Razvoj uređivača Markdown datoteka

Fumić, Roko

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

ROKO FUMIĆ

RAZVOJ UREĐIVAČA MARKDOWN DATOTEKA

Završni rad

Pula, rujan, 2022. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

ROKO FUMIĆ

RAZVOJ UREĐIVAČA MARKDOWN DATOTEKA

Završni rad

JMBAG: 0303092348, redoviti student

Studijski smjer: Informatika

Predmet: Programiranje

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan, 2022. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Roko Fumić**, kandidat za prvostupnika **informatike** ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

U Puli, 18.09.2022. godine

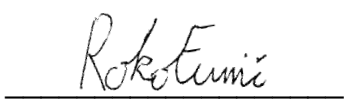
IZJAVA

o korištenju autorskog djela

Ja, **Roko Fumić** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom **Razvoj uređivača Markdown datoteka** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 18.09.2022. godine

Potpis



SAŽETAK

U ovom završnom radu je opisan proces izrade računalnog programa za stvaranje i uređivanje Markdown datoteka, pod nazivom RemisMD. Aplikacija je izrađena u potpunosti koristeći web tehnologije HTML, CSS i JavaScript uz pomoć Electron razvojnog okvira i Vue.js za izradu sučelja. Markdown dokumenti otvoreni u aplikaciji prikazuju se formatirani, a korisnik uređuje svaku liniju datoteke posebno, u izvornom Markdown obliku. Uz to, boje sučelja se mogu prilagođavati definiranjem vlastitih tema. Završni rad se sastoji od opisa funkcionalnosti aplikacije, analize i usporedbe sličnih programskih rješenja, opisa korištenih tehnologija te detaljnog objašnjenja implementacije.

Ključne riječi: Markdown, uređivač, JavaScript, Electron, Vue.js

ABSTRACT

This bachelor's thesis describes the process of developing a desktop program for creating and editing Markdown files, by the name of RemisMD. The application is built entirely using the HTML, CSS and JavaScript web technologies with the help of the Electron framework and Vue.js for creating the interface. Markdown documents opened in the app are shown formatted and the user edits each line of the file separately, in plaintext Markdown. Along with that, the colors of the user interface are customizable by defining themes. The bachelor's thesis consists of a description of the program's functionalities, an analysis and comparison of similar applications, a description of technologies used and a detailed explanation of the implementation.

Keywords: Markdown, editor, JavaScript, Electron, Vue.js

SADRŽAJ

1. UVOD

Markdown je, prema riječima jednog od njegovih kreatora Johna Grubera (Gruber, 2004), alat za pretvorbu običnog teksta u HTML, namijenjen web piscima. Omogućuje pisanje čistog teksta, jednostavnog za pisanje i čitanje, koji se pretvara u strukturirani XHTML ili HTML format. Prema tome, Markdown predstavlja sintaksu za formatiranje običnog teksta, ali i softverski alat koji pretvara taj formatirani tekst u HTML.

Glavni cilj kod razvoja Markdown sintakse bila je čitljivost. Dokument napisan Markdown sintaksom u svom običnom, izvornom obliku mora biti čitljiv sam po sebi, bez složenih sintaktičkih oznaka i bez potrebe za pretvorbu u formatirani oblik. Međutim. svrha Markdown-a nije zamijeniti HTML, već olakšati čitanje, pisanje i uređivanje običnog teksta. HTML je format za objavljivanje, a Markdown za pisanje. Stoga, Markdown sintaksa rješava isključivo probleme koji se tiču teksta.

Uz povećanje popularnosti Markdown formata, rastao je i broj njegovih implementacija koje su ga proširivale, što je, zbog manjka službene specifikacije, dovelo do neslaganja u tumačenju njegove sintakse. Zbog toga je 2014. došlo do objave CommonMark specifikacije čija je svrha bila jasno definirati Markdown format.

Danas se Markdown redovno koristi za bloganje, članke na portalima, online dokumentaciju, bilješke, objave na forumima te za pisanje “readme” dokumenata u repozitorijima koda. Iako je Markdown moguće pisati u običnim uređivačima teksta, s vremenom se pojavilo mnoštvo aplikacija napravljenih specifično za rad s Markdown datotekama, koje obično koriste nastavke .md ili .markdown.

Cilj ovog rada je razviti desktop aplikaciju za stvaranje i uređivanje Markdown dokumenata, pod nazivom RemisMD. Za razliku od mnogih ostalih rješenja, ova aplikacija korisniku prikazuje Markdown datoteku u formatiranom obliku, ali mu omogućuje uređivanje svake linije dokumenta posebno, tako da se linija koja se uređuje prikaže u “sirovom” Markdown-u, a ostale u formatiranom obliku. Uz to, korisnik ima mogućnost prilagodbe boja sučelja izradom vlastitih tema unutar jednostavnog JSON dokumenta. Također, RemisMD podržava GitHub Flavored

Markdown, proširenu verziju CommonMark specifikacije koja dodaje jednostavno stvaranje tablica, precrtanih riječi te automatskih poveznica.

Ovaj rad je podijeljen u pet dodatnih poglavlja. Započinje s poglavljem o analizi i usporedbi postojećih uređivača Markdown datoteka te po čemu se razlikuju od izrađene aplikacije. Zatim slijedi poglavlje u kojem se opisuju funkcionalnosti RemisMD-a i izlažu osnovne upute kako ga koristiti. Nakon toga se izlažu i objašnjavaju razvojne tehnologije koje su se koristile pri stvaranju programskog proizvoda. Potom se detaljno opisuje struktura programskog koda i njegova implementacija. Na kraju se nalazi diskusija o izrađenom proizvodu, korištenim tehnologijama te mogućnostima proširenja.

Aplikacije je dostupna za preuzimanje, zajedno sa svojim izvornim kodom, na GitHub-u: https://github.com/rfumic/RemisMD.

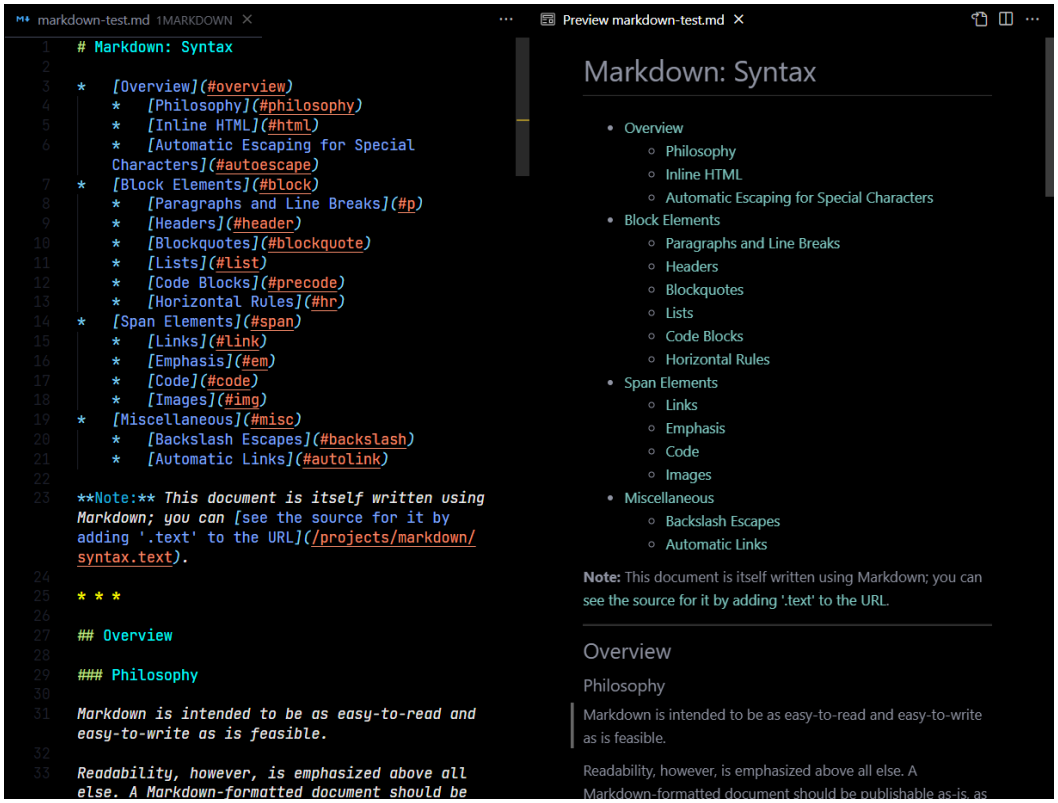
2. ANALIZA POSTOJEĆIH PROGRAMSKIH RIJEŠENJA

U ovom poglavlju analizirane su tri aplikacije za uređivanje Markdown datoteka. Iako postoji mnoštvo takvih aplikacija, ove tri su odabrane zato što su popularne, besplatne i otvorenog koda.

2.1. Visual Studio Code

Visual Studio Code, ili skraćeno VS Code, je besplatni uređivač otvorenog koda, razvijen i održavan od strane Microsoft-a te izgrađen koristeći Electron okvir s TypeScript jezikom (Visual Studio Code, 2022). Prema Stack Overflow anketi za 2022. godinu, najkorišteniji je alat za uređivanje koda (Stack Overlfow Developer Survey, 2022). Iako je dizajniran kao općeniti uređivač teksta i programskog koda, VS Code ima posebne funkcionalnosti za rad s Markdown datotekama. Prvenstveno sadrži isticanje Markdown sintakse, tako što pojedine oznake prikazuje u različitim bojama. Također, sve poveznice koje se nalaze u tekstu mogu se otvoriti izravno iz uređivača. Osim toga, VS Code ima i poseban prozor u kojemu se prikazuje dokument u svom formatiranom obliku.

Iako ima pristojnu podršku za Markdown, uređivanje datoteka moguće je isključivo kao tekstualni dokument, s formatiranim oblikom prikazanim u posebnom dijelu sučelja. Slika 1 prikazuje rad s Markdown datotekom u VS Code-u.

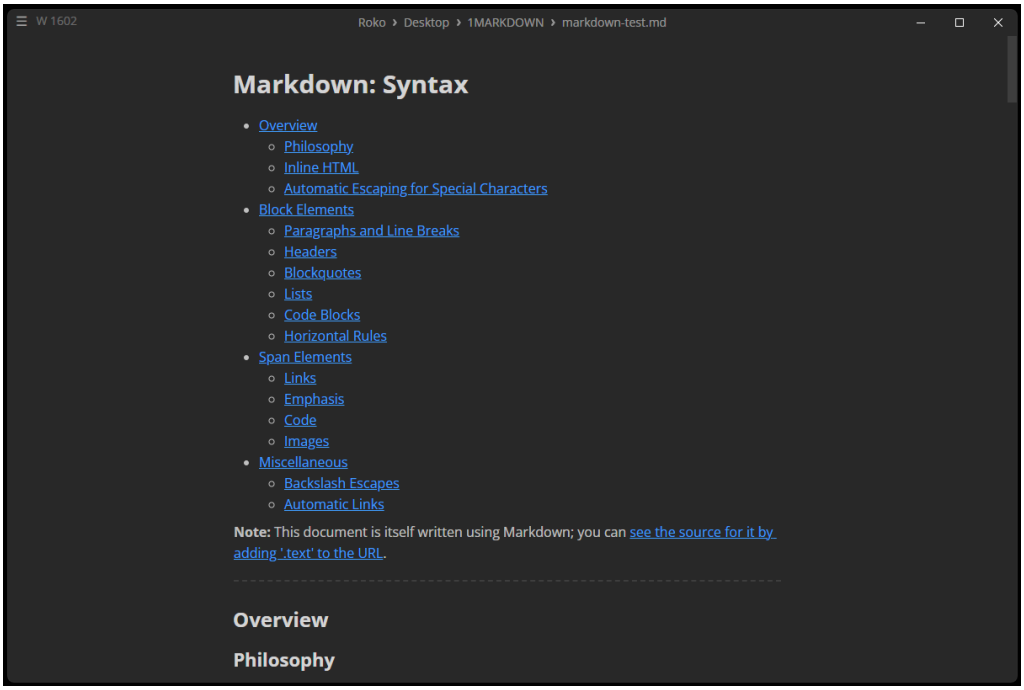


Slika 1. Primjer rada s Markdown datotekom u Visual Studio Code-u

2.2. MarkText

MarkText je uređivač Markdown datoteka, izgrađen koristeći Electron i Vue.js. Dizajniran je kao „What you see is what you get“ uređivač, gdje se Markdown datoteke tokom uređivanja prikazuju formatirane. Omogućuje mijenjanje tema sučelja te podržava GitHub Flavoured Markdown specifikaciju (MarkText GitHub, 2022).

No međutim, većina rada s ovom aplikacijom znatno ovisi o poznavanju svih njenih funkcionalnosti i kontrola, jer se Markdown sadržaj prikazuje u izvornom obliku jedino kad se uređuje pojedina riječ s nekom Markdown oznakom. Također, iako ima nekoliko tema sučelja, ne omogućuje stvaranje vlastitih. Na slici 2 prikazano je sučelje MarkText-a tokom uređivanja datoteke.

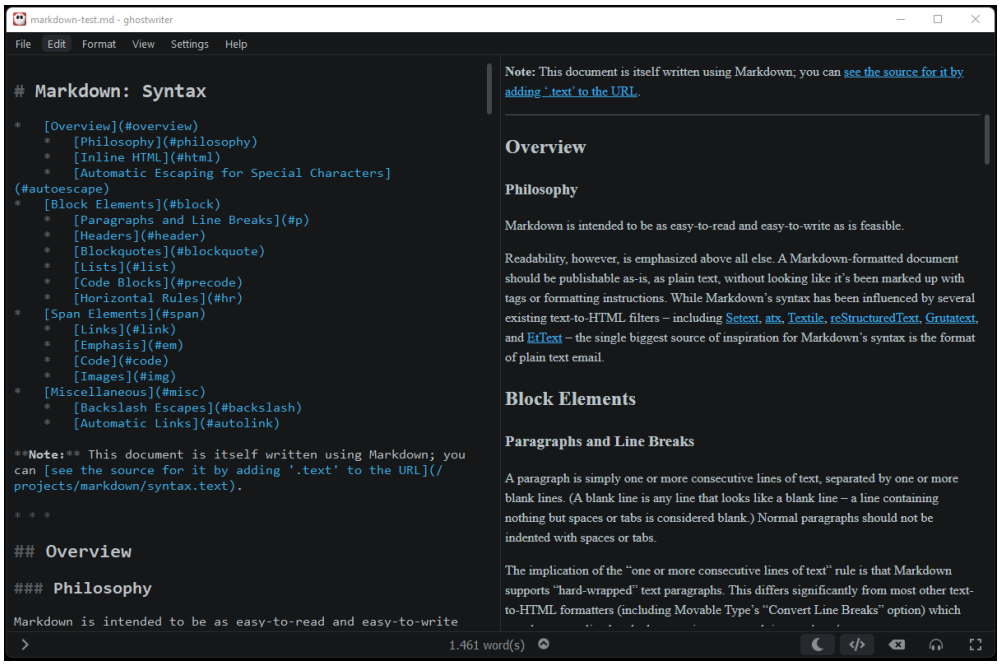


Slika 2. Primjer rada s Markdown datotekom u MarkText-u

2.3. ghostwriter

Ghostwriter je besplatni Markdown uređivač, napisan u C++. Koristi jednostavno i minimalno sučelje, koje se može prilagoditi s vlastitim temama (Ghostwriter website, 2022). Unutar dijela za uređivanje, Markdown dokument se prikazuje u svom izvornom obliku, ali s formatiranjem koje se podudara s sintaktičkim oznakama. Pored dijela za uređivanje nalazi se prozor s formatiranim dokumentom, što je prikazano na slici 3.

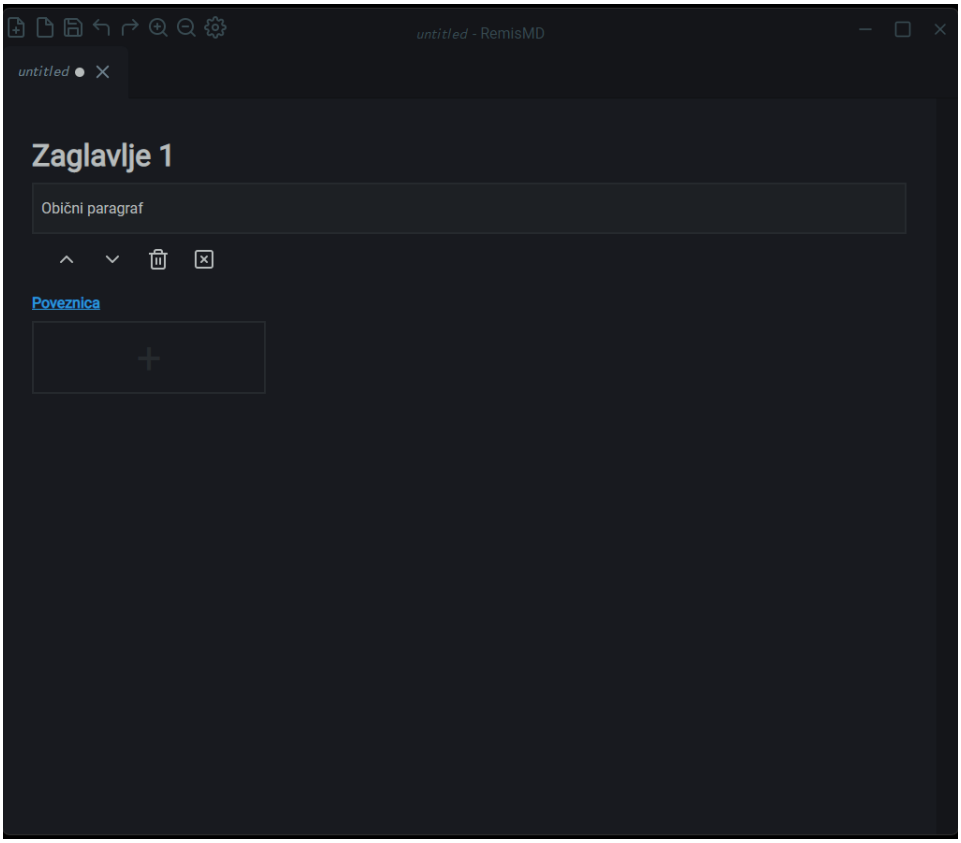
Iako sve tri aplikacije imaju razne funkcionalnosti za rad s Markdown dokumentima, nijedna nema mogućnost uređivanja dokumenta tako da se on prikaže u formatiranom obliku, a da korisnik pojedinu liniju uređuje u običnom neformatiranom Markdown-u.



Slika 3. Primjer rada s Markdown datotekom u ghostwriter-u

3. FUNKCIONALNOSTI APLIKACIJE

Nakon pokretanja aplikacije korisniku se prikazuje početni zaslon. Odavde korisnik može stvoriti novu Markdown datoteku ili otvoriti postojeću. Nakon što je učinio jedno od toga prikazuje mu se njegova datoteka, prazna u slučaju da je odabrao kreiranje nove datoteke. Sadržaj dokumenta prikazuje se formatiran, a ukoliko želi dodati novu liniju, to može učiniti klikom na element sa znakom plus. Kad se stvori nova linija, u nju može unositi željeni Markdown tekst pa pritiskom na tipku „ESC“ ili klikom na ikonu na dnu linije prestati je uređivati i prikazati je formatiranu. Kao što je prikazano na slici 4, na dnu linije se nalaze i dodatne funkcionalnosti, koje također imaju svoje tipkovničke prečace. Prikazanim strelicama može dodati liniju ispod ili iznad trenutne, a klikom na koš za smeće ukloniti trenutnu liniju iz dokumenta. Korištenjem prečaca „CTRL + strelica gore/dolje“ jednostavno može odabirati neku drugu liniju.

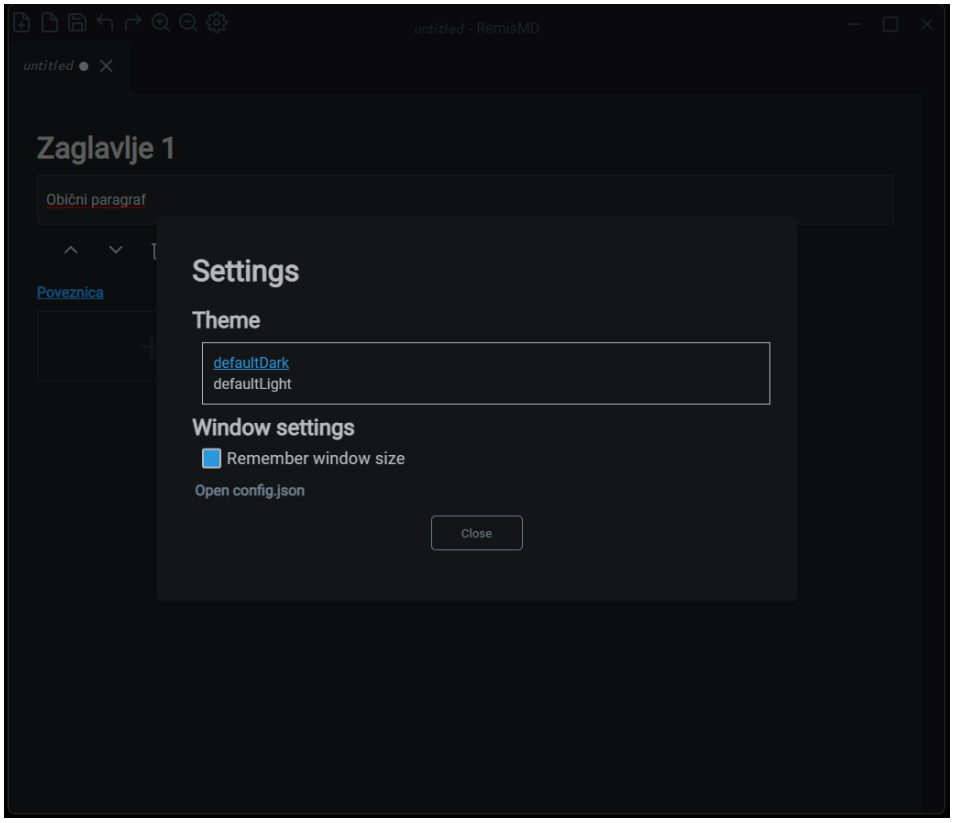


Slika 4. Sučelje aplikacije tokom uređivanja dokumenta

Iznad dijela za uređivanje nalazi se područje s karticama, koje prikazuju sve datoteke otvorene u trenutnoj sesiji. Pomoću njih korisnik kontrolira koji dokument

uređuje te može zatvoriti pojedini dokument bez izlaska iz aplikacije. Uz to, prikazuje mu se status svakog dokumenta, odnosno, postoje li nespremljene promjene.

Na samom vrhu sučelja nalazi se alatna traka, koja korisniku omogućuje korištenje raznih funkcionalnosti. Prve tri ikone u traci odnose se na rad s dokumentima. Prva je za stvaranje novog dokumenta, druga za otvaranje postojećeg, a treća za spremanje trenutno otvorenog. Zatim slijede dvije ikone za poništi i ponovi, pa onda dvije ikone za podešavanje veličine sučelja, odnosno, zumiranja. Posljednja ikona služi za otvaranje postavki. Postavke, prikazane na slici 5, su podijeljene u tri dijela. Prvi se tiče teme sučelja. Ovdje se prikazuju sve teme koje korisnik ima na raspolaganju, uključujući i one koje sam definira, te mu je označena ona koja je trenutno odabrana. Drugi dio postavki odnosi se na postavke prozora. Sadrži postavku koja odlučuje treba li pamtiti dimenzije prozora nakon što se aplikacija zatvori.



Slika 5. Aplikacija s otvorenim postavkama

Posljednji dio je gumb za otvaranje config.json datoteke, gdje se nalaze korisnikove postavke u JSON obliku. Unutar ove datoteke korisnik može i definirati nove teme sučelja, tako što u objekt themes doda objekt koji nosi željeni naziv te sadrži

prilagođene boje nazvane isto kao i u zadanim temama. Sve ikone u alatnoj traci imaju relevantne tipkovničke prečace, koji se mogu pregledati prelaskom miša. Sadržaj datoteke prikazan je na slici 6.

Slika 6. Sadržaj datoteke config.json

4. KORIŠTENE TEHNOLOGIJE

4.1. JavaScript

JavaScript je interpretirani, objektno-orijentirani programski jezik, stvoren za dodavanje interaktivnosti na web stranice. Iako podržava razne programske paradigme, poput imperativnog i funkcijskog programiranja, dizajniran je kao prototype-based objektno-orijentirani jezik. Sintaksa mu je slična jezicima poput C i Java, koristi dinamične tipove, a sve funkcije su prve klase, što znači da se mogu koristiti poput varijabli.

Prvu implementaciju JavaScript-a izradio je Brendan Eich, kao zaposlenik američke tvrtke Netscape za njihov web preglednik Netscape Navigator, 1995. godine (Mozilla Developer Network, 2022). Za svoj preglednik Internet Explorer, Microsoft je zatim izradio vlastitu verziju Javascript interpretera pod nazivom JScript, što je ubrzo dovelo do stvaranja značajnih razlika između ta dva jezika. Povodom toga, neprofitna organizacija Ecma International je definirala specifikaciju ECMAScript kako bi se standardizirali jezici korišteni u web preglednicima. Unatoč tome, nejednakosti između interpretera su postojale sve do objave ECMAScript 5 standarda 2009. godine (Ecma International, 2022). Većina suvremenih preglednika koriste jednu od tri implementacije JavaScript engine-a: Google-ov V8 (Chrome, Edge, Opera), Mozilla SpiderMonkey (Firefox, Netscape Navigator) i JavaScriptCore (Safari).

Trenutno je JavaScript jedan od najzastupljenijih programskih jezika. Prema Stack Overflow anketi za 2022. godinu (Stack Overlfow Developer Survey, 2022), 65.36% programera je izjavilo da koriste JavaScript, a w3techs (W3Techs, 2022) tvrdi da 98% web stranica na internetu sadrži JavaScript kod.

4.2. Node.js

Node.js je asinkrono okruženje za izvođenje JavaScript-a. Ono omogućuje interpretiranje JavaScript koda izvan web preglednika što stvara mogućnost razvoja samostalnih aplikacija, poput web poslužitelja, mrežnih aplikacija ili programskih skripti. Koristeći module posebno napravljene za Node.js, razvojni programeri mogu pisati JavaScript programe koji imaju pristup operativnom sustavu, njegovim datotekama, raznim mrežnim funkcionalnostima (HTTP, TCP, UDP) ili izravnom radu

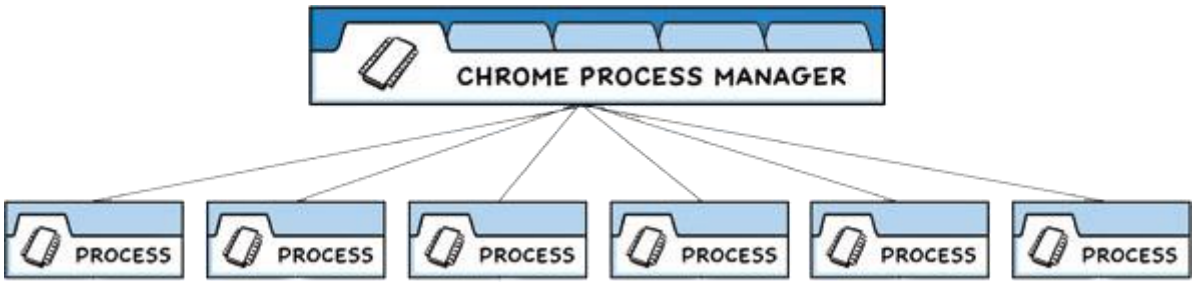
s računalnim hardverom (Node.js, 2022). Ova fleksibilnost omogućuje web programerima da koriste isti programski jezik, JavaScript, pri razvoju i front-end i back-end dijelova web aplikacija.

Node.js je izradio Ryan Dahl 2009. godine, kao projekt otvorenog koda, koristeći Google-ov V8 JavaScript engine. Od tada je postao najkorištenija web razvojna tehnologija (Stack Overlfow Developer Survey, 2022) te su neke funkcionalnosti koje je uveo Node.js, poput rada s modulima, postale dio ECMAScript standarda. Pošto moduli podrazumijevaju učitavanje programskog koda iz vanjskih datoteka, za njihovo upravljanje stvoren je upravitelj programskih biblioteka npm koji dolazi uz Node.js. Npm omogućuje preuzimanje i upravljanje biblioteka programskog koda koje su dostupne u npm bazi podataka. U 2020. godini npm baza je sadržavala više od milijun JavaScript biblioteka (npm, 2022). Danas Node.js održava neprofitna organizacija OpenJS foundation.

4.3. Electron

Electron je razvojni okvir za JavaScript, koji je 2013. godine objavila Microsoft-ova tvrtka GitHub, trenutno održavan od strane OpenJS organizacije. On omogućuje razvojnim programerima stvaranje više-platformnih desktop aplikacija koristeći JavaScript, HTML i CSS. Koristi Node.js za pristup operacijskom sustavu i modificiranu verziju Google-ovog Chromium web preglednika za prikaz grafičkog sučelja aplikacije. Neki od poznatih računalnih programa koji koriste Electron su Visual Studio Code, Discord, Slack, WhatsApp i Figma. Verzija Electron-a korištena za razvoj aplikacije ovog rada je 13.0.0.

Electron od Chromium preglednika nasljeđuje svoju multi-process arhitekturu, ilustriranu na slici 7, što ga čini sličnim običnim suvremenim web preglednicima. Pri dizajniranju Google Chrome web preglednika, njegovi razvojni programeri odlučili su za svaku otvorenu karticu pokrenuti poseban proces na operativnom sustavu. Time se ograničava šteta koja se može prouzročiti lošim ili zlonamjernim kodom na web stranicama. Upravljanje svim procesima vrši se iz jednog, glavnog procesa.



Slika 7. Chrome Multi-Process model

Struktura Electron aplikacija koristi isti model upravljanja. Ima jedan glavni proces, nazvan main, koji služi kao ulazna točka aplikacije. On se pokreće u instanci Node.js okruženja te stoga ima mogućnost koristiti sve funkcionalnosti koje su jedinstvene za Node. Druga vrsta procesa, zvana renderer, stvara Chromium prozore, u kojima se nalazi sučelje aplikacije, izrađeno web tehnologijama HTML, CSS i JavaScript. To omogućuje korištenje JavaScript okvira i biblioteka, kao što su React, Vue.js ili Angular (Electron JS, 2022).

Zbog ovakve arhitekture, renderer nema izravan pristup main procesu, pa time ni samom Node.js instancom. Ipak, većina desktop aplikacija kroz sučelje mora imati pristup pohranjenim dokumentima ili funkcionalnostima operacijskog sustava. Za tu svrhu se koristi Inter-Process Communication (IPC). Ovaj protokol omogućuje komunikaciju između procesa koristeći kanale, definirane s ipcMain i ipcRenderer modulima u Electron-u. Uspostavljanje komunikacije definira se u preload datoteci, koja se izvršava prije učitavanja web sadržaja u renderer procesu. Ova datoteka na window JavaScript objekt u renderer procesu dodaje sve funkcije koje su u njoj definirane.

4.4. Vue.js

Vue je JavaScript razvojni okvir za stvaranje korisničkih sučelja. Nadovezuje se na standardni HTML, CSS i JavaScript te pruža deklarativni model programiranja, baziran na komponentama. Dvije su glavne značajke Vue okvira. Prva je deklarativno prikazivanje (eng. declarative rendering) koje proširuje HTML s dodatnom sintaksom koja omogućuje deklarativno opisivanje HTML izlaza ovisno o stanju u JavaScript procesu. Druga glavna značajka je reaktivnost (eng. reactivity), koja prati stanje u

JavaScript-u pa prema tome učinkovito ažurira stanje u dokumentu (Vue.js, 2022). Za razvoj aplikacije vezane uz ovaj rad, koristi se Vue verzija 3.2.13

Primarni način rada i podjele funkcionalnosti u Vue aplikacijama su komponente u formatu Single-File Component. One enkapsuliraju svu logiku jednog dijela sučelja, HTML, CSS i JavaScript, u jednu datoteku s nastavkom .vue. Komponenta može predstavljati cijelu stranicu ili njezine dijelove poput izbornika, podnožja, formi i sličnog.

Kako bi se pojednostavio rad s HTML oznakama, Vue sadrži vlastite direktive koje ga proširuju. Jedna od često korištenih direktiva jest v-if, koja prikazuje element samo ukoliko je neki uvjet ispunjen. Uz ovu direktivu mogu se koristiti i v-else i v-else-if direktive koje omogućuju kompleksnije grananje. U primjeru na slici 8, button element biti će prikazan isključivo ako je varijabla proizvodDostupan istinita, inače će se prikazati p element.

```
<button v-if="proizvodDostupan==true">Kupi</button>
<p v-else>Proizvod nije dostupan</p>
```

Slika 8. Primjer v-if i v-else direktiva

Za iteraciju kroz elemente u JavaScript polju koristi se v-for direktiva. Pomoću nje se za svaki element polja može prikazati posebna HTML oznaka. U primjeru na slici 9, za svaki proizvod u polju proizvodi se prikazuje poseban p element u kojemu se nalazi cijena tog proizvoda.

```
<p v-for="proizvod in proizvodi">
{{ proizvod.cijena }}
</p>
```

Slika 9. Primjer v-for direktive

Pošto Vue komponente u sebi mogu sadržavati druge Vue komponente, potrebno je moći dijeliti podatke između njih. Za tu svrhu se koriste props atributi. Oni predstavljaju jednosmjernu komunikaciju između komponenti, gdje komponenta iz svoje roditelj-komponente dobiva podatke. No ponekad je potrebno slati podatke u

obrnutom smjeru pa se za tu svrhu koriste emit funkcije. Kada komponenta ima definirane emit funkcije, njezina roditelj-komponenta može “slušati” emitirane događaje te ih procesuirati relevantnim funkcijama. Slika 10 prikazuje prosljeđivanje prop atributa i „slušanje“ emit atributa. Watch funkcije prate vrijednost željene varijable te ukoliko se ona promijeni pozivaju definiranu funkciju.

```
<tab-container
  :tabs="files"
  :currentTab="currentTab.id"
  :saving="saving"
  @closeTab="closeTab"
  @setCurrentTab="setCurrentTab"
/>
```

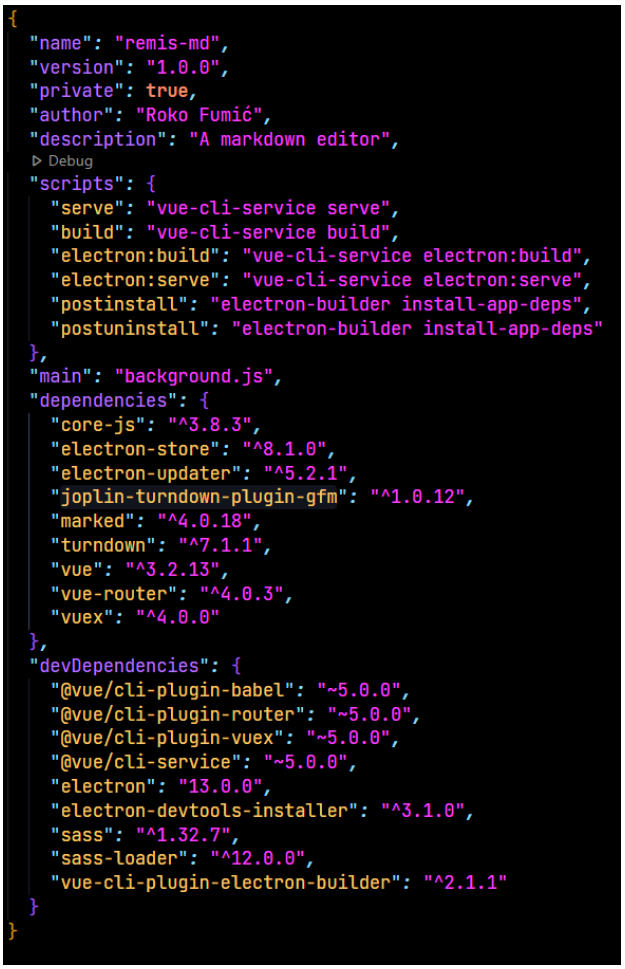
Slika 10. Primjer korištenja prop i emit atributa

4.5. Npm biblioteke

Inicijalno postavljanje aplikacije bilo je pokrenuto koristeći nastavak za Vue CLI pod imenom Vue CLI Plugin Electron Builder. Ovaj nastavak namješta razvojni okoliš za razvoj Electron aplikacija koristeći Vue okvir. Za tu svrhu instalira razne npm pakete koji su potrebni za rad. Oni se definiraju, zajedno s njihovim verzijama, u package.json datoteci u korijenskoj mapi projekta. Core-js je paket koji omogućuje korištenje ECMAScript značajki koje nisu dostupne u Googleovom V8 JavaScript enginu. Vue, vue-router i vuex su biblioteke za rad s vue.js okvirom. Uz to su instalirani paketi koji se koriste isključivo tokom razvoja te se ne nalaze u finalnom proizvodu. Electron i electron-devtools-installer služe stvaranje razvojnog Electron okruženja i instaliranja Chrome produžetaka koji pomažu pri razvoju. Ovdje pripada i sam Vue CLI plugin Electron Builder koji se koristi tokom razvoja i stvaranja binarnih datoteka programa. Sass i Sass Loader su paketi koji omogućuju rad s SASS/SCSS jezikom, koji je proširenje običnog CSS-a. Oni omogućuju pisanje SASS skripti koje onda transformiraju u obični CSS (Sass, 2022).

Osim početnih paketa, za aplikaciju je bilo potrebno instalirati dodatne biblioteke koje olakšavaju rad. Electron-store je biblioteka za stvaranje jednostavnog trajnog skladištenja podatka. U ovom projektu se koristi za spremanje korisničkih postavki i

tema sučelja. Podaci se pohranjuju u datoteku na korisnikovom računalu pod nazivom config.json (Electron-store GitHub, 2022). Electron-updater je modul za automatsko ažuriranje Electron aplikacija. Ovaj paket je odabran zbog svog jednostavnog namještanja. koje omogućuje automatsko ažuriranje bez posvećenog web poslužitelja, koristeći distribucije objavljene na GitHub-u (Electron-updater GitHub, 2022). Marked je paket za brzo i lagano pretvaranje Markdown sadržaja u HTML (Marked.js, 2022). Ovaj paket je primarno rješenje koje se koristi za prikaz korisnikovih datoteka te po već zadanim postavkama poštuje GitHub Flavoured Markdown specifikaciju. Turndown je paket za konverziju HTML stringa u Markdown (Turndown GitHub, 2022). U aplikaciji se koristi kako bi se datoteke mogle hijerarhijski podijeliti da bi se omogućilo uređivanje svake linije posebno. Međutim, ovaj paket ne prati GitHub Flavoured Markdown specifikaciju, stoga je bilo potrebno instalirati i paket joplin-turndown-plugin-gfm. Slika 11 prikazuje package.json datoteku s popisom npm paketa.



Slika 11. package.json datoteka

5. IMPLEMENTACIJA

5.1. Background.js

U background.js datoteci se nalazi ulaz u aplikaciju, odnosno pokretanje main procesa. Kao što je vidljivo na slici 12, ovdje je definirana asinkrona createWindow funkcija, koja stvara glavni renderer prozor za RemisMD. Funkcija započinje stvaranjem instance klase BrowserWindow, dostupne u Electron biblioteci. Kod stvaranja instance definira se visina i širina prozora, čije se vrijednosti dohvaćaju funkcijom windowSettings in setting.js datoteke. Postavljanjem svojstva frame na neistinitu vrijednost uklanja se okvir koji se nalazi na desktop aplikacijama, zato što se za RemisMD koristio prilagođeni okvir. AutoHideMenuBar svojstvo je postavljeno na istinitu vrijednost kako bi se iz aplikacije uklonila početna izborna traka.

```
async function createWindow() {
  const bounds = windowSettings.getBounds();
  // Create the browser window.
  const win = new BrowserWindow({
    width: bounds[0],
    height: bounds[1],
    // minWidth: 640,
    // minHeight: 720,
    frame: false,
    autoHideMenuBar: true,
    backgroundColor: '#141519',

    webPreferences: {
      // Use pluginOptions.nodeIntegration, leave this alone
      // See nklayman.github.io/vue-cli-plugin-electron-builder/g
      nodeIntegration: process.env.ELECTRON_NODE_INTEGRATION,
      contextIsolation: !process.env.ELECTRON_NODE_INTEGRATION,
      preload: path.resolve(__static, 'preload.js'),
    },
  });
```

Slika 12. Funkcija createWindow

Kako bi aplikacija mogla pamtiti korisnikovu prijašnju veličinu prozora, koristi se funkcija BrowserWindow klase zvana on s opcijom resized. Ona poziva neku funkciju svaki put kad korisnik promijeni dimenzije prozora. U ovom slučaju, poziva funkciju setBounds iz datoteke settings.js kojoj kao argument unosi dimenzije prozora. Još

jedna funkcija klase BrowserWindow koja se koristi je webcontents.on s argumentom will-navigate. Ova funkcija se koristi kad korisnik pritisne na poveznicu u svojim Markdown datotekama te, umjesto stvaranja novog prozora aplikacije, otvara poveznicu u korisnikovom preferiranom web pregledniku. Navedene dvije funkcije prikazane su na slici 13.

```
win.on('resized', () => windowSettings.setBounds(win.getSize(), true));

win.webContents.on('will-navigate', (event, url) => {
  // stop Electron from opening another BrowserWindow
  event.preventDefault();
  // open the url in the default system browser
  shell.openExternal(url);
});
```

Slika 13. Funkcije klase BrowserWindow

Zatim slijede funkcije koje prate IPC kanale te time kontroliraju komunikaciju između main i renderer procesa. Prvih nekoliko funkcija kontroliraju funkcionalnosti poput izlaska iz aplikacije, minimiziranje prozora te zumiranje sučelja aplikacije. Ukoliko renderer pozove funkcije openConfig, setCurrentTheme ili updateSettings izvršavaju se relevantne funkcije iz datoteke settings.js. Funkcije su prikazane na slici 14.

```
ipcMain.on('closeApp', () => {
  win.close();
});

ipcMain.on('minimizeApp', () => {
  win.minimize();
});

ipcMain.on('maximizeApp', () => {
  if (win.isMaximized()) {
    win.restore();
  } else {
    win.maximize();
  }
});

ipcMain.on('zoomIn', () => {
  win.webContents.zoomFactor = win.webContents.getZoomFactor() + 0.2;
});
ipcMain.on('zoomOut', () => {
  win.webContents.zoomFactor = win.webContents.getZoomFactor() - 0.2;
});

ipcMain.on('undo', () => {
  win.webContents.undo();
});

ipcMain.on('redo', () => {
  win.webContents.redo();
});

ipcMain.on('openConfig', async () => {
  openSettingsFile();
});

ipcMain.on('setCurrentTheme', async (event, themeName) => {
  setCurrentTheme(themeName);
});
ipcMain.on('updateSettings', async (event, args) => {
  updateSettings(args);
});
```

Slika 14. ipcMain.on funkcije

Iduća IPC funkcija kontrolira slučaj kad korisnik želi otvoriti Markdown datoteku u RemisMD koristeći opciju “Open With”. Na Windows operacijskim sustavima, ako se neka aplikacija otvori putem opcije “Open With” ona kao jedan od svojih argumenata prima putanju datoteke koja se želi otvoriti. Stoga se iz globalnog Node objekta process dohvaćaju argumenti aplikacije te se za svaku datoteku njezino ime dohvaća pomoću Node modula path, njezin sadržaj pomoću modula fs, sinkronom funkcijom readFileSync, i sve se zajedno se vraća kao povratna vrijednost tipa polje.

Zatim je definirana funkcija koja se poziva prilikom IPC metode openFile. Ova funkcija služi za otvaranje novih datoteka u aplikaciju. Prvo, pomoću funkcije showOpenDialogSync iz Electron biblioteke, korisniku prikazuje prozor za odabir datoteke na svom računalu, čija je povratna vrijednost putanja datoteke. Zatim se vraća polje sa nazivom dokumenta na prvom mjestu, sadržaj datoteke na drugom i putanje na trećem, opet koristeći path i fs module is Node.js-a.

SaveFile IPC funkcija služi za spremanje modificirane datoteke, Kao argument prima objekt s putanjom datoteke i njenim sadržajem. Ukoliko objekt nema definiranu putanju korisniku se prikazuje prozor za odabir mjesta na koje će se spremiti datoteka, što se koristi za funkcionalnost “Spremi Kao”.

Posljednje tri IPC funkcije su getCurrentTheme, getAllThemes i getAllSettings. One pozivaju relevantne funkcije iz datoteke settings.js i vraćaju njihove povratne vrijednosti. Implementacija navedenih šest funkcija prikazana je na slici 15.

```
ipcMain.handle('openWith', () => {
  if (app.isPackaged) {
    process.argv.unshift(null);
  }
  if (process.argv.length > 2) {
    let data = process.argv.slice(2).map((f) => {
      return {
        name: path.basename(f),
        content: fs.readFileSync(f).toString(),
        path: f,
      };
    });
    return data;
  }
});

ipcMain.handle('openFile', () => {
  const [file] = dialog.showOpenDialogSync(win, {
    properties: ['openFile'],
    filters: [{ name: 'Markdown', extensions: ['md', 'markdown'] }],
    title: 'Open a file',
  });
  if (file) {
    return [path.basename(file), fs.readFileSync(file).toString(), file];
  }
});

ipcMain.handle('saveFile', async (event, data) => {
  let file = data;
  if (file.path === undefined) {
    file.path = dialog.showSaveDialogSync(win, {
      title: 'Save file',
      filters: [{ name: 'Markdown', extensions: ['md', 'markdown'] }],
    });
  }

  fs.writeFileSync(file.path, file.content);
  return [path.basename(file.path), file.content, file.path];
});

ipcMain.handle('getCurrentTheme', () => {
  return getCurrentTheme();
});
ipcMain.handle('getAllThemes', () => {
  return getAllThemes();
});
ipcMain.handle('getAllSettings', () => {
  return getAllSettings();
});
```

Slika 15. ipcMain.handle funkcije

5.2. Settings.js

Datoteka settings.js je modul za rad s electron-store npm paketom. U njoj se definira struktura config.json datoteke koja sadrži korisničke postavke i teme sučelja. Definirana je kao JavaScript objekt te pohranjena u varijablu schema, čiji je sadržaj prikazan na slici 16. Sastoji se od dvije pred definirane teme te postavki za trenutno odabranu temu, veličinu prozora sučelja i opcije vezane uz njeno pamćenje.

```
const schema = {
  themes: {
    defaultDark: {
      background: '#141519',
      foreground: '#b6bbbb',
      editor: '#181a1f',
      textActive: '#6d808c',
      textInactive: '#394b52',
      textArea: '#1d2024',
      textAreaBorder: '#262a2e',
      highlight: '#2b98db',
    },
    defaultLight: {
      background: '#ebebeb',
      foreground: '#455067',
      editor: '#f4f4f4',
      textActive: '#61C1DA',
      textInactive: '#8B99B1',
      textArea: '#ECF2F3',
      textAreaBorder: '#DCEDF0',
      highlight: '#DA7A61',
    },
  },
  settings: {
    currentTheme: 'defaultDark',
    rememberWindowSize: true,
    windowBounds: [800, 800],
  },
};
```

Slika 16. Struktura config.json datoteke

Prva funkcija definirana u datoteci je checkStorage, koja provjerava postoji li “data” svojstvo u config.json datoteci pomoću funkcija iz electron-storage te ukoliko ne postoji stvara ga prema objektu definiranim u varijabli schema. Ova funkcija se poziva u svakoj drugoj funkciji unutar ove datoteke kako bi se osiguralo da config.json postoji i sadrži sva potrebna svojstva.

Zatim se definira objekt windowSettings koji sadrži dvije funkcije vezane uz postavke prozora sučelja. Prva od njih, getBounds, dohvaća windowBounds varijablu iz pohrane, a druga setBounds postavlja tu njenu vrijednost.

Iduće tri funkcije, getAllThemes, getTheme i getCurrentTheme, vezane su za dohvaćanje tema sučelja. Prva dohvaća sve definirana teme, druga dohvaća jednu

temu prema njenom imenu dobivenom kao argument, a treća dohvaća temu koja se trenutno koristi. Uz njih je i funkcija setCurrentTheme koja postavlja trenutnu temu na vrijednost dobivenu kao argument.

Funkcija openSettingsFile koristi electron-storage funkciju openInEditor koja otvara config.json u korisnikovom preferiranom uređivaču JSON datoteka. GetAllSettings i updateSettings dohvaćaju i ažuriraju korisničke postavke. Slika 17 prikazuje kod svih funkcija u datoteci.

```
function checkStorage() {
  const data = storage.get('data');
  if (!data) {
    storage.set('data', schema);
  }
}
const windowSettings = {
  getBounds: () => {
    checkStorage();
    return storage.get('data.settings.windowBounds');
  },
  setBounds: (bounds, isFromResize) => {
    const setting = storage.get('data.settings.rememberWindowSize');
    if (setting || !isFromResize) {
      storage.set('data.settings.windowBounds', bounds);
    }
  },
};

function getAllThemes() {
  checkStorage();
  return storage.get('data.themes');
}

function getTheme(themeName) {
  checkStorage();
  return storage.get(`data.themes.${themeName}`);
}

function getCurrentTheme() {
  checkStorage();
  const themeName = storage.get('data.settings.currentTheme');
  return {
    theme: themeName,
    ...storage.get(`data.themes.${themeName}`),
  };
}

function setCurrentTheme(themeName) {
  checkStorage();
  storage.set('data.settings.currentTheme', themeName);
}

function openSettingsFile() {
  checkStorage();
  storage.openInEditor();
}

function getAllSettings() {
  return storage.get('data.settings');
}

function updateSettings({ setting, value }) {
  storage.set(`data.settings.${setting}`, value);
}
```

Slika 17. Funkcije datoteke settings.js

5.3. HomeView.vue

Svi elementi vizualnog aplikacije nalaze se unutar Vue komponente HomeView.vue. Ona sadrži sve funkcije koje se pozivaju pri otvaranju, spremanju i kreiranju datoteka, mijenjanju boja na sučelju, te učitava dodatnih pet komponenti aplikacije. Pri stvaranju komponente, postavljaju se dva event-listener-a koji “slušaju” korisnikov unos tipkovnice, te ukoliko pritisne kombinaciju tipki “CTRL + N” ili “CTRL + O” poziva funkciju za kreiranje nove datoteke i funkciju za otvaranje datoteke, respektivno. Kod za event-listener funkcije prikazan je na slici 18.

```
window.addEventListener('keyup', (key) => {
  if ((key.key === 'n' || key.key === 'N') && key.ctrlKey) {
    handleNewFile();
  }
  if ((key.key === 'o' || key.key === 'O') && key.ctrlKey) {
    handleOpenFile();
  }
});
```

Slika 18. Event-listener funkcije

Također, pri učitavanju HomeView komponente poziva se asinkrona anonimna funkcija, čija je definicija prikazana na slici 19. Ona se izvodi u dva koraka. Prvo, koristeći IPC mogućnosti Electron-a, funkcijom getCurrentTheme dohvaća korisnikovu odabranu temu sučelja, te je zatim pomoću funkcije setTheme postavlja kao temu. Funkcija setTheme odabire “korijenski” element :root HTML dokumenta pa na njega dodaje osam CSS varijabli koje se zatim kroz aplikaciju koriste za bojanje raznih elemenata sučelja. Drugi korak funkcije je dohvaćanje argumenata koji su zadani pri pokretanju aplikacije. Stoga, ovaj dio funkcije dohvaća dokumente koji su otvoreni pri pokretanju aplikacije, koristeći openWith IPC funkciju, te ih dodaje u polje koje referencira varijabla “files”.

```
function setTheme(theme) {
  const root = document.querySelector(':root');
  root.style.setProperty('--background', theme.background);
  root.style.setProperty('--editor', theme.editor);
  root.style.setProperty('--foreground', theme.foreground);
  root.style.setProperty('--highlight', theme.highlight);
  root.style.setProperty('--textActive', theme.textActive);
  root.style.setProperty('--textArea', theme.textArea);
  root.style.setProperty('--textAreaBorder', theme.textAreaBorder);
  root.style.setProperty('--textInactive', theme.textInactive);
}
(async () => {
  let currentTheme = await window.electronAPI.getCurrentTheme();
  setTheme(currentTheme);
  let argFiles = await window.electronAPI.openWith();
  if (argFiles !== undefined) {
    argFiles.forEach((f) => {
      let data = {
        id: Date.now(),
        name: f.name,
        content: parseFile(f.content),
        path: f.path,
        unsaved: false,
      };
      files.value.push(data);
      store.commit('addFile', { ...data });
    });
    currentTab.value = files.value[files.value.length - 1];
  }
})();
```

Slika 19. Funkcija koja se poziva pri učitavanju

HomeView.vue komponenta u svom template dijelu sadrži pet komponenti koje predstavljaju različite dijelove sučelja aplikacije. Komponenta koja se nalazi na samom vrhu dokumenta jest MenuBar.vue. Ona služi za pozivanje raznih funkcija kod rada s markdown dokumentima, kao i kontrole za izlaz iz aplikacije i minimiziranje prozora. Slijedeća komponenta je TabContainer.vue koja služi za upravljanje otvorenim dokumentima putem kartica, slično kao kartice u web preglednicima. Treća komponenta je EditorComponent.vue. Ona omogućuje uređivanje Markdown datoteke koja je trenutno odabrana, odnosno, pohranjena u varijablu currentTab. Prikazuje se isključivo ako varijabla showEditor ima istinitu vrijednost. Komponenta DefaultScreen.vue služi za prikaz početnog ekrana, a Settings.vue za prikaz prozora s postavkama. Kod HTML dijela komponente je prikazan na slici 20.

```
<template>
  <menu-bar
    @openFile="handleOpenFile"
    @saveFile="handleSaveFile"
    @newFile="handleNewFile"
    @settings="showModal = true"
    :windowTitle="currentTab.name"
  />

  <tab-container
    :tabs="files"
    :currentTab="currentTab.id"
    :saving="saving"
    @closeTab="closeTab"
    @setCurrentTab="setCurrentTab"
  />
  <editor-component
    v-if="showEditor"
    :file="currentTab"
    :reset="resetEditor"
    @save-file="handleSaveFile"
    @save-as="handleSaveAs"
    @changeUnsaved="changeUnsaved"
  />
  <default-screen @open-file="handleOpenFile" @new-file="handleNewFile" />
  <Teleport to="#modal">
    <settings
      v-if="showModal"
      @closeModal="showModal = false"
      @setCurrentTheme="setCurrentTheme"
    />
  </Teleport>
</template>
```

Slika 20. HTML dio HomeView komponente

Za rad s markdown datotekama na računalu korisnika postoje četiri funkcije, čiji je kod prikazan na slikama 21 i 22. HandleOpenFile se poziva ukoliko je korisnik unio tipkovnički prečac “CTRL + O” ili pritisnuo gumb za otvaranje datoteke u komponenti MenuBar.vue. Pri pozivu te funkcije pomoću IPC funkcije openFile dohvaćaju se naziv, sadržaj i putanja datoteke koju korisnik želi otvoriti. Ti podaci se zatim, u obliku JavaScript objekta, dodaju u polje referencirano varijablom files i time je datoteka učitana unutar aplikacije. HandleNewFile poziva se ukoliko je korisnik unio tipkovnički prečac “CTRL + N” ili pritisnuo gumb za stvaranje nove datoteke u komponentama MenuBar.vue ili DefaultScreen.vue. Funkcija zatim u varijablu files dodaje JavaScript objekt bez definiranog sadržaja, naziva i putanje. Za spremanje novostvorene ili otvorene datoteke služi funkcija handleSaveFile. Poziva se iz komponenti MenuBar.vue i EditorComponent.vue. Ova procedura, pomoću IPC funkcije saveFile,

sprema promjene trenutno odabrane datoteke koju je korisnik uredio tako što šalje putanju i novi sadržaj datoteke u pozadinski Node.js proces koji ima mogućnost pisanja na disk. Funkcija handleSaveAs se koristi kad korisnik želi spremiti kopiju datoteke koju trenutno uređuje. Ona se poziva nakon unosa tipkovničkog prečaca “CTRL + SHIFT+ S” u komponenti EditorComponent.vue.

```
async function handleOpenFile() {
  try {
    let [name, content, path] = await window.electronAPI.openFile();
    let data = {
      id: Date.now(),
      name,
      content: parseFile(content),
      path,
      unsaved: false,
    };
    files.value.push(data);
    store.commit('addFile', { ...data });
    currentTab.value = files.value[files.value.length - 1];
  } catch (error) {
    if (error.name !== 'AbortError') {
      console.error(error);
    }
  }
}

async function handleSaveFile() {
  const file = store.getters.getFile(currentTab.value.id);
  try {
    changeUnsaved(file.id, false);
    saving.value = true;

    let [name, content, path] = await window.electronAPI.saveFile({
      path: file.path,
      content: file.content.join('\r\n\n'),
    });

    saving.value = false;
    return [name, content, path];
  } catch (error) {
    console.error(error);
  }
}
```

Slika 21. Funkcije datoteke HomeView 1.dio

```
async function handleSaveAs() {
  const file = store.getters.getFile(currentTab.value.id);
  try {
    let [name, content, path] = await window.electronAPI.saveFile({
      path: undefined,
      content: file.content.join('\r\n\n'),
    });
    let data = {
      name,
      content: parseFile(content),
      path,
      id: Date.now(),
      unsaved: false,
    };
    files.value.push(data);
    store.commit('addFile', data);
    currentTab.value = files.value[files.value.length - 1];
  } catch (error) {
    console.error(error);
  }
}

async function handleNewFile() {
  try {
    let data = {
      id: Date.now(),
      name: 'untitled',
      content: [],
      path: undefined,
      unsaved: true,
    };

    files.value.push(data);
    store.commit('addFile', data);
    currentTab.value = files.value[files.value.length - 1];
  } catch (error) {
    if (error.name !== 'AbortError') {
      console.error(error);
    }
  }
}
```

Slika 22. Funkcije datoteke HomeView 2.dio

Pošto je za mogućnost uređivanja Markdown datoteka liniju po liniju potrebno od sadržaja učitane datoteke, koja je tipa string, stvoriti polje gdje je svaki element jedna linija datoteke, napisana je funkcija parseFile. Ona se sastoji od dva koraka. Prvo, parse funkcija iz npm paketa Marked pretvara markdown sadržaj datoteke u HTML string. Zatim se taj HTML sadržaj pretvara u HTML dokument instancom JavaScript klase DOMParser. Time se dobiva ispravno ugniježđena struktura datoteke kroz koju se može iterirati. U drugom koraku se DOMParser objekt pretvara u polje koje hijerarhijski dijeli dokument i zatim, pomoću npm paketa Turndown, svaka linija se pretvara nazad u Markdown. Kod cijele funkcije parseFile prikazan je na slici 23.

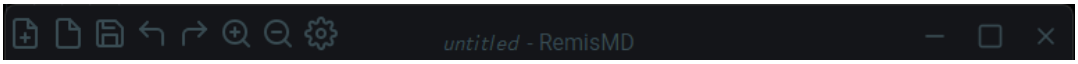
```
function parseFile(fileContent) {
  const td = new TurndownService({
    headingStyle: 'atx',
    codeBlockStyle: 'fenced',
  });
  td.use(gfm);
  const initialHtml = marked.parse(fileContent);

  const doc = new DOMParser().parseFromString(initialHtml, 'text/html');
  return [...doc.body.children].map((child) => td.turndown(child.outerHTML));
}
```

Slika 23. Funkcija parseFile

5.4. MenuBar.vue

Komponenta MenuBar.vue prikazuje i kontrolira traku izbornika na vrhu sučelja aplikacije, koja je prikazana na slici 24. Elementi trake su raspoređeni u tri skupine. Prva se sastoji od gumba koji su vezani za kontrolu rada aplikacijom, poput spremanja, otvaranja i stvaranja datoteka. Druga skupina prikazuje naslov dokumenta koji se trenutno uređuje. Treća skupina sadrži gumbe za minimiziranje. maksimiziranje i gašenje aplikacije.



Slika 24. Izbornička traka aplikacije

Klikom na gumbe za stvaranje, otvaranje i spremanje datoteke te za otvaranje postavki aktivira se Vue emit koji u roditelj-komponenti HomeView.vue poziva relevantnu funkciju. Klikom na gumbe za minimiziranje, maksimiziranje i gašenje aplikacije te za zumiranje, poništavanje i ponavljanje poziva se funkcija handleControl koja kao argument prima naziv procedure koju treba odraditi. Ona sadrži JavaScript switch koji, ovisno o argumentu, poziva relevantnu IPC funkciju. Komponenta također sadrži Vue watch funkciju, koja prati prop windowTitle kako bi se naslov dokumenta mogao dinamički mijenjati. Kod funkcija handleControl i watch prikazan je na slici 25.

```
function handleControl(control) {
  switch (control) {
    case 'close':
      window.electronAPI.closeApp();
      break;
    case 'minimize':
      window.electronAPI.minimizeApp();
      break;
    case 'maximize':
      window.electronAPI.maximizeApp();
      break;

    case 'zoomIn':
      window.electronAPI.zoomIn();
      break;
    case 'zoomOut':
      window.electronAPI.zoomOut();
      break;
    case 'undo':
      window.electronAPI.undo();
      break;
    case 'redo':
      window.electronAPI.redo();
      break;
  }
}

watch(
  () => props.windowTitle,
  () => {
    window.document.title = props.windowTitle || '';
  }
);
```

Slika 25. Funkcija handleControl

5.5. TabContainer.vue

Za upravljanje otvorenim datotekama zadužena je komponenta TabContainer.vue. Div element s klasom tabElement ima svrhu prikazivanja jedne kartice za svaku otvorenu datoteku. Pošto komponenta kao prop prima sve datoteke u varijabli files iz HomeView.vue, kroz njih se može iterirati koristeći v-for Vue direktivu te ih tako prikazati. Uz to, prikazuju se i ikone učitavanja, nespremljenih promjena i zatvaranja datoteke pomoću v-if direktive i relevantnih prop atributa. Vue kod ovih elemenata prikazan je na slici 26.

```
<div
  class="tabElement"
  draggable="true"
  v-if="props.tabs"
  v-for="tab in props.tabs"
  :key="tab.id"
  :class="tab.id == props.currentTab ? 'activeTab' : 'inactiveTab'"
  @click="$emit('setCurrentTab', tab.id)"
>
  {{ tab.name }}
  <div
    class="loading"
    v-if="props.saving && tab.id == props.currentTab"
  ></div>
  <div class="unsaved" v-if="!props.saving && tab.unsaved"></div>
  <svg
    v-if="!props.saving"
    viewBox="0 0 24 24"
    width="23"
    height="23"
    stroke="currentColor"
    stroke-width="2"
    fill="none"
    stroke-linecap="round"
    stroke-linejoin="round"
    class="closeIcon"
    @click.stop="$emit('closeTab', tab.id)"
  >
    <line x1="18" y1="6" x2="6" y2="18"></line>
    <line x1="6" y1="6" x2="18" y2="18"></line>
  </svg>
</div>
```

Slika 26. Template dio komponente TabContainer

5.6. EditorComponent.vue

EditorComponent.vue komponenta služi za uređivanje markdown datoteke koju prima kao prop atribut. Pri postavljanju komponente, aktivira se JavaScript event listener koji prati korisnikov unos tipkovnice te ukoliko je unio tipkovnički prečac poziva relevantnu funkciju. JavaScript kod za event-listener prikazan je na slici 27.

```
window.addEventListener('keyup', (key) => {
  if (key.key === 'Escape') {
    selectedLine.value = null;
  } else if ((key.key === 's' || key.key === 'S') && key.ctrlKey) {
    if (key.shiftKey) {
      emit('saveAs');
    } else {
      emit('saveFile');
    }
  } else if (key.key === 'ArrowDown' && key.ctrlKey) {
    selectedLine.value += 1;
  } else if (key.key === 'ArrowUp' && key.ctrlKey) {
    selectedLine.value -= 1;
  } else if (key.key === 'ArrowRight' && selectedLine.value === null) {
    selectedLine.value = 0;
  } else if (key.key === 'Delete' && key.ctrlKey && key.shiftKey) {
    removeLine(selectedLine.value);
  } else if (
    key.key === 'Enter' &&
    key.ctrlKey &&
    selectedLine.value !== null
  ) {
    if (key.shiftKey) {
      addLine(selectedLine.value);
    } else {
      addLine(selectedLine.value + 1);
    }
  }
});
```

Slika 27. Event-listener funkcije u EditorComponent.vue

Ova komponenta unutar HTML dijela sadrži div element koji se, pomoću v-for direktive, ponavlja za svaku liniju u odabranom markdown dokumentu. Ovaj element okružuje komponentu TextArea.vue, koja kao prop atribut prima jednu liniju dokumenta te služi za uređivanje i prikaz iste. Osim glavnog div elementa, postoji i dodatni element s klasom newLine koji, pri korisnikovom kliku na njega, dodaje novu liniju u otvoreni dokument. Cijeli kod template dijela komponente je prikazan na slici 28.

Slika 28. Template dio komponente EditorComponent

5.7. TextArea.vue

TextArea.vue komponenta koristi se za prikaz i uređivanje svake linije datoteke posebno. Promjenom varijable editMode kontrolira se stanje linije, odnosno, uređuje li se ili prikazuje u HTML formatu. Kada editMode ima istinitu vrijednost prikazuje se HTML textarea oznaka koja korisniku omogućuje unos i uređivanje sadržaja te se pomoću emit funkcije taj sadržaj mijenja i u komponenti EditorComponent.vue, što je prikazano na slici 29. Ukoliko editMode ima neistinitu vrijednost, prikazuje se div element koji pomoću v-html direktive prikazuje HTML u varijabli renderLine. Ova varijabla, čiji je kod prikazan na slici 30, sadrži computed svojstvo koje, koristeći parse funkciju iz npm paketa Marked, automatski pretvara Markdown string u HTML.

```
<template>
  <div v-if="editMode">
    <textarea
      v-model="lineValue"
      cols="30"
      :rows="rowNumber"
      :ref="controlTextArea"
      @input="$emit('update:modelValue', $event.target.value)"
      autocomplete="off"
      wrap="hard"
    ></textarea>
    <div class="textAreaControl">…
    </div>
  </div>
  <div
    v-else
    v-html="renderLine"
    @click="$emit('selectLine')"
    class="render"
  ></div>
</template>
```

Slika 29. Template dio komponente TextArea

```
const renderLine = computed(() => marked.parse(lineValue.value));
```

Slika 30. Varijabla renderLine

5.8. DefaultScreen.vue

DefaultScreen.vue komponenta se prikazuje kad korisnik nije otvorio niti jednu markdown datoteku. Sastavljena je od dva div elementa koji nakon pritiska šalju emit u roditelj-komponentu HomeView.vue, za stvaranje nove datoteke ili za otvaranje datoteke. Kod za HTML dio komponente prikazan je na slici 31.

```
<template>
  <div class="container">
    <div class="box" @click="$emit('newFile')">
      <svg…
      </svg>
      <p>Create new file</p>
    </div>
    <div class="box" @click="$emit('openFile')">
      <svg…
      </svg>
      <p>Open existing file</p>
    </div>
  </div>
</template>
```

Slika 31. Template dio komponente DefaultScreen

5.9. SettingsComponent.vue

SettingsComponent.vue komponenta prikazuje postavke aplikacije koje korisnik može kontrolirati. Prikazuje se preko cijelog sučelja aplikacije koristeći Vue Teleport element, koji komponentu učitava izvan glavnog div elementa aplikacije. Komponenta se sastoji od dijela za odabir teme sučelja, dijela postavke prozora aplikacije te gumba za otvaranje JSON datoteke u kojoj korisnik može kreirati vlastite teme. Kod za HTML dio komponente prikazan je na slici 32.

```
<template>
  <div class="bg" v-if="loaded">
    <div class="modal">
      <h1>Settings</h1>
      <h2>Theme</h2>
      <div class="selectTheme">
        <div
          class="theme"
          v-for="(theme, name) in allThemes"
          :key="theme"
          :class="{ selectedTheme: currentTheme.theme == name }"
          @click="
            setAll();
            $emit('setCurrentTheme', [theme, name]);
          "
        >
          {{ name }}
        </div>
      </div>
      <h2>Window settings</h2>
      <div class="windowSettings">
        <label class="form-control">
          <input
            type="checkbox"
            name="checkbox"
            :checked="settings.rememberWindowSize"
            @click="
              updateSettings('rememberWindowSize', !settings.rememberWindowSize)
            "
          />
          Remember window size
        </label>
      </div>
      <a class="openConfig" @click="openConfig" title="Create custom themes"
        >Open config.json</a
      >
      <div class="buttons">
        <button @click="$emit('closeModal')">Close</button>
      </div>
    </div>
  </div>
</template>
```

Slika 32. Template dio komponente SettingsComponent

Pri učitavanju komponente poziva se funkcija setAll. Ona prvo poziva dvije IPC funkcije: getAllThemes i getAllSettings te njihove povratne vrijednosti sprema u varijable. Ove dvije funkcije dohvaćaju sve teme i postavke koje se nalaze u config.json datoteci koja služi za trajno spremanje korisnikovih podataka. Nakon toga funkcija

getCurrentTheme pomoću istoimene IPC funkcije dohvaća trenutnu korisnikovu temu sučelja. Komponenta ima i dodatne dvije funkcije: updateSettings i openConfig. Prva koristi istoimenu IPC funkciju te tako ažurira promijenjene postavke u trajnoj pohrani, pa zatim poziva funkciju setAll. Druga funkcija koristi istoimenu IPC funkciju kako bi otvorila config.json datoteku u korisnikovom preferiranom programu. Vue kod ovih funkcija prikazan je na slici 33.

```
async function getCurrentTheme() {
  currentTheme.value = await window.electronAPI.getCurrentTheme();
}

async function setAll() {
  try {
    allThemes.value = await window.electronAPI.getAllThemes();
    settings.value = await window.electronAPI.getAllSettings();
    await getCurrentTheme();
  } catch (error) {
    console.error(error);
  }
}

function openConfig() {
  window.electronAPI.openConfig();
}

async function updateSettings(setting, value) {
  try {
    await window.electronAPI.updateSettings(setting, value);
    await setAll();
  } catch (error) {
    console.error(error);
  }
}

onMounted(async () => {
  await setAll();
  loaded.value = true;
});
```

Slika 33. Funkcije komponente SettingsComponent

6. ZAKLJUČAK

U ovom radu opisana je izrada desktop aplikacije za stvaranje i uređivanje Markdown dokumenata, nazvana RemisMD. Iako postoje razni programi za tu svrhu, ova aplikacija morala je zadovoljiti nekoliko zadanih uvjeta. Aplikacija je morala imati minimalno sučelje s mogućnošću podešavanja njegovih boja kroz prilagođene teme. To je bilo postignuto korištenjem vanjske JSON datoteke, koju korisnik uređuje kako bi prilagodio sučelje.

Osim toga, aplikacija je morala prikazivati Markdown dokument formatiran u obliku HTML-a, čije se uređivanje provodi liniju po liniju te se izvršava u izvornom, tekstualnom obliku datoteke. Ova funkcionalnost implementirana je jednostavno i efikasno, omogućavajući korisniku potpunu kontrolu nad linijom koju uređuje, bez potrebe za prikaz ostatka dokumenta u izvornom obliku. Markdown specifikacija koja je morala biti podržana je GitHub Flavoured Markdown, što je izvedeno koristeći Node.js biblioteku Marked. Kako bi korištenje aplikacije bilo intuitivno, moraju postojati tipkovnički prečaci za često korištene funkcionalnosti, što je također implementirano u krajnjem proizvodu.

Iako je RemisMD potpun proizvod, koji se može koristiti kao glavni uređivač Markdown datoteka, postoje mnoge mogućnosti za poboljšanje. Boje sučelja su prilagodljive, ali one ovise o samo osam varijabli, što ograničava paletu boja svake teme. Drugo bitno ograničenje je raspored sučelja. Dijelove sučelja nije moguće premjestiti stoga su korisnici ograničeni na jedan dizajn. Još jedna značajka koja bi poboljšala korisničko iskustvo bila bi mogućnost spremanja formatiranih datoteka u HTML ili PDF format.

Aplikacija je izrađena koristeći web tehnologije HTML, CSS i JavaScript. To je moguće uz pomoć razvojnog okvira Electron, koji koristi prilagođenu verziju Chromium web preglednika kako bi bilo koju web stranicu pretvorio u samostalan računalni program. Za stvaranje korisničkog sučelja korišten je JavaScript razvojni okvir Vue.js, koji proširuje HTML, CSS i JavaScript te pruža deklarativni model programiranja baziran na komponentama. Vue.js se kroz ovaj rad pokazao kao jednostavno i potpuno rješenje za izradu web i desktop aplikacija. Sva ažuriranja aplikacije vrše se automatski i u pozadini, koristeći distribucije objavljene na GitHub repozitoriju projekta.

LITERATURA

Ecma International. (16. September 2022). Dohvaćeno iz https://www.ecma-international.org/

Electron JS. (16. September 2022). Dohvaćeno iz https://www.electronjs.org/docs/latest/tutorial/process-model

Electron-store GitHub. (2022). Dohvaćeno iz GitHub: https://github.com/sindresorhus/electron-store

Electron-updater GitHub. (2022). Dohvaćeno iz GitHub: https://github.com/electron-userland/electron-builder

Ghostwriter website. (2022). Dohvaćeno iz https://wereturtle.github.io/ghostwriter/

Gruber, J. (17. December 2004). Markdown. Dohvaćeno iz Daring Fireball: https://daringfireball.net/projects/markdown/

Marked.js. (2022). Dohvaćeno iz https://marked.js.org/

MarkText GitHub. (2022). Dohvaćeno iz GitHub: https://github.com/marktext/marktext

Mozilla Developer Network. (2022). Preuzeto 16. September 2022 iz https://developer.mozilla.org/

Node.js. (16. September 2022). Dohvaćeno iz https://nodejs.org/

npm. (16. September 2022). Dohvaćeno iz https://www.npmjs.com/

Sass. (16. September 2022). Dohvaćeno iz https://sass-lang.com/

Stack Overlfow Developer Survey. (16. September 2022). Dohvaćeno iz Stack Overflow: https://survey.stackoverflow.co/2022/

Turndown GitHub. (2022). Dohvaćeno iz GitHub: https://github.com/domchristie/turndown

Visual Studio Code. (2022). Dohvaćeno iz https://code.visualstudio.com

Vue.js. (16. September 2022). Dohvaćeno iz https://vuejs.org/

W3Techs. (16. September 2022). Dohvaćeno iz https://w3techs.com/technologies/details/cp-javascript/

POPIS SLIKA