristi se unutar ovog projekta kod dijela za prikaz upravljanja projektima te uređivača prijevoda.

4.2.12. Vue.js

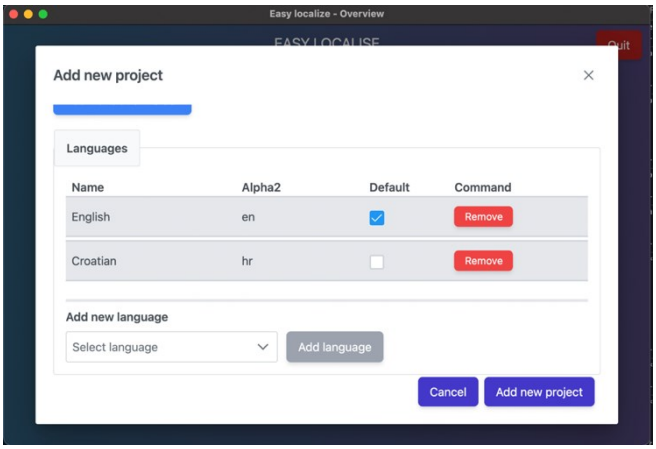
Vue.j*s* je JavaScript okvir koji služi za izgradnju korisničkih sučelja. *Vue.js* se nadovezuje na tehnologije kao HTML, CSS i JavaScript. Kreirao ga je *Evan You* 2014. godine iz nezadovoljstva u radu sa *AngularJS JavaScript okvirom* nakon što je prestao raditi za *Google*. *Vue.js* aplikacija se sastoji od **Vue komponenti**. Svaka komponenta se sastoji od tri dijela. *Template* dio koji sadrži HTML *elemente*, *JavaScript* dijela i dijela za stilove. Sadrži virtualni DOM te je vrlo sličan React.js biblioteci. Glavna razlika između tih dviju tehnologija je što je *React* biblioteka (engl. *library*) dok je *Vue radni* okvir (engl. *framework*). Razlika između biblioteke i radnog okvira je što radni okvir sadrži unutar sebe sve potrebne alate za razvoj proizvoda dok biblioteka je namijenjena rješavanju određenog problema te se očekuje da se uz nju koriste i druge biblioteke. Razlika je također u sintaksi jer React koristi *JSX* dok Vue **HTML template**. U ovom radu se koristi jednostavan Vue.js projekt za dokaz koncepta ove aplikacije. Za prijevode *Vue.js* okvir koristi *Vue i18n plugin* [6] za internacionalizaciju tako da će samo prepoznavanje prijevoda unutar aplikacije biti olakšano formatom koji je zadan ovim dodatkom (engl. *pluginom*).

4.3. Dodavanje korisnikovog projekta

Prilikom pokretanja aplikacije otvara se prozor sa listom projekata koje je korisnik do sada dodao. Na projekte je moguće staviti oznaku tako da uvijek budu pri vrhu. Početni ekran se sastoji od gumba za dodavanje novog projekta, liste projekata, kontekstualnog izbornika unutar liste projekata te gumba za izlaz iz aplikacije. Klikom na gumb za dodavanje novog projekta otvara se dijalog sa formom koju je potrebno popuniti kako bi projekt bio uspješno dodan te bi kasnije mogao učitati u memoriju (Slika 4.3), (Slika 4.4). Unutar liste projekata moguće je desnim klikom aktivirati kontekstni izbornik koji zatim nudi opcije otvaranja projekta, editiranja i brisanja iz liste nadodanih projekata. Također pokraj svakoga projekta nalazi se ikona pomoću koje se projekt može zakačiti tako da bude uvijek na vrhu. Klikom na red od određenog projekta, otvara se uređivač prijevoda u novom ekranu, zatvara se ekran sa upravljanjem projektima te se učitava projekt u memoriju.



Slika 4.3 Unos općih informacija o projektu



Slika 4.4 Unos jezika koje korisnik želi podržavati u projektu

Kako bi korisnik dodao projekt u aplikaciju potrebno je kliknuti na gumb „Add new project“. Zatim se otvara dijalog sa formom koja se sastoji od različitih polja. U prvom polju korisnik unosi naziv projekta, zatim odabire tip projekta. Zbog kompleksnosti podržavanja više tipova projekata za sada je omogućen samo *Vue.js* tip projekta. Nakon toga može odabrati da li će se projekt nalaziti na vrhu liste projekata ili ne. Iduće što korisnik mora unijeti je mapa u kojoj se nalazi projekt te mapu gdje se nalaze prijevodi. Kasnije će se prilikom učitavanja projekta ova informacija koristiti kako bi aplikacija mogla proći po svim datotekama te pronaći prisutne prijevode. Nakon unosa mape nalazi se sekcija za jezike koje korisnik želi podržati u svom projektu. Odabirom jezika i klikom „Add language“ jezik se dodaje u listu jezika te se automatski selektira prvi dodani jezik kao zadani. Kako bi korisnik mogao dodati projekt potrebno je ispuniti sve navedene unose osim kučice za kačenje (engl. *pin*) projekta. Također potrebno je odabrati bar jedan jezik koji će biti zadani. Kasnije će zadani jezik biti jako važan prilikom dodavanja novih prijevoda, prevođenja korištenjem *Google Translate API* te pronalaska neprevedenih riječi. Klikom na gumb „Add new project“ dijalog se zatvara te se projekt dodaje u listu projekata. Svi projekti su spremljeni u datoteku koja je u JSON formatu te se nalazi u mapi od korisničkog računa sustava kojeg korisnik koristi. Ovisno o operacijskom sustavu koji korisnik koristi, Electron programsko sučelje ima predefinirane putanje gdje se te mape nalaze. Prilikom pokretanja aplikacije iz konfiguracijske datoteke učitavaju se sve spremljene vrijednosti, uključujući i lista projekata.

Dijalog za dodavanje novog projekta koristi *Dialog* komponentu *PrimeReact* biblioteke (engl. *library*). Unutar komponente (Kôd 4.5) se nalaze elementi koji se koriste za unos postavki projekta. Komponenta se može koristiti za dodavanje novog projekta ili izmjenu postavki postojećeg projekta. Ovisno o slučaju prikazujemo odgovarajući tekst koji je proslijeđen u header svojstvu *Dialog* komponente. Kroz svojstvo (engl. *property*) footer prosljeđujemo JSX kod koji sadrži naredbe za spremanje, skrivanje ili resetiranje vrijednosti dijaloga.

```
import { Dialog } from 'primereact/dialog';
. . .
export interface ProjectDialogInterface {
      project?: Project;
      type: ProjectDialogEnum;
      displayDialog: boolean;
      setDisplayDialog: any;
```

```
      projectIndex: number;
      clearDefault: any | undefined;
}

const ProjectDialog = (props: ProjectDialogInterface) => {
      . . .
      const renderDialogAddNewFooter = () => {
            return (
                  <div>
                        <button
                              onClick={() => onHideDialog()}
                              className="{ . . . }">
                              Cancel
                        </button>
                        <button
                             onClick={
                              handleSubmit(onConfirmDialog)
                              }
                              disabled={false}
                              className={ . . . }>
                              {
                              props.type === ProjectDialogEnum.add
                              ? 'Add new project' : 'Update
                              project'
                              }
                        </button>
                  </div>
            );
      };

      return (
            <>
                  <Dialog
                        header={`${props.type ===
ProjectDialogEnum.add ? 'Add new' : 'Update'} project`}
                        visible={props.displayDialog}
                        style={{ width: '90vw' }}
                        draggable={false}
                        footer={renderDialogAddNewFooter()}
                        onHide={() => onHideDialog()}>
```

```
                . . .
                </Dialog>
            </>
        );
};
export default ProjectDialog;
```

Kôd 4.5 Komponenta za dodavanje novog projekta

Za upravljanje vrijednostima svih polja koristi se `useForm` od *React Hooks form* biblioteke (Kôd 4.6) [7]. Ona nam olakšava prikaz validacijskih poruka, smanjuje količinu koda koja bi bila potrebna za pripremu podataka te omogućuje dinamičko dodavanje i registriranje novih polja.

```
let formDefaultValues: Project =
                props.type === ProjectDialogEnum.edit &&
                props.project
                ? cloneDeep<Project>(props.project)
                : {
                            name: '',
                            src: '',
                            isPinned: false,
                            translationFolder: '',
                            languages: [],
                            defaultLanguage: null,
                            projectType: ProjectType[1],
                            excludedFolders: [],
                  };
. . .

const {
      register,
      handleSubmit,
      watch,
      setValue,
      getValues,
      trigger,
      formState: { errors },
      control,
      reset,
} = useForm<Project>({
```

```
            defaultValues: { ...formDefaultValues },
        });
```

Kôd 4.6 Postavljanje inicijalnih vrijednosti koristeći React Hooks form

Za dodjeljivanje određenog polja `useForm` objekta koristimo *Controller* komponentu (Kôd 4.7) koja se automatski nadodaje i pretplaćuje na događaje input elementa koji su definirani u *render* proslijeđenom svojstvu. Na `useForm` objektu postoji `errors` objekt koji sadrži sve validacijske poruke u slučajima kada oni postoje. Validacijske poruke su odijeljene nazivom polja tako da možemo lako razlikovati greške za pojedino polje.

```
        <div className="flex flex-col">
            <label htmlFor="projectName">Name:</label>
            <Controller
                name="name"
                control={control}
                rules={{ required: 'Project name is required.' }}
                render={({ field, fieldState }) => (
                    <InputText
                        id={field.name}
                        {...field}
                        autoFocus
                        value={field.value}
                        className={`mr-3 w-72 h-10 my-2 ${
fieldState.invalid && fieldState.isTouched && 'border-red-400'
                        }`}
                    />
                    )}
            />
            {errors['name'] && <small className="p-
error">{errors['name'].message}</small>}
        </div>
```

Kôd 4.7 Način korištenja polja za unos koristeći React Hoks form

Za dodavanje novih jezika u listu jezika, koristimo drugačiji pristup zato jer ne znamo točan broj dodanih jezika. Koristimo *watch* funkciju koja se nalazi na *useForm* objektu koja prima kao argument naziv polja koje želimo pratiti te prilikom bilo kakve promijene dodjeljuje nove vrijednosti tom polju te ih u ovom slučaju ponovo prikazuje (Kôd 4.8).

```
        const watchLanguages = watch('languages');
        . . .
```

```
<Fieldset
            {...register('languages', {
required: 'Setting languages is required',
            })}
            {...register('defaultLanguage', {
required: 'Setting default languages is required',
            })}
legend="Languages">
. . .
{watchLanguages &&
watchLanguages.map((item, key) => {
      return (
        <div className=". . ." key={key}>
            <div className="w-4/12">{item.language}</div>
            <div className="w-3/12">{item.alpha2}</div>
            <div className="w-2/12">
                  <Checkbox
                  inputId={item.alpha3}
                  onChange={(e) =>
      onChangeDefaultLanguageCheckboxClick(item)}
                  checked={isDefaultLanguageCheckboxCheck(item)}
                  />
            </div>
            <button
              onClick={(e) => onRemoveLanguageClick(item)}
              className=". . .">
              Remove
            </button>
      </div>
      );
})}
      <div className="flex flex-col">
            {errors['languages'] && (
<small className="p-error">Setting languages is required</small>
            )}
      . . .
      <Dropdown
            id="languageDropdown"
            value={languageDropdown}
            key="language"
```

```
                    className="mr-3 w-72"
                    placeholder="Select language"
                    options={filteredLanguages}
                    optionLabel="language"
                    onChange={onLanguageChange}
              />
              <button
                    onClick={onAddNewLanguageClick}
                    disabled={!languageDropdown}
                    className={. . .}>
                    Add language
              </button>
        </div>
  </Fieldset>
```

Kôd 4.8 Dodavanje jezika unutar forme za dodavanje projekta

Dodavanje novih jezik se odvija odabirom jezika iz padajućeg izbornika (engl. *dropdown*) te klikom na gumb koji se nalazi odmah pokraj nakon kojega se poziva funkcija. Funkcija (Kôd 4.9) koristi metode definirane na *useForm* objektu za dohvaćanje i postavljanje novih vrijednosti.

```
              function onAddNewLanguageClick() {
                    let languageList = getValues('languages') as
        Language[];
                    languageList.push(languageDropdown);

                    if (!getValues('defaultLanguage'))
        setValue('defaultLanguage', languageDropdown);
                    setValue('languages', languageList);
                    setFilteredLanguages(
                          filteredLanguages.filter(
                                (x) =>
        !languageList.some((selectedLanguage) =>
        selectedLanguage.language === x.language)
                          )
                    );
                    setLanguageDropdown(null);
                    trigger(['languages', 'defaultLanguage']);
              }
```

Kôd 4.9 Funkcija za dodavanje novog jezika unutar forme

Klikom na gumb za dodavanje projekta pokreće se `handleSubmit()` metoda kojoj je proslijeđena funkcija koju smo definirali, `onConfirmDialog`. U `handleSubmit()` metodi provjeravaju se sve validacije te ako su sva polja ispravno unesena izvršava se kod unutar `onConfirmDialog()` funkcije (Kôd 4.10).

```
import { updateProject } from
'../../redux/slices/settingsSlice';
. . .
const onConfirmDialog = (data: any) => {
            if (props.type === ProjectDialogEnum.add)
dispatch(addProject(data));
            else if (props.type === ProjectDialogEnum.edit &&
props.projectIndex >= 0)
                  dispatch(updateProject({ project: data,
index: props.projectIndex }));

            resetState();
            props.setDisplayDialog(false);
      };
```

Kôd 4.10 Procesiranje dodavanja novog projekta

Unutar funkcije (Kôd 4.11) poziva se *updateProject* akcija koja je definirana unutar *Redux store-a* te se u njoj mijenja ili nadodaje taj projekt te se odašilje događaj prema *Electron* dijelu aplikacije koji je odgovoran za spremanje postavki svih projekata koji se nalaze u memoriji.

```
const initialState: Settings = {
      projects: [],
      currentProject: -1,
};

const updateElectronConfig = (state: Settings) => {
      if (window?.electron?.send) {
            window.electron.send('settings-save',
cloneDeep(state));
      }
};
. . .
updateProject(state, action: PayloadAction<{ project:
Project, index: number }>) {
```

```
            state.projects[action.payload.index] =
action.payload.project
            updateElectronConfig(state)
},
```

Kôd 4.11 Slanje događaja za dodavanje novog projekta u Electron

Electron dio odgovoran je za spremanje postavki projekata (Kôd 4.12).

```
let APP_CONFIG_ROOT_PATH_CONFIG: string;

function checkIfConfigFileExists() {
      if (!APP_CONFIG_ROOT_PATH_CONFIG) {
            const APP_CONFIG_ROOT_PATH =
app.getPath('userData');
            APP_CONFIG_ROOT_PATH_CONFIG =
APP_CONFIG_ROOT_PATH + 'config.json';
      }
}
export function writeAppSettings(event:
Electron.IpcMainEvent, args: any[]) {
      checkIfConfigFileExists();
      const configData = JSON.stringify(args);
      fs.writeFileSync(APP_CONFIG_ROOT_PATH_CONFIG,
configData, 'utf-8');
}
. . .
export const writeSettingsListener = ipcMain.on('settings-
save', writeAppSettings)
```

Kôd 4.12 Spremanje novog projekta u korisničke postavke

4.4. Učitavanje projekta u memoriju

Nakon što korisnik odabere projekt otvara se novi prozor te aplikacija započinje sa učitavanjem niza informacija. **React UI** dio poziva metodu (Kôd 4.13) koje se nalaze u **ipcMain** dijelu zadužene za učitavanje informacija o svim mapama i datotekama kako bi kasnije mogli prikazati stablo datoteka i mapa.

```
function loadFileTree() {
            if (window.electron) {
```

```
                    window.electron.send('read-directory-tree',
                    {
                          path: currentProject.src,
                          ignoredDirectory:
                    currentProject.excludedFolders ?
 currentProject.excludedFolders : ['node_modules'],
                    });
              }
}
```

Kôd 4.13 Slanje događaja za učitavanje svih informacija o datotekama i mapama projekta

Zatim se u *Electron* dijelu prolazi po svim datotekama te učitava stablo projekta (Kôd 4.14).

```
export const readDirectoryTreeListener = ipcMain.on(
      'read-directory-tree',
      readDirectoryTreeHandler
);
```

Kôd 4.14 Slušajuća funkcija za učitavanje datoteka i mapa

Također prilikom prolaženja po svim datotekama aplikacija pretražuje prijevode pronađene unutar datoteka te ih sprema u memoriju. Nakon što su svi podaci obrađeni šalju se iz **ipcMain** dijela nazad u **ipcRenderer** dio koji koristi React.js te se spremaju u **Redux store**. Korisnik na svom zaslonu vidi uređivač prijevoda koji se sastoji od tablice sa prijevodima, uređivača, prikaza stabla datoteka i mapa te komandama za prijevode te pojedine otvorene datoteke.

4.4.1. Prikaz stabla datoteka i mapa

Kako bi mogli učitati stablo datoteka i mapa potrebno je proći po svim datotekama u projektu koje postoje (Kôd 4.15). Da bi to bilo moguće potrebno je koristiti rekurziju ili *while* petlju iz razloga što ne znamo do koje razine se nalaze datoteke.

```
function readDirectoryTreeHandler(
      event: Electron.IpcMainEvent,
      { path, ignoredDirectory }: { path: string;
ignoredDirectory: string[] }
) {
      const loadedDirectoryTree: TreeNode =
readDirectoryTree(path, ignoredDirectory);
```

```
      event.sender.send('read-directory-tree-return',
loadedDirectoryTree);
}

. . .

export function readDirectoryTree(path: string,
ignoredDirectory: string[]): TreeNode {
      const root = new TreeNode(path,
path.slice(path.lastIndexOf('/') + 1, path.length));
      const stack = [root];

      while (stack.length) {
            const currentNode = stack.pop();
            ignoredDirectory.push('node_modules');
            ignoredDirectory.push('.git');
            currentNode.isIgnored =
ignoredDirectory.includes(currentNode.name);

            if (currentNode && !currentNode.isIgnored) {
                  const children =
fs.readdirSync(currentNode.path, {
                        withFileTypes: true,
                  });

                  for (let child of children) {
                        const childPath =
`${currentNode.path}/${child.name}`;
                        const childNode = new
TreeNode(childPath, child.name, child.isDirectory());

                        if
(fs.statSync(childNode.path).isDirectory()) {
                              stack.push(childNode);
                        } else {
                              const translationsInFile =
readTranslationsInVueType(childPath);
                              if (translationsInFile) {

      childNode.setTranslations(translationsInFile);
```

```
                        }
                    }
                    currentNode.children.push(childNode);
                }
                currentNode.sort();
            }
        }
        return root;
}
```

Kôd 4.15 Rekurzivno učitavanje informacija o svim datotekama i mapama

Iz mape prijevoda se učitavaju svi prijevodi u memoriju (Kôd 4.16).

```
        function loadTranslationFiles() {
            if (window.electron) {
                window.electron.send('read-translation-
files', {
                    path:
currentProject.translationFolder,
                    defaultLanguage:
currentProject.defaultLanguage,
                });
            }
        }
. . .
export const readTranslationsListener = ipcMain.on(
        'read-translation-files',
        readTranslationsHandler
);
. . .
function readTranslationsHandler(
        event: Electron.IpcMainEvent,
        { path, defaultLanguage }: { path: string;
defaultLanguage: string }
) {
        const translations: Translation =
readTranslations(path);
        event.sender.send('read-translation-files-return',
translations);
}
. . .
```

```
export function readTranslations(url: string): Translation {
	const files = fs.readdirSync(url).filter((file) =>
path.extname(file) === '.json');
	const loadedTranslationsPerLanguage: any = {};
	files.forEach((file, index) => {
		const rawData = fs.readFileSync(path.join(url,
file));
		const lngAlpha2 = file.slice(0, 2);
		loadedTranslationsPerLanguage[lngAlpha2] =
flatten(JSON.parse(rawData.toString()));
	});
	const loadedTranslations: Translation = {};
	for (const [lng, translations] of
Object.entries(loadedTranslationsPerLanguage)) {
		for (const [key, translation] of
Object.entries(translations)) {
			if (loadedTranslations[key]) {
				loadedTranslations[key][lng] =
translation;
			} else {
				loadedTranslations[key] = {
					[lng]: translation,
				};
			}
		}
	}
	return loadedTranslations;
}
```

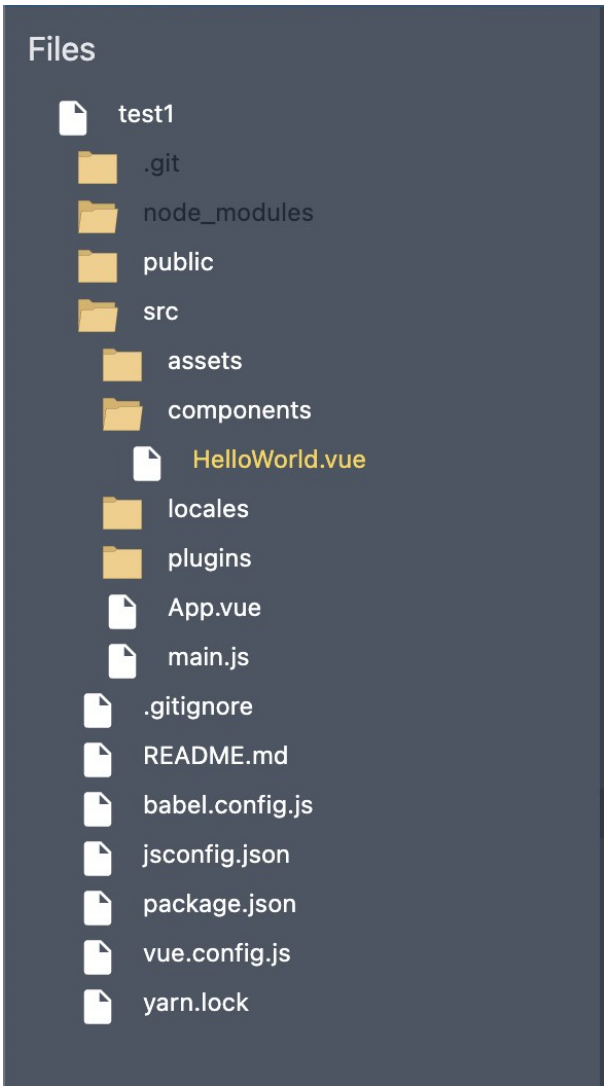
Kôd 4.16 Učitavanje prijevoda iz JSON datoteka

Kako prijevodi unutar JSON datoteka mogu biti različite dubine potrebno je sve staviti u jednu razinu tako da bi to postigli dodajemo znak za točku za svaku novu dubinu (Kôd 4.17).

```
export function flatten(data: any) {
	const result: { [key: string]: any } = {};
	function recurse(cur: { [key: string]: any }, prop:
string) {
		if (Object(cur) !== cur) {
			result[prop] = cur;
		} else if (Array.isArray(cur)) {
			for (let i = 0; i < cur.length; i++) {
```

```
                        recurse(cur[i], prop ? prop + '.' + i
: '' + i);
                  }
                  if (cur.length === 0) result[prop] = [];
            } else {
                  let isEmpty = true;
                  for (const p in cur) {
                        isEmpty = false;
                        recurse(cur[p], prop ? prop + '.' + p
: p);
                  }
                  if (isEmpty) result[prop] = {};
            }
      }
      recurse(data, '');
      return result;
}
```

Kôd 4.17 Prebacivanje svih prijevoda u istu razinu



Slika 4.5 Prikaz stabla datoteka i mapa učitanog korisnikovog projekta

Kod prikaza unutar uređivača, mape i datoteke se razlikuju po ikonama koja se nalaze uz njihov naziv (Slika 4.5). Radi lakšeg prikaza sve mape su zatvorene tako da korisnik vidi samo prvu razinu. Klikom na mapu ona se otvara te korisnik može vidjeti sve mape i datoteke unutar te mape. Kako bi se jasno raspoznala razlika između razina, otvaranje svake dublje razine dodaje razmak u desno. Isto tako mape koje su otvorene imaju drugačiju ikonu od zatvorenih. Klikom na datoteku ona se prikazuje žutom bojom kako bi korisnik mogao raspoznati otvorenu datoteku. Nakon što korisnik odabere prijevod u tablici sa prijevodima, unutar prikaza stabla pokraj datoteke se pojavljuje okrugla plava ikona koja označava da se unutar te datoteke nalazi prijevod.

4.4.2. Prikaz odabrane datoteke u uređivaču prijevoda

Klikom na određenu datoteku iz prikaza stabla datoteka, učitava se ta datoteka u memoriju te se prikazuje korisniku unutar uređivača prijevoda. Prilikom učitavanja pretražuju se prijevodi unutar datoteke i prilikom renderiranja sadržaja datoteke prijevodi se ističu posebnom bojom te ih je moguće selektirati.

Prilikom odabira datoteke u prikazu stabla prikazuje se *FileTree* komponenta koja prilikom inicijalizacije šalje događaj za učitavanje datoteke zajedno sa nazivom te datoteke (Kôd 4.18).

```
      useEffect(() => {
            window.electron.on('read-file-return', (event:
any, file: string) => {
                  setIsLoading(false);
                  processTranslationsInFile(file);
            });
            loadFileTree();
            return () => {
                  window.electron.removeAllListeners('read-
file-return');
            };
      }, [activeFile]);
. . .
      function loadFileTree() {
            if (window.electron && activeFile) {
                  window.electron.send('read-file', {
                        path: activeFile.path,
                  });
            }
      }
```

Kôd 4.18 Slanje i procesiranje odabrane datoteke prema Electronu

Electron dio zatim učitava tu datoteku te je šalje nazad u aplikaciju preko drugog događaja. (Kôd 4.19).

```
export function readFileContent(path: string): string {
      const rawData = fs.readFileSync(path);
      return rawData.toString();
}
. . .
```

```
function readFileHandler(event: Electron.IpcMainEvent, { path
}: { path: string }) {
      const loadedFile = readFileContent(path);
      event.sender.send('read-file-return', loadedFile);
}
```

Kôd 4.19 Učitavanje sadržaja datoteke

Zatim funkcija unutar *React.js* dijela koja sluša taj događaj prosljeđuje datoteku u funkciju koja zatim pronalazi sve prijevode unutar datoteke te zatim prolazi znak po znak te ovisno da li znak pripada prijevodu dodjeljuje određenoj varijabli. Algoritam (Kôd 4.20) radi na način da kada je idući na redu znak za novi red, kreira *JSX element* sa vrijednostima koje je do tada skupio ili ako je to zadnji znak za taj prijevod.

```
function processTranslationsInFile(file: string) {
      let vueI18nRegex = /\$t([^\)]+\))/g;
      let match;
      let foundTranslations = [];
      while ((match = vueI18nRegex.exec(file))) {
            foundTranslations.push({
                  value: match[0].substring(match[0].indexOf('"') +
                  1, match[0].lastIndexOf('")),
                  start: vueI18nRegex.lastIndex - match[0].length,
                  end: vueI18nRegex.lastIndex - 1,
            });
      }
      let translationRanges = foundTranslations.map((x) =>
[x.start, x.end, x.value]);

      let text = '';
      let translation = '';
      let activeTranslation: object | null = null;
      let row: any = [];
      let temp: any = [];
      file.split('').forEach((x, index) => {
            for (let i = 0; i < translationRanges.length; i++) {
                  if (x === '\n') {
                        if (text) {
                              row.push({
                                    type: 'text',
                                    text,
```

```
            });
        }
        if (translation) {
            row.push({
                type: 'translation',
                translation,
            });
            translation = '';
        }
        activeTranslation = null;
        temp.push(
        <div className=". . ." key={x + index
        + i}>
          {row.map((item: any) => {
            if (item.type === 'text') {
             return <div>{item.text}</div>;
            } else {
                return (
                <div className=". . ."
                onClick={(e) =>
            onTranslationClick({
            ...item.activeTranslation })}>
                {item.translation}
                </div> );
            }
          })
        }
        </div>
        );
        translation = '';
        text = '';
        row = [];
        return;
    } else if (
        index === translationRanges[i][0]) {
    activeTranslation = translationRanges[i];
    translation += x;
        if (text) {
            row.push({
                type: 'text',
```

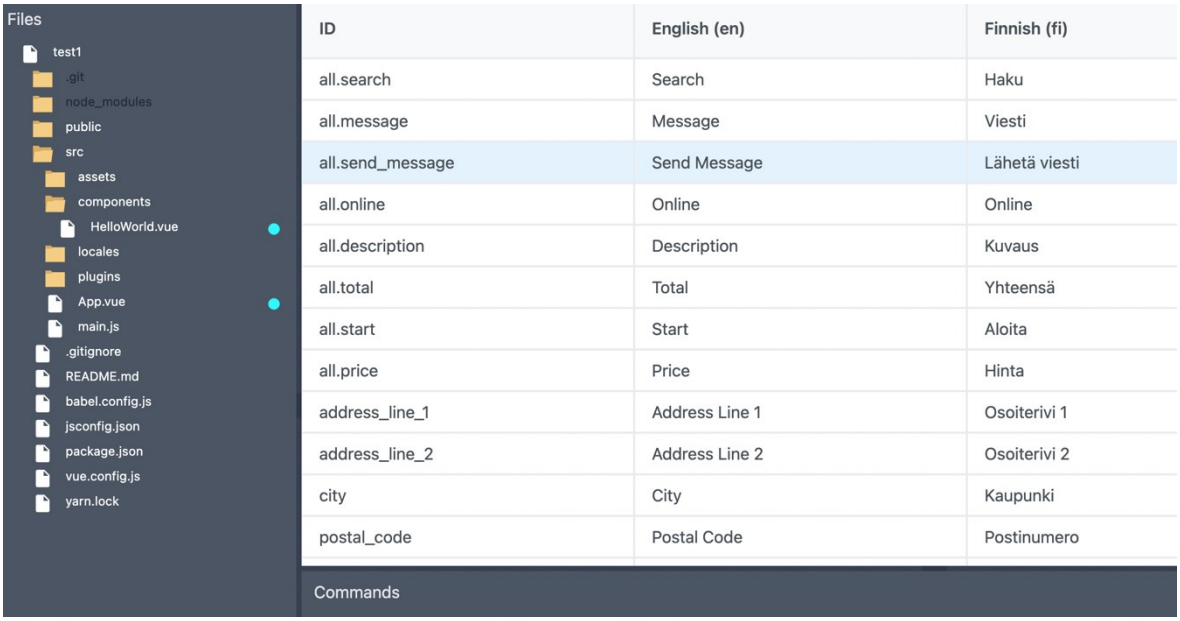
```
                                    text,
                              });
                        }
                        return;
                  } else if (
                        index === translationRanges[i][1]) {
                        translation += x;
                        row.push({
                              type: 'translation',
                              translation,
                              activeTranslation,
                        });
                        text = '';
                        translation = '';
                        return;
                  } else if (index > translationRanges[i][0]
                  && index < translationRanges[i][1]) {
                        translation += x;
                        return;
                  } else if (translationRanges.length - 1 ===
                  i) {
                        if (!translation) {
                              text += x;
                        } else {
                              translation = '';
                        }
                        return;
                  }
            }
      });
      setLines(temp);
}
```

Kôd 4.20 Priprema sadržaja datoteke za prikaz u uređivaču prijevoda

Svaki prijevod na sebi ima dodan događaj za klik prilikom kojega se taj prijevod označava u aplikaciji te se na njemu mogu izvršavati promjene.

4.5. Prikaz svih upotreba za određeni identifikator

Klikom na red unutar tablice sa prijevodima (Slika 4.6), korisniku se unutar stabla datoteka i mapa prikazuju pronađena korištenja za odabrani prijevod pomoću plave ikone.



Slika 4.6 Prikaz tablice sa prijevodima

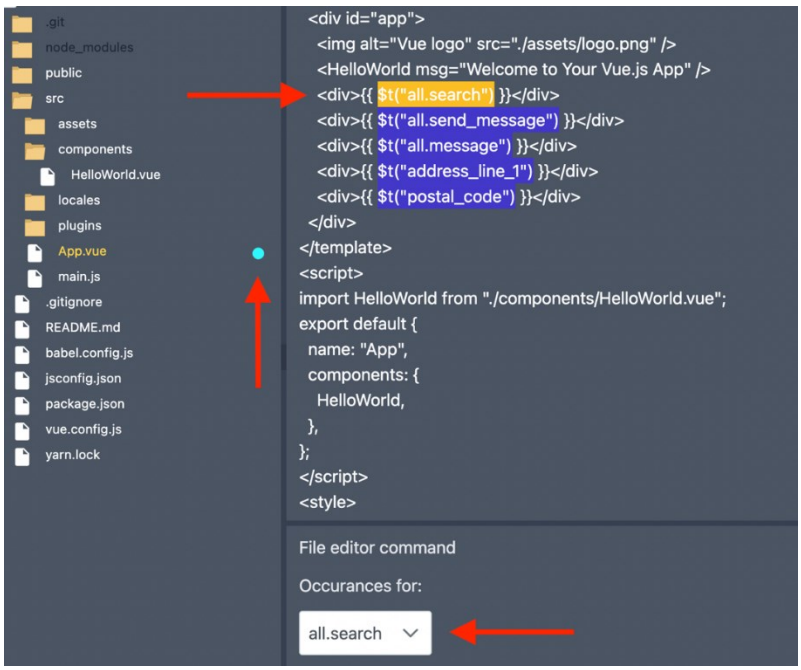
Korisnik zatim može kliknuti na datoteku sa tim prijevodom te raditi izmjene (Slika 4.7). Kako bi se postiglo pretraživanje određenog prijevoda aplikacija prolazi po svim prijevodima za datoteke koje su spremljene u memoriju te vraća listu prijevoda sa datotekama koje sadrže odabrani prijevod. Funkciji se prosljeđuje objekt po referenci (engl. *pass by reference*[9]) [5] koji se zatim ispunjava pronađenim prijevodima. Koristi se rekurzija koja ide po već učitanom stablu svih datoteka koje unutar sebe već imaju pronađene prijevode. Nakon što se pronađe prijevod, njegov identifikator se nadodaje kao ključ proslijeđenom objektu te se kao vrijednost dodaje u listu objekt koji sadrži ime datoteke te početnu i završnu poziciju prijevoda unutar te datoteke (Kôd 4.21).

```
setFileTranslations(state, action: PayloadAction<TreeNode>) {
      let translations: {} = {};
      readAllTranslationsFromTree(action.payload, translations);
      state.fileTranslations = translations;
}
```

[9] *Pass by reference* je mehanizam u JavaScriptu prema kojemu su objekti i funkcije proslijeđeni direktno pokazujući na taj objekt za razliku od primitiva koji su *Pass by value*.

```
. . .
function readAllTranslationsFromTree(node: TreeNode, acumulator: {}): {}
{
      if (node.children.length > 0) {
            for (let i = 0; i < node.children.length; i++) {
                  readAllTranslationsFromTree(node.children[i],
                  acumulator);
            }
      } else {
            if (node.translations) {
                  for (let i = 0; i < node.translations.length; i++) {
                        const tx = node.translations[i];
                        const id = tx.name;
                        if (acumulator.hasOwnProperty()) {
                              acumulator[id].push({ fileName: node.name,
                              start: tx.start, end: tx.end });
                        } else {
                              acumulator[id] = [{fileName: node.name,
                              start: tx.start, end: tx.end }];
                        }
            }}
            return acumulator;
}}
```

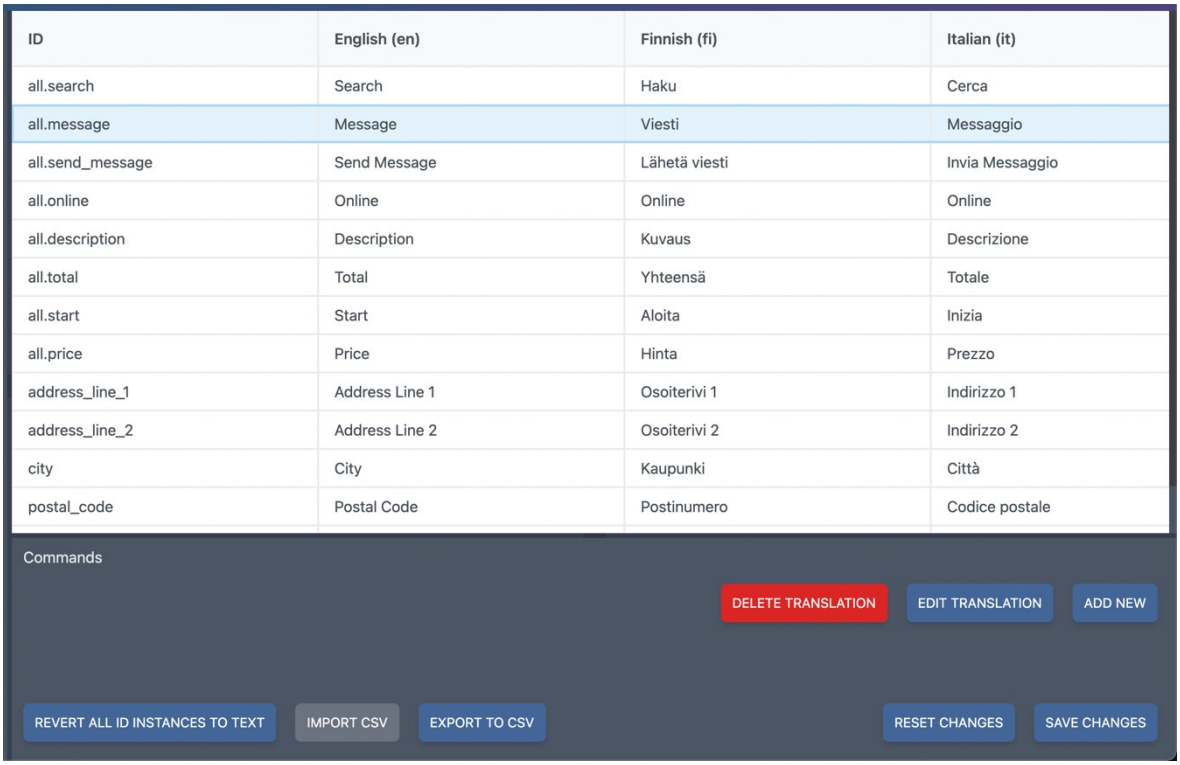
Kôd 4.21 Postavljanje pronađenih prijevoda unutar datoteka u store



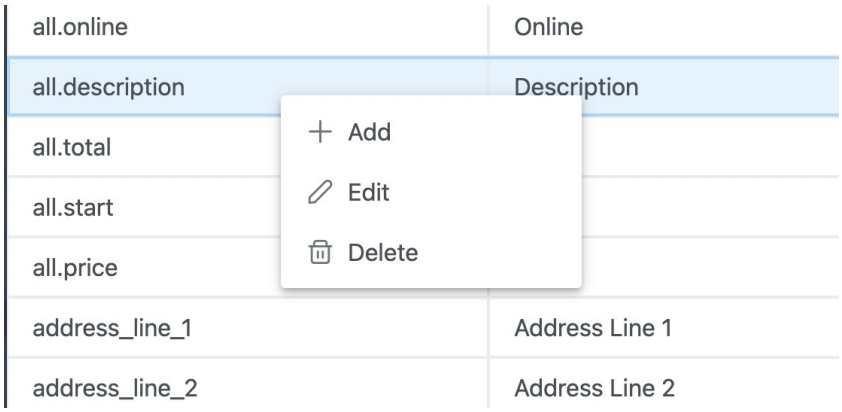
Slika 4.7 Prikaz pronađenih prijevoda unutar datoteke

4.6. Upravljanje prijevodima

Prijevodima se upravlja pomoću tablice za prijevode te dijela sa njezinim komandama (Slika 4.8). Svaki red unutar tablice predstavlja jedan prijevod. Prvi stupac predstavlja identifikator prijevoda dok ostali predstavljaju prijevode za jezike koje je korisnik definirao. Identifikator prijevoda mora biti u određenom formatu kako bi bio valjan. Identifikator ne smije sadržavati praznine, posebne znakove osim crte i donje crte te se sa točkom može označiti razina u koju želimo spremiti prijevod.



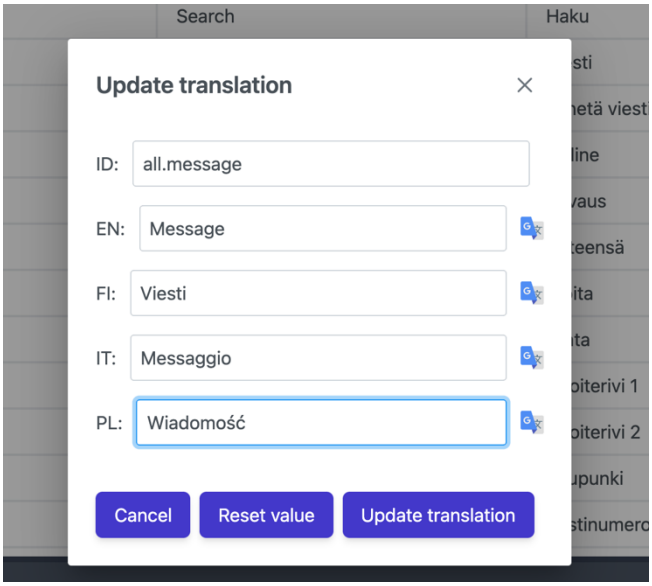
Slika 4.8 Tablica za prijevode



Slika 4.9 Kontekstni izbornik iznad tablice

Upravljanje identifikatora se može izvršavati tako da korisnik klikne na gumb u komandnom dijelu (Slika 4.9) duplim klikom te pomoću kontekstnog izbornika koji se otvara desnim klikom na odabrani prijevod. Kontekstni izbornik se sastoji od komandi za dodavanje, osvježavanje te brisanje prijevoda.

Odabirom komande za dodavanje ili osvježavanje otvara se dijalog sa prijevodima (Slika 4.10). Dijalog se sastoji od polja za unos identifikatora, prijevoda definiranih jezika, gumba za automatsko prevođenje, gumba za zatvaranje, gumba za resetiranje promjena te gumba za spremanje svih promjena za odabrani prijevod. Također moguće je klikom na red odabrati prijevod te zatim klikom na određenim gumbom u komandnom dijelu pristupiti dijalogu sa prijevodima za taj odabrani prijevod.



Slika 4.10 Dijalog za editiranje prijevoda

Klikom na gumb „delete“ ili komandu „delete“ u kontekstnom izborniku prijevod se briše iz liste prijevoda. Također ukoliko želimo poništiti promjene postoji gumb za resetiranje.

4.7. Spremanje svih promjena

Unutar komandnog dijela tablice sa prijevodima nalazi se gumbi za resetiranje svih promjena te gumb za spremanje svih promjena na tablici. Klikom na gumb za spremanje poziva se akcija definirana u *Redux store-u*[10] (Kôd 4.22) koja zatim uzima podatke prijevoda na kojoj su rađene promjene u tablici sa prijevodima te ih prilagođava u format pogodan za kasniju obradu i spremanje. Nakon pripreme prijevoda u novom obliku, podaci se šalju putem događaja u *Electron* dio aplikacije

```
            saveTranslations(state, action: PayloadAction<string>) {
                  const translations = {};
                  for (let i = 0; i < state.translationData.length; i++)
{
                        const tx = state.translationData[i] as
Translation;
                        let temp = {};
                        let id = '';
                        for (const [key, value] of Object.entries(tx)) {
                              if (key === 'id') {
                                    id = value as unknown as string;
                              } else {
                                    temp[key] = value;
                              }
                        }
                        translations[id] = temp;
                  }
                  state.translations = translations;
                  electronSaveTranslations(action.payload, translations);
            }
            . . .
            const electronSaveTranslations = (url: string, txs:
            Translation) => {
                  if (window?.electron?.send) {
                        window.electron.send('write-translation-files', {
            url, translations: txs });
                  }
```

[10] Redux store je mjesto na kojem živi trenutno stanje aplikacije.

```
};
```

Kôd 4.22 Priprema prijevoda za spremanje unutar store

Electron dio aplikacije zatim prima događaj zajedno sa prijevodima i putanjom koju ćemo koristiti za spremanje prijevoda. Prijevodi se razvrstavaju po jeziku kako će na kraju biti i zapisani u datoteke (Kôd 4.23). Prije zapisivanja prijevoda u datoteke moramo prijevode staviti na razini na kojoj su i prije bili tako da za to koristimo funkciju `unflatten()` (Kôd 4.24).

```
export const writeTranslationsListener = ipcMain.on(
      'write-translation-files',
      writeTranslationsHandler
);
. . .
function writeTranslationsHandler(
      event: Electron.IpcMainEvent,
      { url, translations }: { url: string; translations:
Translation }
) {
      writeTranslations(url, translations);
}
. . .
export function writeTranslations(url: string, translations:
Translation) {
      let fileTranslations = {};
      for (const [id, txs] of Object.entries(translations)) {
            for (const [lang, translation] of Object.entries(txs))
{
                  if (!fileTranslations[lang]) {
                        fileTranslations[lang] = {};
                  } else {
                        fileTranslations[lang][id] = translation;
                  }
            }
      }
for (const [langName, txs] of Object.entries(fileTranslations)) {
            const unflattenTranslations = unflatten(txs);
            const fileName = path.join(url, langName + '.json');
            fs.writeFileSync(fileName,
JSON.stringify(unflattenTranslations, null, 2));
```

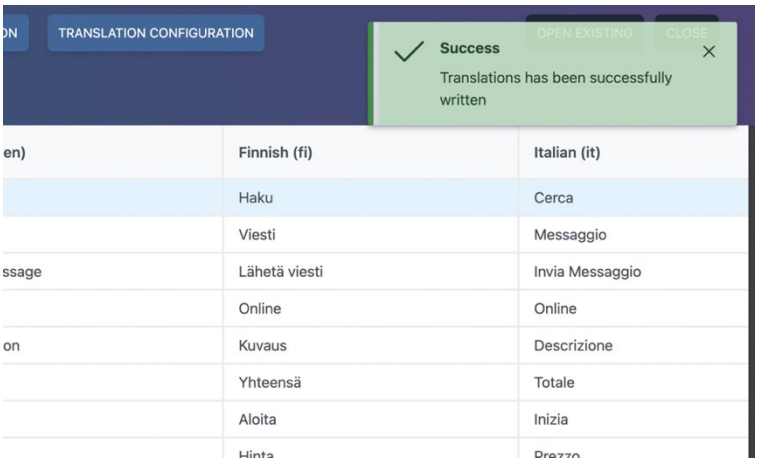
```
	}
}
```

Kôd 4.23 Spremanje prijevoda u zasebne datoteke za svaki jezik

```
export function unflatten(data: { [x: string]: any }) {
	if (Object(data) !== data || Array.isArray(data)) return
data;
	const result: { [key: string]: any } = {};
	for (const p in data) {
		let cur = result;
		let prop = '';
		const parts = p.split('.');
		for (let i = 0; i < parts.length; i++) {
			const idx = !isNaN(parseInt(parts[i]));
			cur = cur[prop] || (cur[prop] = idx ? [] : {});
			prop = parts[i];
		}
		cur[prop] = data[p];
	}
	return result[''];
}
```

Kôd 4.24 Vraćanje prijevoda u izvorno definiranu razinu

Nakon što su datoteke uspješno zapisane korisniku se prikazuje u gornjem desnom kutu notifikacija sa porukom o uspješnom spremanju prijevoda (Slika 4.11) (Kôd 4.25).



Slika 4.11 Prikaz uspješne poruke za spremanje prijevoda

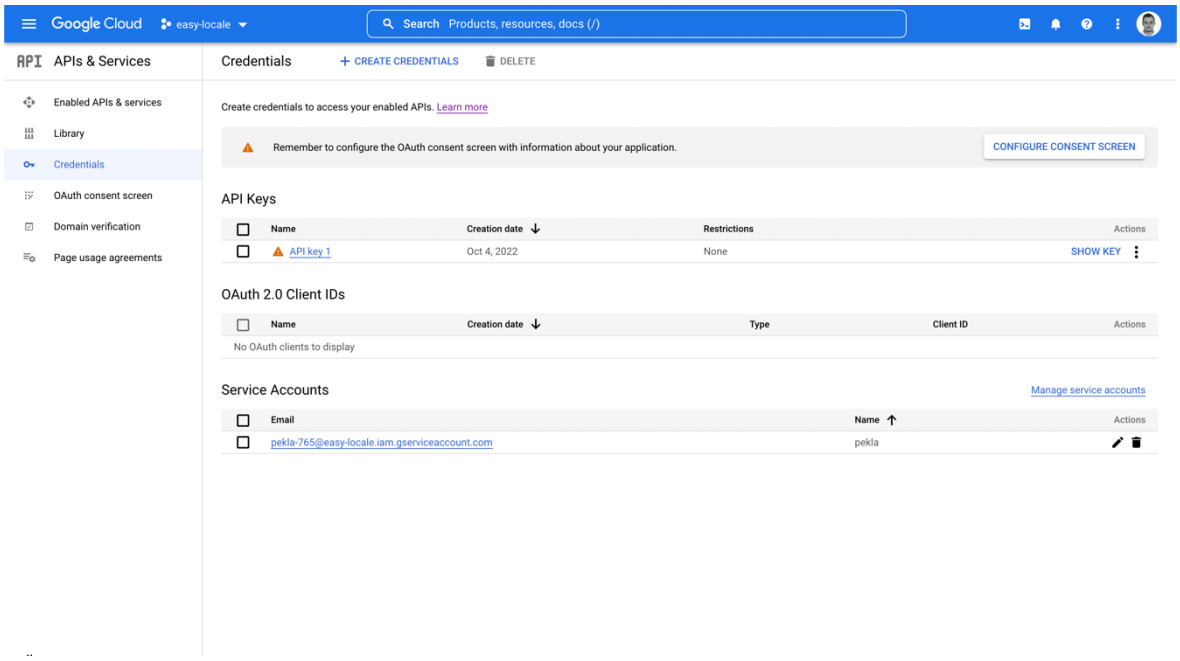
```
	import { Toast } from 'primereact/toast';
	const toast = useRef() as MutableRefObject<Toast>;
	. . .
```

```
toast.current.show({
                    severity: 'success',
                    summary: 'Success',
                    detail: 'Translations have been
successfully written',
                    life: 3000,
});
. . .
<Toast ref={toast} />
```

Kôd 4.25 Uspješna poruka za spremanje prijevoda

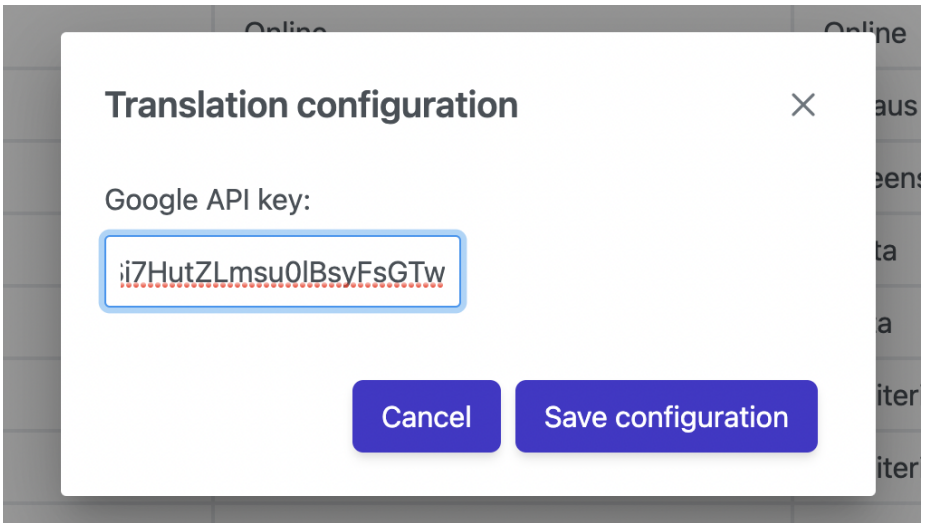
4.8. Prevođenje pomoću Google Translate API

Kako bi korisniku bilo olakšano prevođenje tekstova kako ne bi morao svaki puta ručno prevesti tekst za određeni jezik dodana je podrška za **Google Translate API.** Kako bi bilo moguće koristiti *Google Translate API* potrebno je napraviti *Google Cloud* korisnički račun, napraviti projekt unutar *Google Clouda* [8], omogućiti korištenje *Translate API* servisa unutar projekta [9], dodati API[11] ključ (Slika 4.12) u aplikaciju te dodati *Google Translate NPM paket* u projekt.



Slika 4.12 Google Cloud konzola za dodavanje API ključeva

[11] API je programsko sučelje koje sadrži skup pravila i specifikacija koja omogućuju programerima korištenje tih servisa.



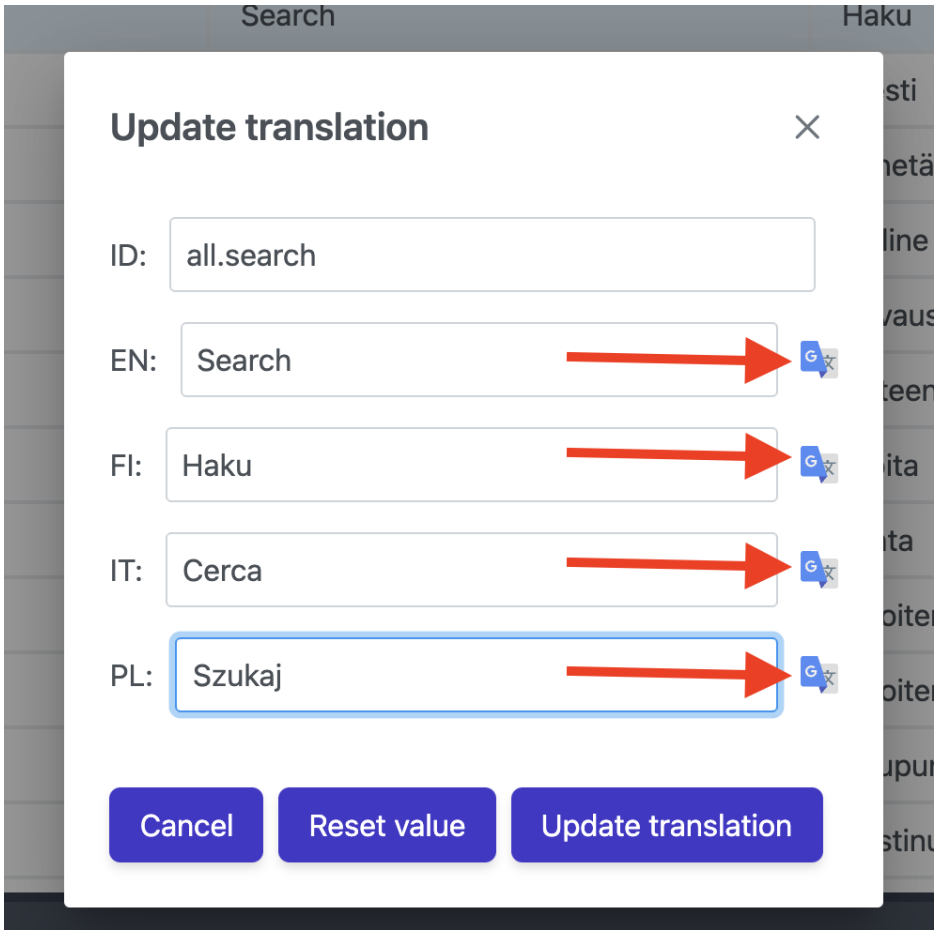
Slika 4.13 Forma za dodavanje API ključa u aplikaciji

Kako bi mogli koristiti *Google translate API* očekuje se da korisnik podesi svoj *Google API key* unutar aplikacije (Slika 4.13). To je moguće tako da korisnik klikne u info traci na gumb za konfiguraciju prijevoda nakon čega se pojavljuje dijalog sa poljem za unos ključa.

Postoji više verzija koje postoje sa više vrsta složenosti koje omogućuju. Za potrebe ovog projekta se koristi osnovna v2 verzija, ali postoji još i napredna verzija te v3 verzija koja je *beta verzija* tako da nije stabilna za korištenje. Kako bi korisnik koristio automatsko prevođenje potrebno je odabrati prijevod te otvoriti dijalog za editiranje prijevoda. Pokraj svakog polja za određeni jezik nalazi se gumb sa *Google Translate* ikonom (Kôd 4.26).

```
<img
  className="h-5 my-auto cursor-pointer"
  alt=""
  src={googleTranslateIcon}
  onClick={
  (e) => onTranslateClickHandler(item.name, field.name)
}/>
```

Kôd 4.26 Ikona za korištenje Google Translate API-a



Slika 4.14 Prikaz gumba za prevođenje unutar forme za editiranje prijevoda

Kako bi se moglo izvršavati prevođenje potrebno je imati unesen prijevod za zadani jezik. Klikom na gumb pokraj polja određenog jezika uzima se tekst zadanog jezika za taj prijevod te se izvršava kod koji radi poziv prema *Translate API* servisu gdje se šalje taj tekst, jezik iz kojega se prevodi te jezik u koji se prevodi (Slika 4.14) (Kôd 4.28).

Funkcija koja se izvršava klikom na gumb za prijevode (Kôd 4.27):

```
const apiKey = useAppSelector(
      (state) => state.settings.googleApiKey
);
. . .
async function onTranslateClickHandler(target:  any,
                        translationArrayId: string) {
      const translations = getValues('translation');
      const translation = translations.find(
            (x: { name: string }) => x.name === defaultLanguage
```

```
            ).value;
      setActiveTranslation(translationArrayId);
      if (window.electron) {
            window.electron.send('get-translation', {
                  text: translation,
                  source: defaultLanguage,
                  target: target,
                  key: apiKey,
            });
      }
}
```

Kôd 4.27 Funkcija za slanje događaja za prevođenje prema Electronu

```
import { Translate } from '@google-
cloud/translate/build/src/v2/index';
async function getTranslationHandler(
      event: Electron.IpcMainEvent,
      { text, source, target, key }: { text: string; source:
string; target: string; key: string; }
) {
      const translate = new Translate({ key });
      const options = {
            from: source,
            to: target,
      };
      let [translation] = await translate.translate(text, options);
      event.sender.send('get-translation-return', translation);
}

export const getTranslationListener = ipcMain.on('get-translation',
getTranslationHandler);
```

Kôd 4.28 Funkcija za prevođenje u nove jezike unutar Electrona

Nakon što je došao odgovor sa servisa, tekst koji sadrži polje pokraj kojeg je gumb kliknut osvježava se sa tekstom kojega smo dobili sa servisa. Ukoliko nismo zadovoljni sa promjenama onda možemo zatvoriti dijalog te se promjene neće spremiti. Ukoliko smo

zadovoljni promjenama onda možemo kliknuti na gumb za osvježavanje prijevoda (Kôd 4.29).

Postavljanje vrijednosti u prijevodu na dohvaćeni tekst:

```
      useEffect(() => {
            if (window.electron) {
                  window.electron.on('get-translation-return',
(event: any, translation: any) => {
                        setValue(activeTranslation, translation);
                  });
            }
            return () => {
                  window.electron.removeAllListeners('get-
translation-return');
            };
      }, [activeTranslation]);
```

Kôd 4.29 Postavljanje novog prijevoda unutar React komponente

4.9. Izvoz u CSV formatu

Za izvoz u CSV formatu koristimo funkcionalnost koja je već implementirana na *DataTable* komponenti (Kôd 4.30) od *PrimeReact* biblioteke (engl. *library*) [4]. Kako bi je koristili potrebno je uvesti `useRef()` React funkciju, promjenjivi tip za referencu (*MutableRefObject*) te je definirati.

Zatim tu referencu trebamo dodijeliti unutar komponente tako da je dodijelimo svojstvu naziva *ref*.

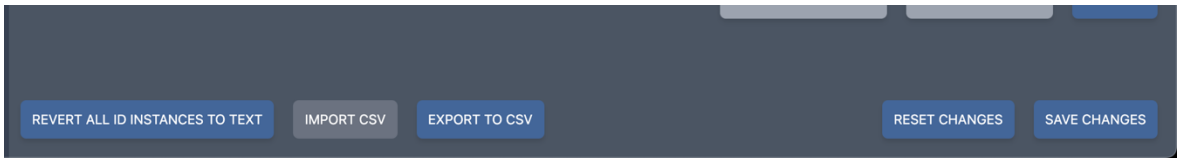
```
import { useRef, MutableRefObject } from 'react';
import { DataTable } from 'primereact/datatable';
const translationTableRef = useRef() as MutableRefObject<DataTable>;


<DataTable
      ref={translationTableRef}
      style={{ width: '100%', height: '100%', overflow: 'scroll' }}
      value={translationData}
      selectionMode="single"
      selection={selectedTranslation}
      onContextMenuSelectionChange={(e)=>dispatch(setSelectedTransl
      ation(e.value))}
      onContextMenu={(e) => cm.current.show(e.originalEvent)}
      onRowDoubleClick={(e)=>{dispatch(setTranslationDialogType(
            TranslationDialogEnum.edit));
            dispatch(setShowEditDialog(true));}}
onSelectionChange={(e)=>dispatch(setSelectedTranslation(e.value))}
      className="mt-0"
      resizableColumns
      scrollable
      scrollHeight="600px"
      virtualScrollerOptions={{ itemSize: 20 }}
      columnResizeMode="expand"
```

```
            showGridlines
            reorderableColumns>
            <Column style={{ width: '320px' }} field="id" header="ID" />
            {renderColumns}
        </DataTable>
```

Kôd 4.30 Tablica za prijevode

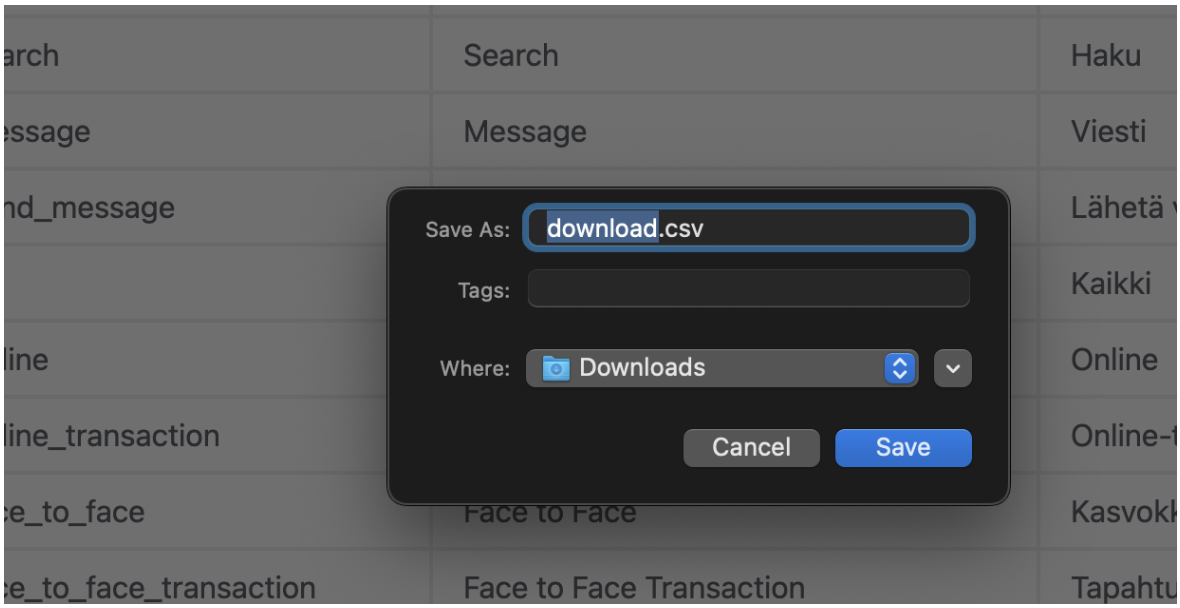
Nakon toga klikom na gumb „Export to CSV“ pokreće se metoda na tablici za izvoz u CSV formatu (Slika 4.15).

```
translationTableRef?.current.exportCSV();
```



Slika 4.15 Traka sa naredbama za tablicu unutar koje se nalaze prijevodi

Korisniku se zatim prikazuje dijalog sa odabirom naziva datoteke te mjesta gdje je želi spremiti (Slika 4.16). Unutar datoteke vrijednosti su razdijeljene zarezom.



Slika 4.16 Izgled prikazanog dijaloga tijekom izvoza u CSV formatu

Zaključak

Radom na ovoj aplikaciji došao sam do nekoliko spoznaja. Neadekvatno strukturiran kôd kompleksnih programskih rješenja uvelike otežava nadogradnju i održavanje aplikacije.

Rad na aplikaciji je tekao kroz veći vremenski period tijekom kojega je dolazilo do velikih promjena u mojemu znanju, načinu razmišljanja i pristupu. Veća greška na početku rada na projektu, a koju sam tek kasnije primijetio je bilo fokusiranje na detalje te dizajn aplikacije umjesto na samu srž problema koji aplikacija rješava. Kasnije sam napustio takav pristup te se okrenuo prema rješavanju srži problema, a tek onda sporednih stvari. Nove funkcionalnosti koje je moguće implementirati te integracija sa drugim tehnologijama ostavljaju puno prostora za daljnji rad na aplikaciji.

Na tržištu trenutno stanje se tijekom izrade ovoga rada drastično mijenjalo. Pojavila su se nova rješenja u obliku programa kao usluga (engl. *Software as a Service*, skraćeno SaaS[12]) koja su se bavila problemom prijevoda. Međutim razlika između ove aplikacije i njih je način i pristup na koji oni rješavaju problem. Ova aplikacija rješava problem iz ugla programera čime joj je najvažnije korisničko iskustvo programera (engl. *Developer Experience*, skraćeno DX[13]) dok rješenja na tržištu pristupaju iz ugla prevoditelja te korisnika. Za dobar proizvod vjerojatno je potrebno oboje te sigurno postoji potreba i prostor za korištenje ovakvog tipa aplikacije.

[12] Software as a Service, SaaS je model kojima se pruža usluga koja se nalazi u oblaku. Isporučuje se preko interneta te je dostupna preko interneta na svim uređajima.
[13] Developer Experience, DX je korisničko iskustvo programera tijekom razvoja programa.

Popis kratica

API	*Application Programming Interface*	internacionalizacija
CRA	*Create React App*	alat za postavljanje React aplikacija
CSV	*Comma separated values*	vrijednosti odvojene zarezom
CSS	*Cascading Style Sheets*	kaskadni stilski listovi
DX	Developer Experience	iskustvo programera
DOM	*Document Object Model*	objektni model dokumenta
HTML	*HyperText Markup Language*	tekstualni označni jezik
I18N	*Internationalization*	internacionalizacija
IPC	*Inter-process communication*	inter procesna komunikacija
JSON	*JavaScript Object Notation*	JavaScript objektna notacija
JSX	*JavaScript XML*	sintaksa za opisivanje korisničkih sučelja
L10N	*Localization*	lokalizacija
NPM	Node Package Manager	upravitelj JavaScript paketa
UI	*User interface*	korisničko sučelje
URL	*Uniform Resource Locator*	jedinstveni lokator resursa
SaaS	Software as a Service	program kao usluga
XML	*Extensible Markup Language*	jezik za proširivanje

Popis slika

Slika 3.1 Prikaz komunikacije između Node.js i React.js dijela u aplikaciji 5

Slika 3.2 Podjela koda unutar aplikacije 9

Slika 4.1 Prikaz početnog ekrana aplikacije 14

Slika 4.2 Prikaz različitih sekcija editora 17

Slika 4.3 Unos općih informacija o projektu 29

Slika 4.4 Unos jezika koje korisnik želi podržavati u projektu 29

Slika 4.5 Prikaz stabla datoteka i mapa učitanog korisnikovog projekta 43

Slika 4.6 Prikaz tablice sa prijevodima 48

Slika 4.7 Prikaz pronađenih prijevoda unutar datoteke 49

Slika 4.8 Tablica za prijevode 50

Slika 4.9 Kontekstni izbornik iznad tablice 51

Slika 4.10 Dijalog za editiranje prijevoda 51

Slika 4.11 Prikaz uspješne poruke za spremanje prijevoda 54

Slika 4.12 Google Cloud konzola za dodavanje API ključeva 55

Slika 4.13 Forma za dodavanje API ključa u aplikaciji 56

Slika 4.14 Prikaz gumba za prevođenje unutar forme za editiranje prijevoda 57

Slika 4.15 Traka sa naredbama za tablicu unutar koje se nalaze prijevodi 61

Slika 4.16 Izgled prikazanog dijaloga tijekom izvoza u CSV formatu 61

Popis kodova

Kôd 3.1 Uvezeni dijelovi korišteni unutar glavne Electron datoteke 6

Kôd 3.2 Definiranje varijabli unutar glavne Electron datoteke 6

Kôd 3.3 Funkcija za kreiranje prozora 7

Kôd 3.4 Postavljanje prozora kada je aplikacija spremna 7

Kôd 3.5 Konfiguracija prozora 8

Kôd 3.6 Omogućavanje metoda unutar renderer procesa 10

Kôd 3.7 Slanje događaja za postavljanje prozora 11

Kôd 3.8 Primanje događaja unutar React.js dijela 11

Kôd 4.1 Organizacija React komponenti u projektu 13

Kôd 4.2 Komponenta početnog ekrana 16

Kôd 4.3 Komponenta za upravljanje projektima 17

Kôd 4.4 Editor komponenta 20

Kôd 4.5 Komponenta za dodavanje novog projekta 32

Kôd 4.6 Postavljanje inicijalnih vrijednosti koristeći React Hooks form 33

Kôd 4.7 Način korištenja polja za unos koristeći React Hoks form 33

Kôd 4.8 Dodavanje jezika unutar forme za dodavanje projekta 35

Kôd 4.9 Funkcija za dodavanje novog jezika unutar forme 35

Kôd 4.10 Procesiranje dodavanja novog projekta 36

Kôd 4.11 Slanje događaja za dodavanje novog projekta u Electron 37

Kôd 4.12 Spremanje novog projekta u korisničke postavke 37

Kôd 4.13 Slanje događaja za učitavanje svih informacija o datotekama i mapama projekta 38

Kôd 4.14 Slušajuća funkcija za učitavanje datoteka i mapa 38

Kôd 4.15 Rekurzivno učitavanje informacija o svim datotekama i mapama 40

Kôd 4.16 Učitavanje prijevoda iz JSON datoteka 41
Kôd 4.17 Prebacivanje svih prijevoda u istu razinu 42
Kôd 4.18 Slanje i procesiranje odabrane datoteke prema Electronu 44
Kôd 4.19 Učitavanje sadržaja datoteke 45
Kôd 4.20 Priprema sadržaja datoteke za prikaz u uređivaču prijevoda 47
Kôd 4.21 Postavljanje pronađenih prijevoda unutar datoteka u store 49
Kôd 4.22 Priprema prijevoda za spremanje unutar store 53
Kôd 4.23 Spremanje prijevoda u zasebne datoteke za svaki jezik 54
Kôd 4.24 Vraćanje prijevoda u izvorno definiranu razinu 54
Kôd 4.25 Uspješna poruka za spremanje prijevoda 55
Kôd 4.26 Ikona za korištenje Google Translate API-a 56
Kôd 4.27 Funkcija za slanje događaja za prevođenje prema Electronu 58
Kôd 4.28 Funkcija za prevođenje u nove jezike unutar Electrona 58
Kôd 4.29 Postavljanje novog prijevoda unutar React komponente 59
Kôd 4.30 Tablica za prijevode 61

Literatura

[1] Steven Kinney; Electron in Action (2018); Manning; ISBN-13: 978-1617294143

[2] Alex Banks, Eve Porcello; Learning React: Modern patterns for developing React Apps (2020); O'Reilly Media; ISBN-13:978-1492051725

[3] https://www.electronjs.org/docs/latest/tutorial/ipc, dokumentacija sa opsiom komuniciranja unutar Electron aplikacije, 7.04.2022

[4] https://www.primefaces.org/primereact/setup/, dokumentacija za korištenje *PrimeReact* komponenti, 12.05.2022

[5] Marijn Haverbeke; Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming (2018); No Starch press;ISBN-13: 978-1593279509

[6] https://kazupon.github.io/vue-i18n/guide/formatting.html - formatting, dokumentacija sa opisom korištenja prijevoda unutar Vue aplikacije, 10.04.2022

[7] https://react-hook-form.com/api/useform, dokumentacija za korištenje React Hooks Form, 15.04.2022

[8] https://cloud.google.com/resource-manager/docs/creating-managing-projects, upute za postavljanje Google Cloud projekta, 16.09.2022

[9] https://cloud.google.com/translate/docs/setup - creating_service_accounts_and_keys, upute za kreiranje API ključa unutar projekta, 16.09.2022

[10] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction, objašnjenje pojma, 17.11.2022