IZRADA VUE.JS WEB APLIKACIJE ZA OBLIKOVANJE I IZRADU PECS KARTICA

Bebić, Igor

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

IGOR BEBIĆ

Z A V R Š N I R A D

IZRADA VUE.JS WEB APLIKACIJE ZA OBLIKOVANJE I ISPIS PECS KARTICA

Split, rujan 2019.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Predmet: Oblikovanje web stranica

Z A V R Š N I R A D

Kandidat: Igor Bebić

Naslov rada: Izrada Vue.js web aplikacije za oblikovanje i izradu PECS kartica

Mentor: Haidi Božiković, viši predavač

Split, rujan 2019.

Sadržaj

SAŽETAK

Cilj ovog rada je izraditi aplikaciju za jednostavnu izradu, dijeljenje i ispis kolekcija PECS kartica namijenjenih za korištenje u komunikaciji s djecom i osobama koje imaju poteškoće sa standardnim oblicima komunikacije.

Postojeći alati za izradu PECS kartica su preskupi, zastarjeli i neintuitivni te je dijeljenje kolekcija preko njih u većini slučajeva nemoguće. Radi toga korisnici tih aplikacija pojedinačno troše previše vremena na izradu kolekcija istih tema, umjesto da koriste postojeće kolekcije i fokusiraju se na izradu novih, jedinstvenih kolekcija. PECS *card maker* aplikacija bi uz minimalnu obuku trebala omogućiti defektolozima, pedagozima i obrazovnim rehabilitatorima jednostavnu izradu kolekcija, te dijeljenje tih kolekcija s kolegama kroz aplikaciju.

Dolaskom na stranicu i jednostavnom registracijom preko Google računa, korisnicima se daje pristup javno objavljenim kolekcijama, te im se pruža mogućnost da sami kreiraju i objave svoje kolekcije, kao i da kopiraju postojeće i prilagode ih svojim potrebama. Administratori web aplikacije kontroliraju sadržaj kolekcija te ovisno o prikladnosti odobravaju ili zabranjuju javno objavljivanje, uz slanje tekstualnog obrazloženja autoru kolekcije.

Također, korisnici aplikacije mogu ostavljati komentare na javno dostupne kolekcije, te time bolje informirati druge korisnike aplikacije o njenoj kvaliteti i eventualnim nedostacima.

Ključne riječi: PECS, kartice, kolekcije, edukacija, rehabilitacija

SUMMARY

Development of *Vue.js* application for PECS card creation and printout

The purpose of this work is to make an application for simple creation, sharing and printing of the PECS card collection, whose purpose is to help in communication with children and adults who are having problems with standard forms of communication.

Existing tools for PECS card creation are expensive, obsolete and unintuitive, and sharing collections between them is almost impossible. Because of that, users of those applications are individually spending too much time to make collection which someone has already made, instead of creating something new and unique. PECS card maker application should, with minimal training, enable pedagogues, education rehabilitators and defectologists to simply create PECS collection, as well as share them with their colleagues through the application.

Once on the web page, users are registered via the Google account, and given access to publicly available collections. They are also given the tools to create their own collections and publish them, as well as to copy existing collections and adapt them to their needs. Web application administrators control the collections content and, depending on it, approve or deny publishing, while sending a message to the author regarding the reasons for the rejection.

User can also leave comment on published collections, which helps inform other users about the quality and potential imperfections of the collection in question.

Keywords: PECS, cards, collections, education, rehabilitation

1. UVOD

Način na koji se web aplikacije razvijaju je posljednjih godina doživio velike promjene. Alati za izradu korisničkog i poslužiteljskog dijela aplikacije danas su pristupačniji nego ikada, što omogućava i nestručnim korisnicima da u relativno kratkom roku steknu znanje potrebno za izradu estetski ugodnih i responzivnih statičkih web stranica, pa čak i jednostavnijih web aplikacija s kompletnom poslužiteljskom stranom.

Cilj ovog rada je kroz izradu web aplikacije sa svojom korisničkom i poslužiteljskom stranom prikazati glavne značajke i prednosti novih alata, a u prvom planu *Vue.js-a*. To će biti prikazano kroz primjer aplikacije za izradu i dijeljenje kolekcija PECS kartica (sustav komuniciranja razmjenom slike).

PECS je sustav komunikacije kojim se pomoću slika razvijaju komunikacijske sposobnosti djece i osoba s komunikacijskim poteškoćama, kao i poteškoćama s učenjem. Svaka PECS kartica sastoji se od ilustracije i teksta na dnu koji opisuje ilustraciju.

Trenutno dostupni alati za izradu kartica su zastarjeli, skupi i nepotrebno komplicirani za korištenje. Također, dijeljenje kolekcija kartica je otežano radi nekompatibilnosti među navedenim aplikacijama, kao i činjenici da većina aplikacija radi isključivo u izvanmrežnom načinu.

Web aplikacija za izradu PECS kolekcija omogućiti će jednostavnu izradu PECS kartica, njihovo preuzimanje u .pdf formatu koji je spreman za izrezivanje, kao i dijeljenje kolekcija među korisnicima. Sve to biti će nadgledano od strane administratora koji će imati mogućnost zabrane dijeljenja kolekcija s neprimjerenim sadržajem kao i dodjeljivanje i uklanjanje administratorskih prava drugim korisnicima.

U idućim poglavljima prikazane su i opisane tehnologije korištene pri izradi aplikacije, kao i odabrano razvojno okruženje. Također, prikazan je način rada poslužiteljske i klijentske aplikacije, te njihova međusobna komunikacija, kao i upravljanje poslužitelja podacima u bazi podataka.

2. KORIŠTENE TEHNOLOGIJE

2.1. JavaScript

JavaScript je trenutno najkorišteniji programski jezik na svijetu. Originalno zamišljen isključivo kao skriptni jezik koji se pokreće na korisničkoj strani u sklopu HTML stranica, često je stigmatiziran i smatran neozbiljnim u usporedbi s drugim jezicima. Međutim, danas se način njegovog korištenja uvelike promijenio.

JavaScript se osim za pisanje kôda koji se pokreće na korisničkoj strani, koristi i za pisanje poslužiteljskog kôda, kako kod malih tako i kod velikih aplikacija kojima pristupa više tisuća korisnika. Streloviti rast potražnje za programerima koji znaju pisati JavaScript pokazuje da se njegova iznimna važnost u današnjem svijetu ne može zanemariti.

JavaScript je razvijen od strane Brendana Elcha 1995. godine pod imenom *LiveScript,* da bi se kasnije preimenovao u JavaScript. Razlog za to je bio isključivo marketinške prirode, s obzirom da je u to vrijeme Java bila iznimno popularna. Službeno je standardiziran 1997. godine od strane *ECMA* organizacije pod imenom *ECMA-262* ili *ECMAScript.* 1998. godine izdan je *ECMAScript* 2, a već 1999. *ECMAScript* 3. Četvrta verzija *ECMAScripta* je zbog neslaganja razvojnih timova o kompleksnosti napuštena, te je daljni razvoj nastavljen koristeći *ECMAScript 3.1* verziju, koju je u to vrijeme razvijao Microsoft. Ta verzija je preimenovana u *ECMAScript 5* i temelj je svih budućih verzija, uključujući i trenutno najnoviju, *ECMAScript 2018*.

JavaScript je višeparadigmatski, dinamički jezik s tipovima i operatorima, standardnim ugrađenim objektima i metodama. Sintaksa se temelji na Java i C programskim jezicima a mnoge strukture iz tih jezika također vrijede i kod JavaScripta. JavaScript podržava objektno orijentirano programiranje preko objektnih prototipova umjesto uobičajenih klasa. Osim toga, JavaScript podržava funkcionalni pristup programiranju. Funkcije su ujedno i objekti, pa se kao takvi mogu prosljeđivati u druge objekte.

JavaScript dinamički pridjeljuje tipove, što znači da je tip vezan uz vrijednost a ne uz sam izraz. Drugim riječima, u JavaScriptu neka varijabla koja je bila vezana za *Number* ili *Boolean*, može lako kasnije postavi vezana uz *String*. Tipovi koji postoje u JavaScriptu su:

Boolean, *Number*, *String*, *Symbol*, *Object*, *null* i *undefined*, dok se objekti još mogu dijeliti na funkcije, nizove, *Date* objekt te *RegExp*.

2.2. Node.js

Node.js je *multiplatformno* JavaScript okruženje otvorenog kôda koje omogućuje izvršavanje JavaScript kôda na poslužiteljskoj strani. Radi na Google V8 *engineu* koje je napisan u C++ jeziku, što omogućava jako brz rad.

Programiranje za Node.js se obavlja sinkrono, što znači da dok se jedna linija kôda izvršava, sustav čeka rezultat. Primitkom rezultata, sustav ga procesuira i nastavlja izvršavanje iduće linije kôda. Taj način izvršavanja naredbi nije idealan, pogotovo u slučajevima kada se određeni dio kôda izvršava dugo vremena. Ovakvi problemi se u jezicima kao što su C# i Java rješavaju uvođenjem novih niti izvršavanja, pa se nekoliko zadataka može paralelno izvršavati, bez da utječu jedni na druge. Iako to u teoriji rješava probleme čekanja, programiranje za takve sustave treba odrađivati s mnogo pažnje, jer u protivnom mogu uzrokovati nove vrste problema. Prvenstveno je potrebno voditi računa o tome da dvije niti izvršavanja ne smiju nikada pristupati i raditi na istim resursima u isto vrijeme.

Programiranjem u Node.js-u tih problema nema radi potpuno drukčijeg načina rada JavaScripta. U njemu se i dalje uvijek izvršava samo jedna nit, međutim, kod sporih operacija kao što je čitanje podataka iz baze, program ne čeka na izvršavanje nego ide na izvršavanje slijedeće linije. Tek kada se podatak uspješno dohvati iz baze, pokreće se povratna funkcija i rezultat se dalje procesuira.

Pri programiranju u Node.js-u teži se kreiranju malih modula koji samostalno odrađuju samo jednu funkciju. Prednost tog pristupa je mogućnost ponovnog korištenja tih modula kroz aplikaciju, ali i sama jednostavnost pri razvoju aplikacija. Kôd je lakše čitljiv i održiv, a podizanje složenosti aplikacije ostvaruje se dodavanjem novih modula.

Jedan od najbitnijih prednosti modularnog načina pisanja kôda je u velikoj programerskoj zajednici koja piše i daje na korištenje stotine milijuna malih modula dostupnih preko *npm* alata (*Node Package Manager*). Na taj se način programer pri izradi aplikacije može fokusirati samo na onaj dio aplikacije koji je specifičan samo za sebe. Svi ostali problemi općenite prirode, kao što su obrađivanje formata datuma, kompresija,

prebacivanje podataka iz jednog tipa u drugi i tome slično, već su riješeni, nerijetko na više načina i dostupni kroz spomenute module.

2.3. Vue.js

Vue.js je JavaScript programski okvir koji služi za izgradnju korisničkog sučelja. Uz *React* i *Angular* spada među najpopularnije JavaScript okvire. Tome u prilog ide činjenica da su neki od tvrtki koje ga koriste i financijski podržavaju *Alibaba, Facebook, Netflix, Xiaomi, Adobe* i mnoge druge.

Ideja iza *Vue.js-a* je pojednostavniti i bolje organizirati programiranje za web. Glavni koncept iza *Vue.js-a* je podjela i izrada sučelja na komponente, te osvježavanje korisničkog sučelja nakon promjena u realnom vremena, i to na način da se osvježavaju samo one komponente koje to zahtijevaju na temelju novih podataka.

S obzirom da korištenje *Vue.js-a* nije ni na koji način restriktivno po pitanju već postojećih tehnologija kojima su do sada rađene internetske stranice, vrlo ga je lako postepeno uvoditi u već postojeće internetske stranice.

Jedna od glavnih značajki *Vue.js-a* je u korištenju predložaka za oblikovanje stranice. Predlošci u *Vue.js-u* su bazirani na HTML-u, uz potrebne dodatke koji omogućavaju povezivanje DOM-a (objektni model HTML-a) s podacima *Vue.js* instance.

Druga i već spomenuta bitna značajka je reaktivnost. Pri promjeni podataka unutar *Vue.js* instance, po potrebi se mijenja i osvježava samo onaj dio predložaka na koje ta promjena utječe. Takav način rada znatno poboljšava sveukupne performanse rada stranice.

Treća značajna i prednost *Vue.js* okvira je u korištenju komponenti. Iz programerskog aspekta, ona je vjerojatno i najbitnija, jer uvelike olakšava proces izrade stranice i organizaciju kôda. Svaka komponenta u sebi sadrži tri osnovna dijela:

1. Predložak temeljen na HTML-u koji služi za organizaciju i raspored pojedinih elemenata
2. JavaScript kôd, koji kontrolira promjenu podataka na predlošku
3. CSS dio, koji služi za detaljno uređivanje i oblikovanje HTML-a iz predloška

Ovako organizirana, svaka komponenta je cjelina sama za sebe, te se kao takva može vrlo jednostavno premještati ili koristiti više puta na više mjesta unutar aplikacije, pa i unutar drugih komponenti.

Usmjeravanje je također jedna od iznimno moćnih prednosti *Vue.js-a*. To je pogotovo očito pri izradi jednostraničnih aplikacija. Kod takvih aplikacija je do sada bilo nemoguće dijeliti određeni dio aplikacije, odnosno njenu pod stranicu. *Vue.js* taj problem rješava uvođenjem umjetnih ruta, kojima se pristupa isto kao i pravim rutama. To omogućuje korisnicima aplikacije da, ako žele, mogu dijeliti određeni dio stranice kao da to rade kod klasičnih aplikacija, koristeći dodatke kao što su *„/users"* i slično.

2.4. PostgreSQL / Sequelize

PostgreSQL je sustav za upravljanje relacijskim bazama podataka otvorenog kôda. To je objektno relacijska baza što znači da se svi podaci spremaju u odvojene tablice, koje su međusobno povezane definiranim relacijama. Ta struktura istovremeno omogućuje veliku brzinu pretraživanja baze i fleksibilnost.

PostgreSQL se temelji na modelu klijent-poslužitelj, gdje su te dvije strane međusobno povezane i komuniciraju preko TCP/IP sučelja. *PostgreSQL* je iznimno fleksibilan, te se kao takav koristi kako kod malih i kod velikih aplikacija koje istovremeno poslužuju velik broj korisnika.

Osnovni dio *PostgreSQL* baze je shema. Svaka novoizrađena baza ima shemu imena *default*, ali korisnik može dodati druge sheme, jer ta inicijalna nije obvezna. PostgreSQL baza podržava veliki broj tipova podataka, kao što su: *Boolean*, *Character*, *Binary*, *Date/time*, *Enum*, *Money*, *Bit strings*, *Text*, *Composite*, JSON, JSONB, itd.

Korisnik pristupa *PostgreSQL* bazi podataka uz pomoć neke od dostupnih sučelja i platformi. Neke od najkorištenijih su *pgAdmin*, koji je besplatan, te *Postico* čija je besplatna verzija dovoljna za osnovne administrativne radove.

Iako korištenje navedenih platformi znatno olakšava rukovanje *PostgreSQL* bazom, za napredne funkcije i pretraživanja unutar aplikacija, u današnje vrijeme je neophodno korištenje ORM-a ili objektno relacijskog *mappera*. ORM omogućava pisanje kôda za dohvaćanje i promjenu podataka u bazi korištenjem jezika poslužiteljske strane aplikacije.

Drugim riječima, omogućava da se u *Node.js* aplikaciji upiti na bazu pišu JavaScriptom umjesto standardnom SQL skriptom. To dozvoljava da se klasična SQL skripta (Ispis 1) piše na čitkiji način i u JavaScriptu (Ispis 2).

Ispis 1: Primjer SQL skripte

```
SELECT * FROM USERS WHERE first_name = "Marko";
```

Ispis 2: Primjer Sequelize kôda

```
var users = User.findAll({ where: { firstName: 'Marko' } });
```

Sequelize je jedan od najpoznatijih ORM-ova, kako za *PostgreSQL*, tako i za mnoge druge tipove baza (*MySQL*, *SQLite*, Microsoft SQL, itd.). Ta činjenica olakšava programerima prelazak s jedne vrste baze na drugu jer pri tome nema potrebe za ponovno pisanje kôda za upravljanjem. *Sequelize* sadrži velik set alata koji znatno olakšavaju, i na neki način proširuju mogućnosti pri rukovanju bazom podataka. Neke od tih mogućnosti su sinkronizacija, validacija, *lazy-loading*, kontrola toka bazirana na *Promise*-ima itd.

2.5. Auth0

Jedan od većih problema pri izradi naprednije web aplikacije koja implicira izmjenu podataka od strane korisnika je autentifikacija. Ona nam omogućava identifikaciju korisnika te u skladu s tim, dozvolu i zabranu pristupa samoj aplikaciji. Sam proces autentifikacije nužno je odraditi na poslužiteljskoj strani, da bi izbjegli ikakvu mogućnost manipulacije autentifikacijskom logikom.

U novije vrijeme, programeri za autentifikaciju sve češće koriste vanjske autentifikacijske servise, kao što su Google, Facebook, Twitter i drugi. Dolaskom na aplikaciju, korisnika se preusmjerava na stranicu za prijavu na neki od navedenih servisa. Nakon uspješne prijave, i odobrenja od strane korisnika da aplikacija koristi korisničke podatke s tog servisa, servis aplikaciji šalje autorizacijski kôd. Aplikacija taj kôd koristi zajedno sa identifikatorom i tajnim ključem da bi zatražila pristupni token. Pristupni token se nadalje koristi za pristup korisničkim podacima. Koristeći ovaj oblik autentifikacije,

korisnik ne mora pamtiti još jedno korisničko ime i lozinku, a web aplikacija ih ne mora pohranjivati u svoju bazu.

Auth0 je platforma za univerzalnu autentifikaciju i autorizaciju, namijenjena korištenju na web, mobilnim i *legacy* aplikacijama. Omogućuje developerima jednostavnu implementaciju autorizacije korisnika korištenjem postojećih profila s popularnih društvenih mreža.

Da bi pristupio web aplikaciji, korisnik ne mora prolaziti kroz proces registracije, već za autorizaciju i autentifikaciju može koristiti svoj Google, Facebook, Twitter, Yahoo ili bilo koji od računa poznatijih društvenih mreža. Na taj način se znatno umanjuje šansa za krađom lozinke, jer se u ovom slučaju lozinka ne pohranjuje u bazi same aplikacije, već se aplikacija oslanja na Google, Facebook ili neki treći servis da bi dobila informaciju od identitetu korisnika.

Osim korištenja postojećih servisa, *Auth0* nudi i opciju registracije korisnika u slučaju da osoba koja želi koristiti aplikaciju nema račun ni na jednom od podržanih društvenih platformi. No ni u tom slučaju se lozinka ne pohranjuje u bazi same aplikacije već je pohranjuje sam *Auth0* servis u svojoj bazi.

Implementacija *Auth0* autentifikacije je relativno jednostavna, i započinje registracijom na *Auth0* platformu. Besplatna verzija *Auth0* računa podržava maksimalno 7000 aktivnih korisnika, što je više nego dovoljno za aplikaciju u povojima. Nakon registracije je potrebno konfigurirati postavke: ime aplikacije, vrsta, korištena domena, podržani servisi za autentifikaciju i slično. Da bi proces autentifikacije bio što jednostavniji i čišći za krajnjeg korisnika, od podržanih društvenih mreža odabrao sam samo Google. Po završetku konfiguracije, *Auth0* platforma generira klijentski ID i certifikat, koji su potrebni za implementaciju autorizacije u aplikaciji.

Da bi implementacija autorizacije bila još bezbolnija, korišten je *npm* paket *„auth0-js”*, službeni set alata za komunikaciju s *Auth0* platformom, te „*Passport*”, autentifikacijski *middleware* za Node.js. Kad korisnik posjeti web stranicu, to pokreće funkciju autorizacije. Funkcija pristupa lokalnoj pohrani (*local storageu)* web preglednika u potrazi za Auth0 pristupnim tokenom. Ukoliko token postoji i ukoliko nije istekao, funkcija očitava korisničke podatke koji su također u *local storageu* te propušta korisnika na aplikaciju. Ako token ne postoji, funkcija preusmjerava korisnika sa web aplikacije na login stranicu.

Sama login stranica nije dio web aplikacije, već se nalazi na *Auth0* domeni. Njen izgled i funkcionalnost se konfiguriraju preko *Auth0* kontrolnog panela. Na login stranici, korisnik bira način autentifikacije, ili u slučaju *PECS card maker* aplikacije, Google. Odabirom Googlea, pokreće se standardna Google login forma. Korisnik odabire profil koje želi koristiti za autentifikaciju i nastavlja s autentifikacijom. *Auth0* platforma tada generira autorizacijski token te ga prosljeđuje natrag u web aplikaciju koja taj token pohranjuje u *local storage* te mu definira rok trajanja.

Osim tokena, u *local storage* se prosljeđuju te pod „user" ključem pohranjuju podaci o korisniku. To su osnovni podaci s Google društvene platforme, te sadrže jedinstveni *google-oauth2* ključ, email, ime, prezime i slično. Navedeni *google-oauth0* ključ se koristi u aplikaciji za identifikaciju korisnika i definiranje vlasništva nad PECS kolekcijama. Svakom autentifikacijom korisnika, na poslužiteljskoj se strani aplikacije u bazi podataka traži navedeni korisnik i ukoliko korisnik nije u bazi, dodaje se.

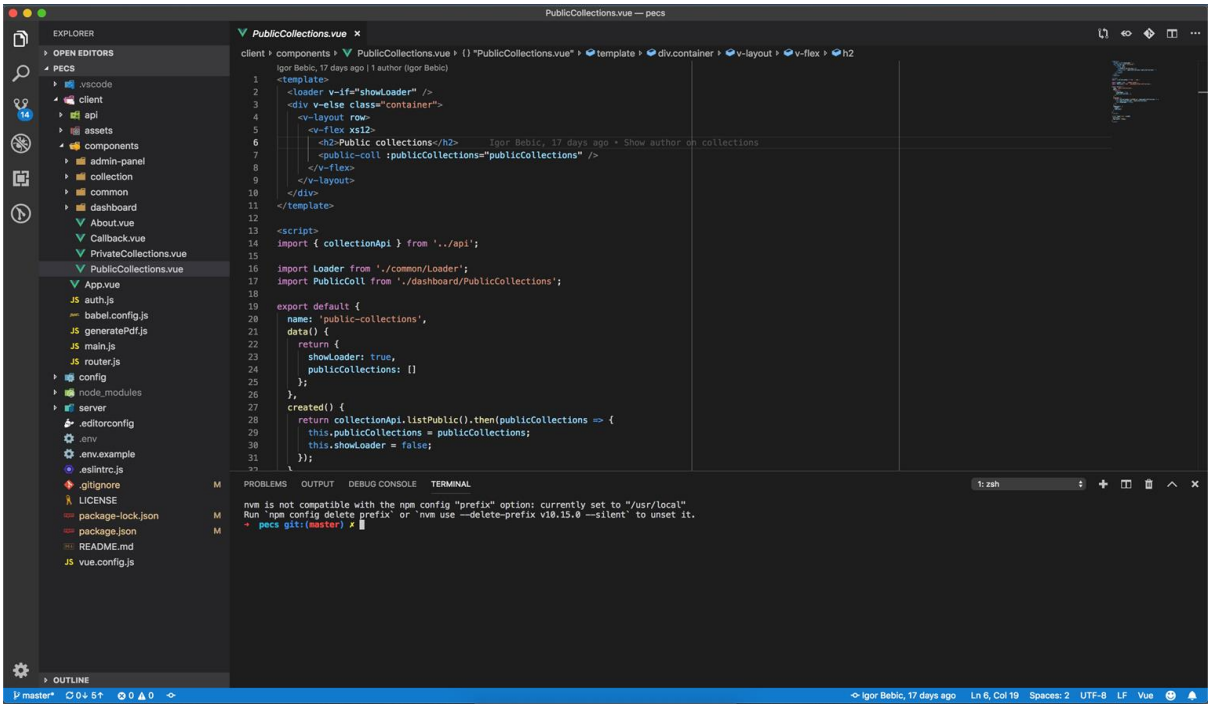
2.6. Razvojno okruženje

Odabir razvojne okoline bitan je aspekt pri razvoju aplikacije. Pri odabiru je potrebno definirati zahtjeve projekta te u skladu s njima odabrati rješenje koje nudi sve ono što programeru treba, a sa što manje popratnih elemenata koji nisu potrebni za taj projekt, a usporavaju i kompliciraju proces razvoja aplikacije.

S obzirom da su i poslužiteljska i klijentska strana PECS aplikacije rađena u JavaScriptu, odabir idealnog okruženja je nešto jednostavniji, jer su potrebni elementi okruženja u oba slučaja jednaki. U velikoj ponudi pretežito besplatnih rješenja koja variraju od krajnje minimalističkih do velikih detaljnih aplikacija sa širokim spektrom mogućnosti prilagodljivih za razvoj aplikacija u bilo kojem programskom jeziku, *Visual Studio Code* (*VSCode*) pokazao se kao idealno rješenje.

VSCode razvijen je od strane Microsofta, te je dostupan za besplatno preuzimanje za Windows, macOs i Linux operativne sustave. Jedna zanimljivost je da se temelji na Electronu, koji je razvijen koristeći *Node.js*. Iako to u teoriji znači da nije brz ni kompaktan kao neki drugi alati (npr. *Sublime*), na modernijim računalima je razlika u brzini praktički neprimjetna. *VSCode* je dostupan pod MIT licencom što znači da je njegov izvorni kôd u potpunosti dostupan široj javnosti.

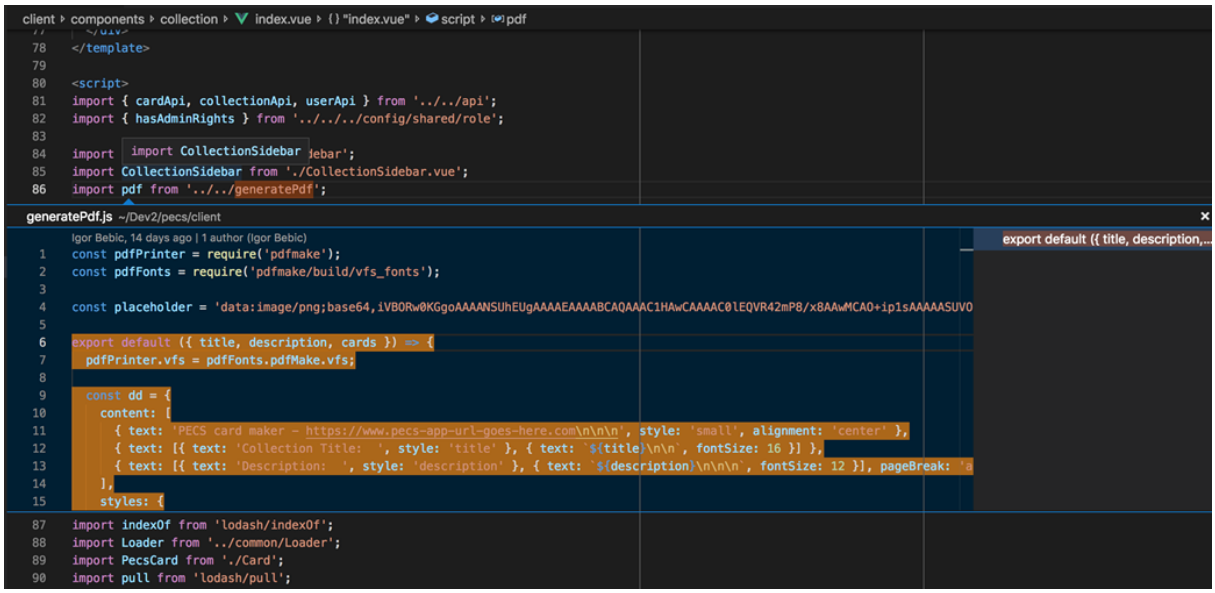
VSCode je poprilično minimalistički nastrojen, te se sastoji od tri osnovna dijela: središnji dio na kojem se nalazi uređivač kôda, bočni izbornik koji se koristi za navigaciju kroz datoteke aplikacije, pronalaženje grešaka u kôdu i kontrolu verzioniranja, te terminal na dnu (Slika 1). Sama konfiguracija postavki *VSCodea* se odrađuje preko JSON datoteka kojima se pristupa preko izbornika. Iako taj proces inicijalno nije osobito intuitivan, zahvalno je rješenje jer olakšava prebacivanje konfiguracije na druga računala jednom kada se sve postavi po želji, neovisno o operativnom sustavu.



Slika 1: Visual Studio Code

Jedno od glavnih prednosti *VSCodea* je i njegov *Marketplace*. Na njemu je moguće pronaći dodatke za aplikaciju koji su sposobni inicijalno minimalistički alat pretvoriti u potpuno detaljno okruženje koje može punopravno konkurirati drugim, plaćenim alatima. Zahvaljujući iznimno aktivnoj zajednici, vrste dodataka koji su dostupni na *Marketplaceu* variraju od onih jednostavnih koji mijenjaju generalni izgled samog alata, do neophodnih dodataka kao što su *ESLint* ili *Vetur*, koji upućuju na sintaksne greške u kôdu te formatiraju i stiliziraju kôd da bi bio čitkiji.

Još jedna moćna i često korištena mogućnost *VSCodea* je zavirivanje u definiciju funkcija s mjesta na kojem se funkcija poziva. Desnim klikom na ime funkcije i odabirom opcije „*Peek definition*" otvara se prozor unutar kojeg se nalazi kôd odabrane funkcije (Slika 2).



Slika 2: Peek definition funkcionalnost

Programer na taj način ne mora izlaziti iz konteksta u kojem se trenutno nalazi samo da bi provjerio funkcionalnost određenih elemenata koji su van tog konteksta.

Vjerojatno najkorisnija mogućnost *VSCode* je ugrađeni alat za pronalaženje grešaka u kôdu. Uz jednostavnu konfiguraciju moguće je vrlo detaljno pratiti tok radnje u kôdu te promjenu stanja varijabli. Ovo svojstvo je osobito došlo do izražaja pri otklanjanju pogrešaka na poslužiteljskoj strani PECS aplikacije.

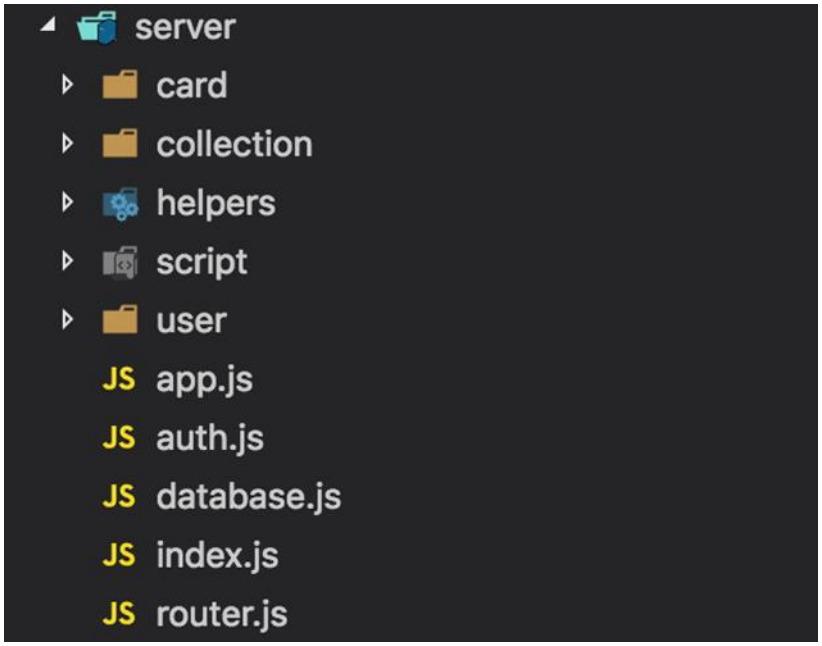
3. OPIS PRAKTIČNOG RADA

3.1. Implementacija poslužiteljske aplikacije u Node.js-u

Kao što je navedeno u uvodu, za izradu poslužiteljske strane *JavaScriptom* korišten je *Node.js*.

3.1.1. Organizacija poslužiteljskog kôda

Za izgradnju poslužiteljske strane kao osnovni okvir korišten je *Express.js*. *Express.js* je minimalni i krajnje fleksibilni programski okvir za izradu aplikacija u *Node.js-u*. Pruža širok spektar mogućnosti za razvoj web i mobilnih aplikacija, te se mnogo drugih programskih okvira temelji na njemu.



Slika 3: Organizacija poslužiteljskog kôda

Glavni direktorij poslužiteljskog kôda je „server" (Slika 3). On je podijeljen u poddirektorije od kojih svaki sadrži skup srodnih funkcionalnosti, te nekoliko *JavaScript* datoteka.

Datoteka *index.js* je polazna točka poslužiteljske aplikacije (Ispis 3). Pozivom *app.js* datoteke vrši se inicijalizacija i konfiguracija, te autentifikacija korisnika. Nakon autentifikacije, glavni usmjerivač se postavlja na definiranu rutu te se aplikacija pokreće na određenom portu.

Ispis 3: Polazna točka poslužiteljske aplikacije

```
const app = require('./app');

app.listen(process.env.PORT, ()=> console.log('App listening'));
```

Kao što je ranije spomenuto, u *app.js* datoteci se vrši konfiguracija aplikacije, autorizacija, namještanje glavne rute i podešava način rukovanja nepostojećim rutama.

Datoteka *auth.js* koja služi za poslužiteljsku autentifikaciju korisnika koristi „*passport*" *npm* paket i JWT (JSON Web Token) strategiju autentifikacije. To je jedna od više od 500 strategija autentifikacije koje „*passport*" paket podržava, a preporučen je za korištenje s *Auth0* platformom.

Datoteka *database.js* služi za inicijalizaciju baze koristeći *Sequelize* ORM, a *router.js* služi za konfiguriranje glavnog usmjerivača. U ovoj datoteci se dodaju usmjerivači svih modela koji se koriste u aplikaciji, *userRoutera, cardRoutera*, i *collectionRoutera*.

Svaki od triju *card, collection* i *user* direktorija sadrže po tri datoteke:

- *index.js*: Usmjerivač za entitet. Sadrži sve rute preko kojih se pozivaju funkcije koje služe za rukovanje entitetom.
- *<entitet>.controller.js*: Sadrži sve funkcije potrebne za rukovanje entitetom. Funkcije se pozivaju iz *index.js* datoteke.
- *<entitet>.model.js*: Služi za oblikovanje entiteta, definiranje atributa i relacija s drugim entitetima.

Direktorij *helpers* služi za spremanje pomoćnih funkcija koje se koriste u više kontrolera. U slučaju PECS aplikacije, ovaj direktorij sadrži *resizeImage.js* datoteku koja služi za kompresiju slika prije spremanja u bazu.

Direktorij *script* sadrži *syncDatabase* skriptu koja služi za kreiranje entiteta u bazi prema trima modelima.

Poslužitelj radi na način da čeka i prima HTTP zahtjeve. Svaki HTTP zahtjev se autentificira, te se u slučaju uspješne autentifikacije prosljeđuje na određenu rutu. Ruta podatke iz zahtjeva prosljeđuje u kontroler koji na temelju tih podataka izvršava određenu radnju nad bazom podataka.

Nakon odrađene radnje, poslužitelj šalje povratnu informaciju pošiljatelju zahtjeva. To mogu biti podaci iz baze, potvrda o uspješno odrađenoj izmjeni podataka u bazi ili informacija o greški koja se dogodila prilikom izvršenja naložene radnje.

3.1.2. Baza podataka

Kao što je spomenuto u uvodnom dijelu, za bazu podataka korišten je *PostgreSQL* u kombinaciji s *Sequelize* ORM-om. Sama struktura baze je relativno jednostavna, te se sastoji od četiri entiteta.

„USER" entitet koristi se za pohranjivanje korisničkih podataka koji preko *Auth0* platforme dolaze sa Google autentifikacijskog servisa. Osim tih podataka, „USER" entitet sadrži i informaciju o razini ovlasti koju korisnik ima na platformi, kao i informaciju o tome je li korisniku uskraćen pristup na platformu.

Atributi „USER" entiteta vidljivi su na Tablici 1.

Tablica 1: Atributi „USER" entiteta

Naziv atributa	Tip atributa	Opis
`id`	CHAR(255)	Primarni ključ, identifikacijski ključ autentifikacijskog servisa
`email`	CHAR(255)	Email adresa
`name`	CHAR(255)	Puno ime korisnika
`role`	ENUM	Vrsta korisnika (SUPERADMIN, ADMIN, USER)
`hasAccess`	BOOLEAN	Označava pravo pristupa platformi
`createdAt`	TIMESTAMP WITH TIMEZONE	Vrijeme dodavanja korisnika
`updatedAt`	TIMESTAMP WITH TIMEZONE	Vrijeme zadnje izmjene korisnika

„COLLECTION" entitet koristi se za pohranjivanje podataka o PECS kolekciji. Svaka PECS kolekcija pripada jednom korisniku te ima više PECS kartica. Atributi „COLLECTION" entiteta vidljivi su u Tablici 2.

Tablica 2: Atributi „COLLECTION“ entiteta

Naziv atributa	Tip atributa	Opis
`id`	INTEGER	Primarni ključ entiteta koji se inkrementalno generira pri kreiranju kolekcije
`title`	CHAR(255)	Naslov kolekcije
`description`	CHAR(255)	Kratak opis kolekcije
`thumbnail`	TEXT	Naslovna slika PECS kolekcije kôdirana u *base64* formatu
`publishRequest`	BOOLEAN	Označava je li vlasnik kolekcije zatražio objavljivanje
`publishRequestDate`	TIMESTAMP WITH TIMEZONE	Datum kreiranja zahtjeva za objavljivanje
`publish`	BOOLEAN	Označava javnu dostupnost kolekcije
`reason`	CHAR(255)	Razlog odbijanja zahtjeva za objavljivanjem
`publicDate`	TIMESTAMP WITH TIMEZONE	Datum objavljivanja kolekcije
`createdAt`	TIMESTAMP WITH TIMEZONE	Datum kreiranja kolekcije
`updatedAt`	TIMESTAMP WITH TIMEZONE	Datum zadnje izmjene na kolekciji
`deletedAt`	TIMESTAMP WITH TIMEZONE	Datum brisanja kolekcije
`userId`	INTEGER	Strani ključ „USER“ entiteta

„COLLECTION“ entitet osim osnovnih podataka o kolekciji sadrži i informacije o njenom statusu. Preko atributa „publishRequest“, „publishRequestDate“, „publish“, „reason“ i „publicDate“ može se vidjeti kada je dan zahtjev za objavljivanjem, je li odbijen i s kojim razlogom, te ukoliko je zahtjev prihvaćen, kada je kolekcija objavljena.

„CARD” entitet koristi se za pohranjivanje podataka o pojedinačnim PECS karticama. Svaka PECS kartica pripada kolekciji koja može imati više kartica. Atributi „CARD” entiteta vidljivi su u Tablici 3.

Tablica 3: Atributi „CARD“ entiteta

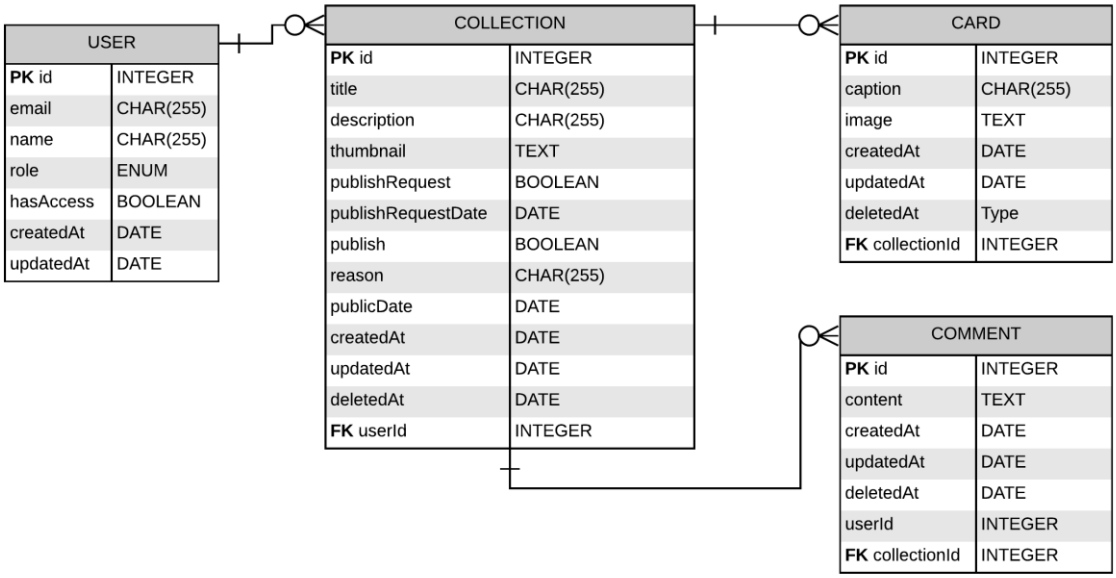
Naziv atributa	Tip atributa	Opis
id	INTEGER	Primarni ključ entiteta koji se inkrementalno generira pri kreiranju kolekcije
caption	CHAR(255)	Natpis na dnu PECS kartice
image	TEXT	Slika s PECS kartice kôdirana u bas64 formatu
createdAt	TIMESTAMP WITH TIMEZONE	Datum kreiranja kartice
updatedAt	TIMESTAMP WITH TIMEZONE	Datum zadnje izmjene na kartici
deletedAt	TIMESTAMP WITH TIMEZONE	Datum brisanja kartice
collectionId	INTEGER	Strani ključ „COLLECTION“ entiteta

„COMMENT” entitet koristi se za pohranjivanje komentara o javno dostupnim PECS kolekcijama. Atributi „COMMENT“ entiteta vidljivi su u Tablici 4.

Tablica 4: Atributi „COMMENT“ entiteta

Naziv atributa	Tip atributa	Opis
id	INTEGER	Primarni ključ entiteta koji se inkrementalno generira pri kreiranju kolekcije
content	TEXT	Tekst komentara
createdAt	TIMESTAMP WITH TIMEZONE	Datum kreiranja kartice
updatedAt	TIMESTAMP WITH TIMEZONE	Datum zadnje izmjene na kartici
deletedAt	TIMESTAMP WITH TIMEZONE	Datum brisanja kartice
userId	INTEGER	ID autora komentara
collectionId	INTEGER	Strani ključ „COLLECTION“ entiteta

Navedeni entiteti i njihovi odnosi također su prikazani i uz pomoć ER dijagrama na Slici 4.



Slika 4: ER dijagram baze podataka

Entitet „COLLECTION“ pripada entitetu „USER“ preko *„userId“* stranog ključa. Svaki „COLLECTION“ može pripadati samo jednom „USERU“, dok „USER“ može imati nijednu ili više „COLLECTION“-a (1-n relacija).

Entitet „CARD“ pripada entitetu „COLLECTION“ preko *„collectionId“* stranog ključa. Svaki „CARD“ može pripadati samo jednom „COLLECTIONU“, dok „COLLECTION“ može imati nijedan ili više „CARDOVA“ (1-n relacija).

Entitet „COMMENT“ pripada „COLLECTION” entitetu preko *„collectionId“* stranog ključa. Svaki „COMMENT“ može pripadati samo jednom “COLLECTION“-u, dok „COLLECTION“ može imati nula ili više „COMMENTA“ (1-n relacija).

3.1.3. Tijek rada poslužiteljske aplikacije

Tijek rada aplikacije najbolje je objasniti na primjeru dodavanja nove PECS kartice u kolekciju.

Korisnik odabirom tipke za kreiranje nove kartice šalje HTTP zahtjev POST metodom na rutu *„/card“*. Uz sam zahtjev, na rutu se kao parametar šalje i *id* kolekcije u koju se želi dodati kartica.

Usmjerivač na poslužitelju po ruti prepoznaje zahtjev i njegove parametre, te ih zajedno s dodanim podacima korisnika koji je poslao zahtjev prosljeđuje na *create()* funkciju koja se nalazi unutar *„card.controler.js“* datoteke.

Funkcija unutar parametara traži *collectionId* te koristeći *Sequelize* metodu *findByPk()* dohvaća podatke kolekcije iz baze podataka. Ako kolekcija nije pronađena u bazi, poslužitelj na klijentsku stranu šalje HTTP status 404 (*not found*). Ako je kolekcija pronađena, poslužitelj dohvaća njene podatke. Među podacima kolekcije nalazi se *i userId*.

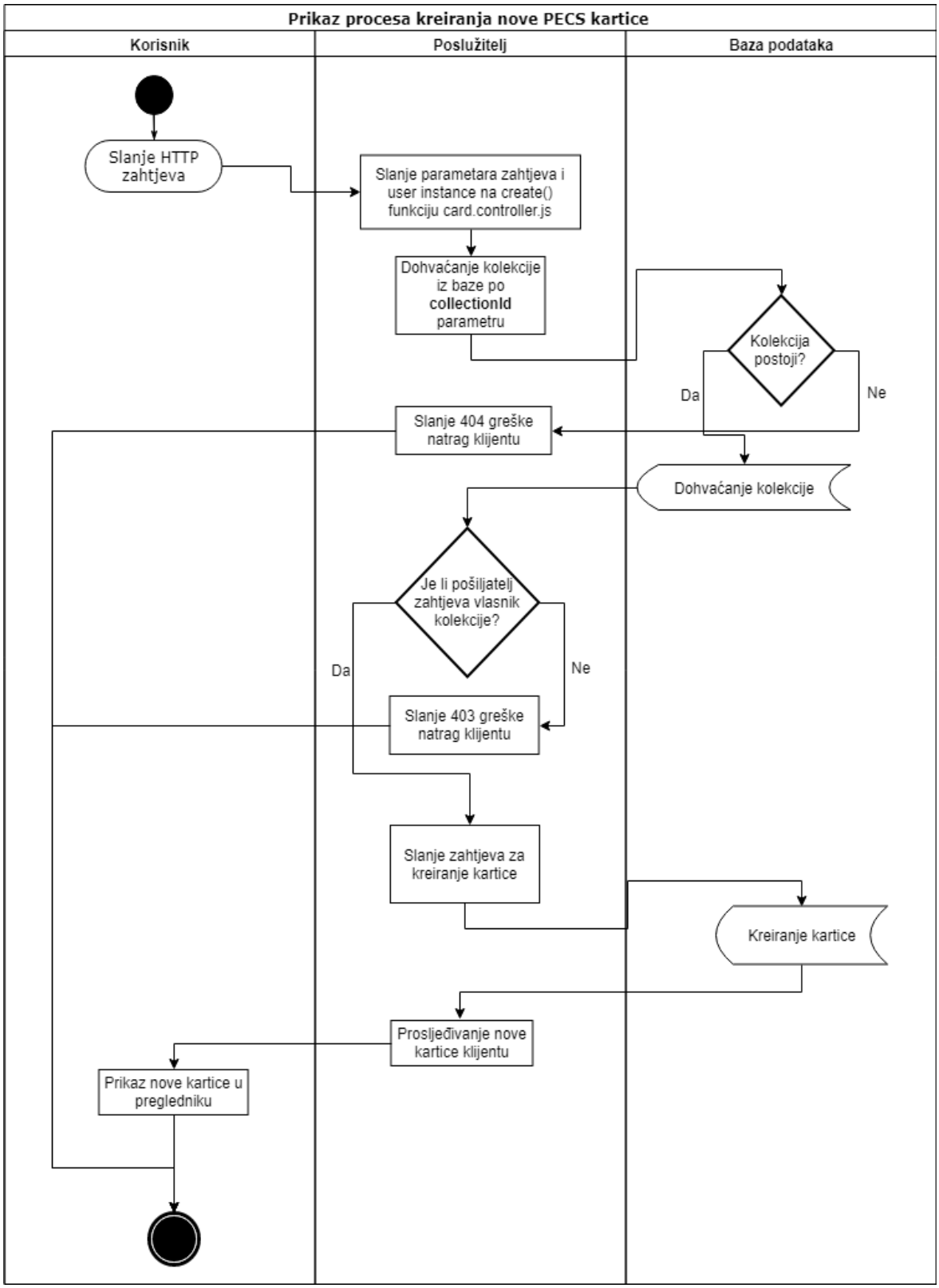
Ukoliko *userId* iz kolekcije i *userId* korisnika koji je poslao zahtjev nisu isti, to znači da je korisnik koji nije vlasnik kolekcije poslao zahtjev za dodavanjem kartice. S obzirom da u aplikaciji trenutno nije dopušteno mijenjanje kolekcija koje nisu u vlasništvu korisnika (čak i ako korisnik ima administratorska prava), funkcija se tog trena prekida te se na klijentsku šalje HTTP status 403 (*forbidden*) koji daje do znanja da je takva radnja zabranjena.

U slučaju da *userId* kolekcije odgovara *userId*-u korisnika koji je poslao zahtjev, provodi se kreiranje prazne kartice čiji se *collectionId* postavlja na *collectionId* iz parametara HTTP zahtjeva. Nakon toga se na klijentsku stranu šalju podaci nove kartice zajedno s HTTP statusom 200, što znači da je zahtjev uspješno obrađen.

Ako tijekom obrade zahtjeva dođe do neke neočekivane pogreške na poslužitelju, klijentska strana će zaprimiti HTTP status 500 (*internal server error*). Taj status korisniku daje do znanja da uzročnik pogreške nije ispravnost njegovog zahtjeva, već je problem u poslužitelju.

Nova kartica se na klijentskoj strani uz pomoć *Vue.js-a* dodaje na kraj liste kartica trenutno odabrane kolekcije. Ta promjena stanja liste kartica uzrokuje osvježavanje prikaza liste u pregledniku, što naposlijetku rezultira prikazom nove, prazne PECS kartice.

Slika 5 prikazuje dijagram toka dodavanja nove PECS kartice u kolekciju.



Slika 5: Dijagram toka dodavanja nove PECS kartice u kolekciju

3.1.4. Kompresija slika prije spremanja u bazu

S obzirom da su dimenzije PECS kartice relativno male, slike koje se koriste na njima ne moraju imati veliku rezoluciju. Štoviše, korištenje velikih slika u PECS kolekcijama s velikim brojem kartica moglo bi usporiti brzinu rada, kako poslužitelja, tako i klijentske strane. Najjednostavniji način za izbjegavanje tog problema je postavljanje ograničenja na veličinu slike koju korisnik može koristiti za PECS kartice. Međutim, to je daleko od idealnog, jer pogoršava iskustvo korištenja same aplikacije, gdje korisnik mora sam raditi kompresiju slike prije korištenja.

Alternativno rješenje je implementacija funkcije (Ispis 4) koje će prije pohrane svake slike u bazu, odraditi kompresiju. „*Sharp*" je alat dostupan u obliku *npm* paketa koji na strani poslužitelja obavlja taj posao. Podržava širok spektar formata slika brojne mogućnosti podešavanja razine i vrste kompresije uz zavidnu brzinu i nisku hardversku zahtjevnost.

Ispis 4: Kôd za kompresiju slike

```
const sharp = require('sharp');

function compressImage(image) {
  const parts = image.split(';');
  const mimType = parts[0].split(':')[1];
  const imageData = parts[1].split(',')[1];
  const img = Buffer.from(imageData, 'base64');
  return sharp(img).resize(350, 350, { withoutEnlargement: true })
    .jpeg({ quality: 50, chromaSubsampling: '4:4:4', force: false })
    .png({ quality: 50, force: false })
    .toBuffer()
    .then(resizedImageBuffer => {
      const resizedImageData = resizedImageBuffer.toString('base64');
      const resizedB64 = `data:${mimType};base64,${resizedImageData}`;
      return resizedB64;
    });
}

module.exports = compressImage;
```

U slučaju PECS kartica, *„sharp"* je postavljen na način da sve slike rezolucije veće od 350x350 piksela smanjuje na tu rezoluciju, te potom nad slikama JPEG i PNG formata postavlja razinu kvalitete na 50%. Ta razina se empirijski pokazala kao idealan omjer kvalitete i veličine u bazi. S obzirom da je *„sharp"* sposoban detektirati format slike, JPEG i PNG kompresiju je moguće postaviti jednu iza druge. Kompresija će se izvršiti samo nad

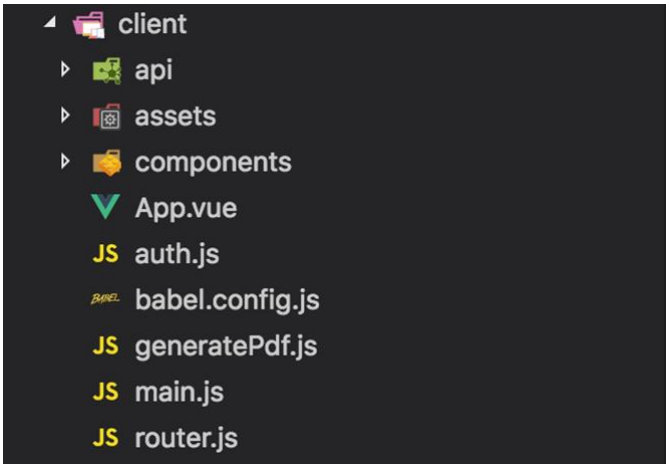
slikom odgovarajućeg formata, što znači da se nad slikom JPEG formata neće odraditi PNG kompresija, i obratno. Nakon što se kompresija slike odradi, slika se prebacuje natrag u *base64* format te se kao takva sprema u bazu.

3.2. Implementacija klijentske aplikacije u Vue.js-u

Kao što je navedeno u uvodu, za izradu klijentskog dijela aplikacije korišten je *Vue.js* programski okvir.

3.2.1. Organizacija klijentskog kôda

Glavni direktorij poslužiteljskog kôda je „server" (Slika 6) koji je podijeljen u poddirektorije od kojih svaki sadrži skup srodnih funkcionalnosti, te nekoliko JavaScript datoteka.



Slika 6: Organizacija klijentskog kôda

Datoteka *main.js* je polazna točka klijentske aplikacije. Pozivom *main.js* datoteke izvodi se autentifikacija, te inicijalizacija i konfiguracija Vue instance, te se preko *App.vue* datoteke pokreće aplikacija.

App.vue je glavna komponenta aplikacije. Sadrži uzglavlje i podnožje aplikacije, te središnji dio koji se dinamički mijenja ovisno o ruti.

Datoteka *auth.js* se poziva u *main.js* datoteci, te služi za provjeru ispravnosti pristupnog tokena i korisničkih podataka u *local storageu* kao i za postavljanje te uklanjanje istih (u slučaju odjavljivanja s aplikacije). Koristi ranije spomenuti *„auth0-js" npm* paket koji pojednostavljuje komunikaciju s *Auth0* platformom.

Datoteka *generatePdf.js* sadrži funkciju za generiranje PDF verzije kolekcije.

Datoteka *router.js* sadrži sve rute klijentske aplikacije (primjer dijela ruta se može vidjeti na Ispisu 5). Inicijalni problem jednostraničnih aplikacija je nemogućnost dijeljenja poveznica na točno određeni dio unutar aplikacije. Vue.js taj problem rješava uvođenjem Vue usmjerivača.

Ispis 5: Primjer dijela ruta klijentske aplikacije

```
const router = new Router({
  mode: 'history',
  routes: [
    {
      path: '/',
      name: 'dashboard',
      component: Dashboard
    }, {
      path: '/private-collections',
      name: 'private',
      component: PrivateCollections
    }, {
      path: '/about',
      name: 'about',
      component: About
    }
  ]
});
```

Datoteka *router.js* sadrži listu svih definiranih ruta i pod ruta. Međutim, umjesto da pozivaju potpuno novu stranicu unutar aplikacije, te rute samo zamjenjuju jednu komponentu unutar stranice drugom komponentom. Ostatak stranice ostaje nepromijenjen te ga nije potrebno ponovno učitavati. U slučaju *PECS card maker* aplikacije, rute mijenjaju samo središnji dio aplikacije, dok je uzglavlje i podnožje aplikacije uvijek isto.

Api direktorij sadrži sve funkcije koje klijentska aplikacija koristi da bi komunicirala direktno s poslužiteljskom stranom aplikacije. Kad god korisnik klikne na PECS kolekciju ili karticu ili kad pokuša izbrisati ili objaviti neku kolekciju, to rezultira API pozivom na poslužiteljsku aplikaciju.

Da bi se taj API poziv ispravno oblikovao i poslao, koristi se *„axios" npm* paket. *„axios"* je HTTP klijent za internetski preglednik i Node.js. Uz pomoć njega je moguće iz preglednika slati HTTP zahtjeve na bilo koji poslužitelj te primati i odgovore i automatski ih prebacivati u JSON format koji preglednik razumije.

Api direktorij sadrži više kolekcija API poziva na poslužitelj, grupiranih po kategorijama (*user*, *card*, *collection*).

Direktorij *assets* sadrži sve statičke dokumente kao što su slike koje se koriste u aplikaciji, dok *components* direktorij sadrži sve Vue.js datoteke, tj. okvire koji se koriste za oblikovanje i prikazivanje podataka koji dolaze se poslužiteljske strane.

3.2.2. Funkcija za generiranje PDF datoteke na klijentskoj strani

Datoteka *generatePdf.js* sadrži funkciju za generiranje PDF verzije kolekcije. Zadnji korak svakog alata za izradu PECS kartica je ispis istih uz pomoć pisača u formatu u kojem će kartice biti jednostavno izrezati, plastificirati i koristiti pri komunikaciji. S obzirom na to, iznimno je bitno omogućiti korisnicima da generiraju PDF dokument sa svim PECS karticama unutar kolekcije.

Iako generiranje PDF-a samo po sebi nije pretjerano zahtjevan proces po pitanju potrebne računalne snage, korištenje poslužitelja u tu svrhu moglo bi predstavljati problem u slučaju da velik broj korisnika istovremeno zatraži generiranje.

Idealno rješenje je stoga koristiti snagu klijentskog računala za generiranje PDF-a. To će rezultirati većom brzinom generiranja za korisnike, kao i nižim opterećenjem poslužitelja. *Npm* paket *„pdfmake"* pokazao se kao idealno rješenje za generiranje PDF-a na strani klijenta, zahvaljujući svojim širokim rasponom mogućnosti oblikovanja teksta, slika, okvira i sl.

Kada korisnik na alatnoj traci kolekcije pritisne tipku „Download PDF", na klijentskom računalu se pokreće *generatePdf()* funkcija koja prima podatke kolekcije sa svim pripadajućim karticama. Taj sadržaj se oblikuje i sprema u objekt. Koristeći razne mogućnosti „pdfmake" paketa, PDF se oblikuje na slijedeći način.

Prva stranica u zaglavlju sadrži ime aplikacije i njen URL, ispod čega je naveden naslov kolekcije i opis. Naredne stranice sadrže PECS kartice oblikovane tako da svaka stranica sadrži 3 reda po 2 kartice (Ispis 6). Nakon 3 reda je potrebno eksplicitno kreirati novi list, jer će u protivnom idući red PECS kartica biti djelomično na jednoj a djelomično na drugoj stranici, što znači da se neće moći izrezati i koristiti.

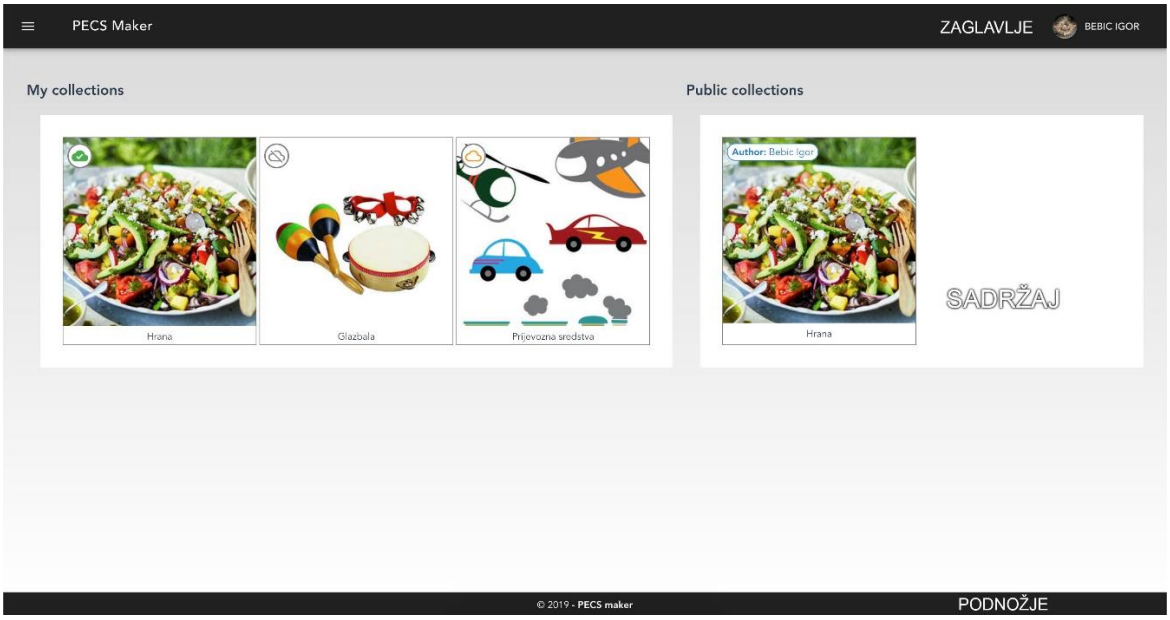
Ispravno napravljen objekt se zatim prosljeđuje u *createPdf()* funkciju, a rezultat te funkcije se prosljeđuje u *download()* funkciju. Funkcija *download()* funkcija pokreće preuzimanje generiranog PDF-a na korisničkom web pregledniku.

Ispis 6: Dio createPdf() kôda za prikaz PECS kartica

```
cards.forEach((card, index) => {
  if (index % 2 !== 0) return;
  const element = {
    alignment: 'justify',
    columns: []
  };
  element.columns[0] = {
    table: {
      widths: [232],
      heights: [200, 30],
      body: [
        [{ image: card.image || placeholder,
           alignment: 'center', width: 230, height: 198 }],
        [{ text: card.caption || '',
           alignment: 'center', margin: 7 }]
      ]
    }
  };
  const second = cards[index + 1];
  if (second) {
    element.columns[1] = {
      table: {
        widths: [232],
        heights: [200, 30],
        body: [
          [{ image: second.image || placeholder,
             alignment: 'center', width: 230, height: 198 }],
          [{ text: second.caption || '',
             alignment: 'center', margin: 7 }]
        ]
      }
    };
  }
  dd.content.push(element);
  let textPart = index % 2 !== 0 ? '' : `\n`;
  const pageBr = index === 4 || ((index - 4) % 6 === 0)
      ? 'after'
      : '';
  if (pageBr) textPart = '';
  if (cards[ index + 2 ]) dd.content.push (
      { text: textPart, pageBreak: pageBr }
  );
});
```

3.2.3. Proces navigacije klijentskom aplikacijom

Ranije je već spomenuto da Vue.js omogućava korištenje ruta na jednostraničnim aplikacijama, te da se navigacijom kroz aplikaciju ne mijenja cijeli prikaz aplikacije već samo njen dio. Idealan primjer toga je glavna stranica PECS *card maker* aplikacije (Slika 7).



Slika 7: Glavna stranica PECS card maker aplikacije

Na slici je stranica bojama podijeljena na tri dijela: zaglavlje, podnožje i sadržaj. (Ispis 7)

Ispis 7: Prikaz kôda glavne stranice

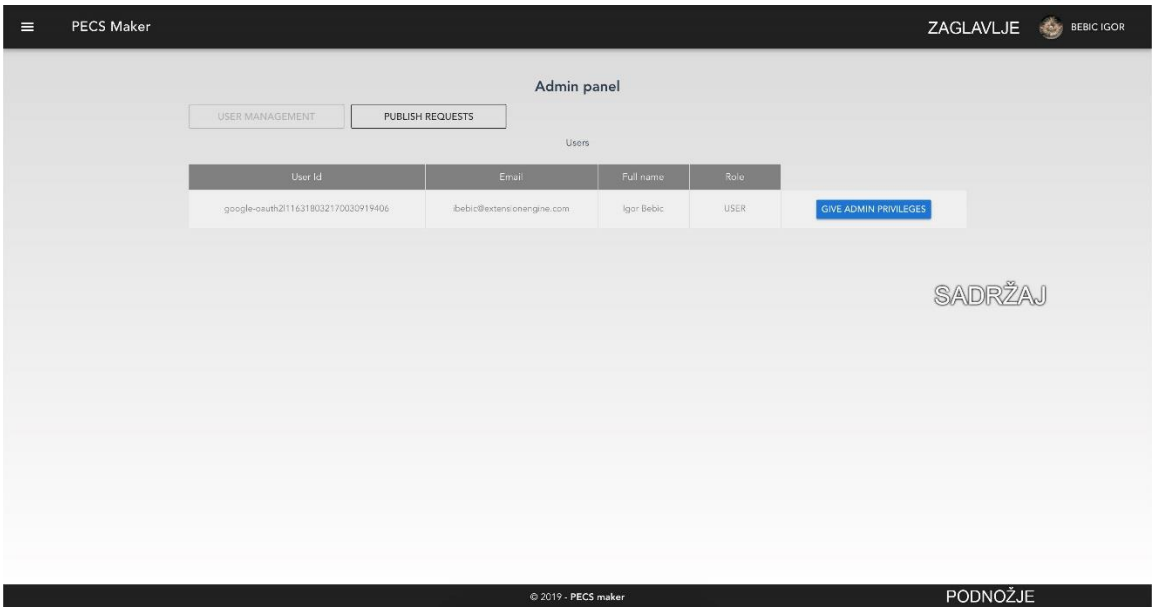
```
<template>
  <v-app id="app">
    <navbar
      v-if="$route.name !== 'callback'"
      @addNew="showModal = true" />
    <new-collection-dialog
      v-if="showModal"
      @closeModal="showModal = false"
      @addNew="createNewCollection" />
    <router-view />
    <footer-comp />
  </v-app>
</template>
```

U kôdu su vidljive četiri komponente. Prva komponenta je *„navbar“* i sadrži kôd za prikaz zaglavlja stranice. Iduća komponenta je *„new-collection-dialog“* koja sadrži kôd za prikaz prozora za kreiranje nove kolekcije, te nije bitna za trenutnu temu.

Komponenta ispod je *„router-view“*. To je takozvana dinamička komponenta jer u sebi može sadržavati bilo koju drugu komponentu. Na kraju, tu je *„footer-comp“* koji sadrži komponentu za prikaz podnožja aplikacije.

Komponente zaglavlja i podnožja su statične i ne mijenjaju svoj sadržaj s obzirom na to gdje se korisnik nalazi u aplikaciji. *„router-view“* komponenta se međutim mijenja s obzirom na rutu na kojoj se korisnik nalazi. Na glavnoj stranici aplikacije, *„router-view“* komponenta zapravo prikazuje *„dashboard“* komponentu.

Međutim, navigacijom na *„/admin“* rutu kroz preglednik, *„router-view“* mijenja svoj sadržaj, te umjesto *„dashboard“* prikazuje *„admin-panel“* komponentu (Slika 8).



Slika 8: Admin stranica PECS card maker aplikacije

Da bi se to postiglo, potrebno je koristiti *„vue-router“ npm* paket. *„vue-router“* na klijentskoj strani generira instancu preusmjerivača koji se konfigurira tako da mu se proslijede važeće rute uparene s komponentama koje one se na tim rutama namještaju (Ispis 5).

Na ovaj način korisnik ima osjećaj da se kreće po klasičnoj web stranici, te može bez problema koristiti poveznice koje će ga odvesti na konkretni dio u aplikaciji, što prije nije bilo moguće kod jednostraničnih aplikacija.

4. ZAKLJUČAK

Cilj ovog rada bio je napraviti aplikaciju za izradu PECS kartica i kolekcija, koja je besplatna, jednostavna i lako dostupna za korištenje. Komercijalna rješenja su zbog relativnog male skupine ciljanih korisnika iznimno skupa s obzirom na to što nude. Također, rađena su koristeći zastarjele tehnologije, zahtijevaju instalaciju na računalo i poprilično su neintuitivna za korištenje. Radi toga nije rijetkost da ustanove koje imaju potrebu za ovakvom vrstom aplikacija organiziraju obuke, koje su, kao i same aplikacije, preskupe.

Dijeljenje PECS kolekcije je još jedan veliki problem. Dostupne aplikacije međusobno nisu kompatibilne te spremaju svoje kolekcije u formate koji nisu čitljivi uz pomoć drugih alata.

Jednostavnost i minimalizam bili su glavna vodilja pri dizajniranju PECS *card maker* aplikacije. Također, naglasak je stavljen i na mogućnost dijeljenja kolekcija. Korisnik mora biti u mogućnosti jednostavno preuzeti javno dostupnu kolekciju, kopirati njen sadržaj i prilagoditi svojim potrebama. Također, komentiranje javno dostupnih kolekcije omogućuje korisnicima da obavijeste druge korisnike o prednostima i nedostacima pojedinih kolekcija.

Jedno od budućih mogućnosti aplikacije biti će ocjenjivanje kolekcija od strane korisnika. To će omogućiti stvaranje liste najbolje rangiranih kolekcija. Broj ukupnih preuzimanja također je jedno od ideja za buduće nadogradnje, kao i pretraživanje po *meta* podacima.

Nadalje, zanimljiva mogućnost bila bi pretraživanje i korištenje individualnih kartica svih objavljenih kolekcija. Na taj način bi se znatno ubrzala izrada novih kolekcija.

Online pretraživanje slika putem alatne trake jedno je od ambicioznijih ideja za nadogradnju. Međutim, kvalitetni servisi za pretragu slika, koji imaju potrebne filtere, plaćena su usluga. Korištenje tih usluga zahtijeva implementaciju nekog oblika monetizacije putem reklama, a u svrhu pokrivanja troškova pretrage, kao i rada poslužitelja.

U međuvremenu, aplikacija će se koristiti u malom krugu ljudi koji će njene nedostatke moći zanemariti zahvaljujući prednostima koje ima nad drugim rješenjima.

LITERATURA

[1] Vue.js dokumentacija, https://vuejs.org/v2/guide/

Datum zadnjeg pristupa: 20. svibnja, 2019

[2] *Node.js* dokumentacija, https://nodejs.org/en/docs/

Datum zadnjeg pristupa: 15. svibnja, 2019

[3] *Express.js* dokumentacija, https://expressjs.com/en/guide/routing.html

Datum zadnjeg pristupa: 15. svibnja, 2019

[4] *Sequelize* dokumentacija, http://docs.sequelizejs.com/

Datum zadnjeg pristupa: 20. svibnja, 2019

[5] *Auth0* dokumentacija, https://auth0.com/learn/

Datum zadnjeg pristupa 10. svibnja, 2019