Web aplikacija za prodaju personaliziranih proizvoda

Huten, Perica

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
STRUČNI STUDIJ RAČUNARSTVO

PERICA HUTEN

IZRADA WEB APLIKACIJE ZA PRODAJU PERSONALIZIRANIH PROIZVODA

ZAVRŠNI RAD

Čakovec, rujan 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI STUDIJ RAČUNARSTVO

PERICA HUTEN

IZRADA WEB APLIKACIJE ZA PRODAJU PERSONALIZIRANIH PROIZVODA

DEVELOPMENT OF A WEB APPLICATION FOR SALE OF PERSONALIZED PRODUCTS

ZAVRŠNI RAD

Mentorica:

Dr. sc. Sanja Brekalo, prof. v. š.

Čakovec, rujan 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
ODBOR ZA ZAVRŠNI RAD

Čakovec, 1. veljače 2022.

država: **Republika Hrvatska**
Predmet: **Izrada Web sadržaja-izborni**
Grana: **2.09.06 programsko inženjerstvo**

ZAVRŠNI ZADATAK br. 2021-RAC-R-70

Pristupnik: **Perica Huten (0313024399)**
Studij: redovni preddiplomski stručni studij Računarstvo
Smjer: Programsko inženjerstvo

Zadatak: **Web aplikacija za prodaju personaliziranih proizvoda**

Opis zadatka:

Tema rada je izraditi u JavaScriptu (korištenjem JavaScript biblioteka i programskih okvira kao što su React library, Node.js, Express framework, no SQL baza, MERN stack i dr.) web aplikaciju za prodaju koja omogućuje personalizaciju proizvoda koji se prodaju. Izrađena aplikacija osim standardnih dijelova web trgovine posjeduje dio za upravljanje personalizacijom artikala koji se prodaju. Registrirani korisnici mogu postavljati narudžbe personaliziranih proizvoda te pratiti njihovu isporuku i realizaciju. Administratori prate narudžbe i upisuju zaprimanje i realizaciju naručenih proizvoda.

Zadatak uručen pristupniku: 1. veljače 2022.
Rok za predaju rada: 20. rujna 2022.

Mentor:

dr. sc. Sanja Brekalo, prof. v. š.

Predsjednik povjerenstva za završni ispit:

ZAHVALA

Ovim putem se posebno zahvaljujem svojoj mentorici prof.dr.sc Sanji Brekalo na stručnim savjetima, strpljenju i pomoći kod izabira teme i alata potrebnih za izradu ovog rada.

Sažetak

Tema ovog završnog rada je izrada web aplikacije za prodaju personaliziranih proizvoda izrađena u MERN stogu. Prednost rada u MERN stogu jest činjenica da se svaki dio aplikacije može pisati JavaScript programskim jezikom. Poslužiteljski dio aplikacije (engl. *backend*) koristi Express.js i Node.js radno okruženje te MongoDB bazu podataka. Firebase, Cloudinary i Stripe su aplikacije treće strane integrirane u web aplikaciju za autentifikaciju korisnika, upravljanje slikama i plaćanje. Instalacija potrebnih modula radi se preko Node Package Managera, a aplikacija se može isporučiti u prilično maloj datoteci jer se isporučuje bez instaliranih modula.

Za primaran rad klijentskog dijela aplikacije koristi se React.js biblioteka. Iskoristive komponente React.js biblioteke omogućuju njihovo korištenje kroz cijelu aplikaciju. Redux reduktori pomažu u postavljanju globalnih stanja. Aplikacija je podijeljena na 3 korisničke uloge. Gost ima mogućnost pregleda proizvoda, dodavanja proizvoda u košaricu, personalizaciju i registraciju ili prijavu. Korisnik može pregledavati vlastite narudžbe, ispisivati PDF račune, ocjenjivati proizvode i, primarno, kupovati proizvode. Administrator stranice ima najveće ovlasti. To podrazumijeva pristup posebnom sučelju preko kojeg može dodavati, uređivati i brisati trenutne proizvode, kategorije, potkategorije i kupone. Također ima pristup i ispisu svih narudžbi na stranici i može mijenjati njihov status koji korisnici mogu vidjeti na svojem sučelju.

Korisnik kod registracije dobiva automatizirani e-mail s poveznicom za upis lozinke u sustav. U slučaju zaboravljanja lozinke, korisnik ima mogućnost njenog ponovnog postavljanja.

Personalizacija proizvoda izvršava se putem Cloudinary upravitelja medija u kojemu korisnik može odabrati logo ili sliku koju stavlja na proizvod na prethodno označeno mjesto, a ima i mogućnost upisa teksta. Personalizacija proizvoda je opcionalna. Administrator može pristupiti popisu korisnika aplikacije putem Firebase servisa te im onemogućiti račun ili ga u potpunosti izbrisati. Također ima pristup svim slikama korištenim u aplikaciji putem Cloudinary konzole te može manipulirati slikama za bolji prikaz na web aplikaciji.

Stripe sustav čuva ispis svih transakcija provedenih na stranici. Bilježi uspješne i neuspješne transakcije.

***Ključne riječi**: MERN, JavaScript, Express, Node, MongoDB, React, Personalizacija*

SADRŽAJ

1. UVOD

Personalizacija proizvoda omgućuje dodanu vrijednost nekom proizvodu. Personalizacijom proizvoda omogućen je izbor dizajna proizvoda, bilo da je riječ o nekoj majici s ispisom ili o računalnoj komponenti s personaliziranom bojom ili nekim uzorkom. Personalizacija korisniku omogućuje kontrolu izgleda proizvoda. Trend personalizacije tijekom posljednjih godina raste zahvaljujući napretku tehnologije.[1] Prema studiji Bain&Company, povećanje stope zadržavanja kupaca za samo 5% može povećati profit za 25% do 95%. [2]

Cilj završnog rada je izraditi *full-stack* web aplikaciju za prodaju koja omogućuje personalizaciju proizvoda korištenjem *JavaScript* biblioteka i programskih okvira kao što su *React library*, Node.js, Express *framework*, NoSQL baza podataka i MERN stog. Korisnicima aplikacija omogućuje praćenje statusa svojih narudžbi, a administratorima omogućuje praćenje svih narudžbi te izmjenjivanje podatka o njihovom zaprimanju, realizaciji i isporuci te upravljanje korisničkim računima preko Firebase platforme.

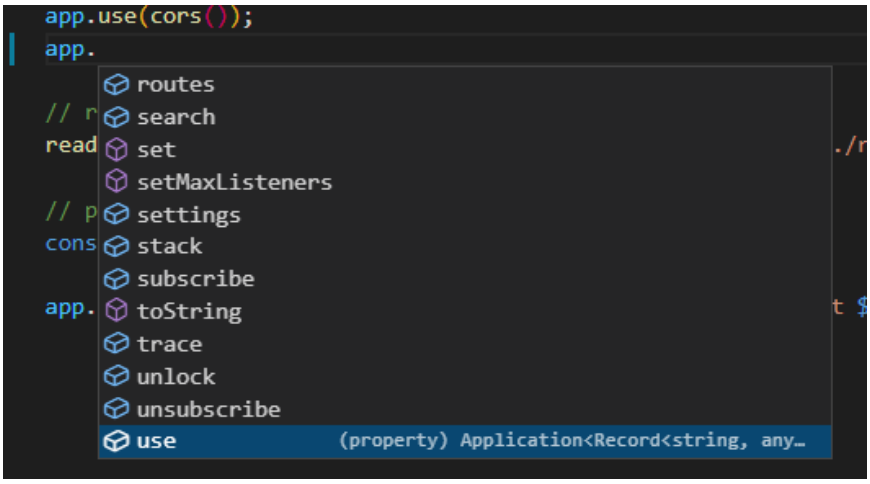
2. RAZVOJNI ALATI I TEHNOLOGIJE

Kod izrade zadane web aplikacije koristi se više programa i alata, kao primjerice Visual Studio Code, HTML, JavaScript, React.js, Node.js, Express.js, Firebase, Cloudinary i MongoDb.

2.1. Visual Studio Code

Visual Studio Code je uređivač kôda stvoren u tvrtki Microsoft koji podržava sve operacije razvoja aplikacija kao što su ispravljanje grešaka, pokretanje zadataka i kontrolu inačica. Cilj alata je pružiti mogućnosti potrebne za brzo pisanje kôda i otklanjanje grešaka. Složeniji zadaci uobičajeno se obavljaju u kompleksnijim IDE-ovima (engl. *Integrated Development Environment*) kao što je Visual Studio IDE. Visual Studio Code podržava sve operacijske sustave (Windows, Linux, macOS) i u potpunosti je besplatan za privatno ili komercijalno korištenje te kao i mnogi drugi razvojni alati podržava *IntelliSense*. [3]

IntelliSense je termin koji se koristi za razne mogućnosti editiranja kôda kao što su završavanje kôda, informacije o parametrima kod pisanja kôda, prikaz brzih informacija o raznim metodama i varijablama. Visual Studio Code omogućava IntelliSense za JavaScript, TypeScript, JSON, HTML, CSS, SCSS, a moguće ga je dodati i za druge programske jezike, biblioteke i programske okvire pomoću proširenja unutar Visual Studio Code-a. Slika 1 prikazuje IntelliSense pomoć pri pisanju JavaScript kôda. Pri izradi aplikacije korišten je Visual Studio Code sa svim potrebnim dodacima za MERN stack izradu web aplikacije, kao što su Mongo snippets, ExpressJS snippets, ReactJS Code Snippets, NodeJS snippets i drugi.[4]



Slika 1. IntelliSense primjer

Izvor: autor

2.2. HTML

HTML (engl. *HyperText Markup Language*) je skriptni jezik koji definira značenje i strukturu web sadržaja. *HyperText* označava međusobno povezivanje web stranica pomoću linkova. To je označni jezik, što znači da koristi oznake za definiranje sadržaja unutar nekog dokumenta. Lako je čitljiv, što znači da datoteke za označavanje koriste standardne riječi, a ne standardnu programsku sintaksu. Označnih jezika ima više, no najpopularniji su HTML i XML. Sadržaj svake stranice definiran je HTML oznakama. Glavna oznaka svakog HTML dokumenta je <html> te njome započinje i završava svaka html stranica. Postoje i ostale oznake kao što su <head>, <body>, koje služe za sekcije stranice, dok elementi poput <a> <p> <table> definiraju same elemente unutar stranice. Kod definiranja elementa preporuča se početna i završna oznaka kao npr. . [5]

2.3. JavaScript

JavaScript ili JS je programski jezik koji se interpretira, tj. kompilira u trenutku izvođenja (engl. *just-in-time*). Iako je najpoznatiji kao skriptni web jezik, može se koristiti u okruženjima koja nisu preglednici, kao što su npr. Node.js, Apache CouchDB, Adobe Creative Cloud i drugi. JavaScript je prototipski baziran, više paradigmatski, jednonitni dinamički jezik koji podržava objektno orijentirane i deklarativne stilove programiranja. JavaScript se izvodi na klijentskoj strani weba (na korisničkom računalu), što omogućava

programiranje načina ponašanja web stranica u raznim uvjetima. To je programski jezik koji se široko koristi za kontrolu ponašanja web stranica.

JavaScript je dinamičan skriptni jezik koji podržava konstrukciju objekta na bazi prototipa. Osnovna sintaksa je slična Java i C++ programskim jezicima kako bi se smanjio broj novih koncepata potrebnih za učenje jezika. *If* funkcije, *for* i *while* petlje, *switch*, *try* i *catch* blokovi funkcioniraju prilično slično kao i u drugim jezicima.

JavaScript može funkcionirati kao proceduralan ili objektno orijentiran jezik. Objekti se u JavaScript-u stvaraju programski, dodavanjem metoda i svojstava na prazne objekte u vremenu izvođenja, za razliku od definiranja sintaktičkih klasa u prevedenim jezicima (C++ i Java). Nakon stvaranja objekta može se koristiti kao prototip ili nacrt za kreiranje sličnih objekata. [6]

2.4. MERN stack

MERN je jedna od nekoliko varijacija MEAN stoga (MonogoDB, Express, Angular, Node) u kojemu je tradicionalan Angular.js zamijenjen s React.js-om. MERN je skraćenica za MongoDB, Express.js, React.js i Node.js četiri ključne tehnologije koje čine stog (*engl. stack*). MongoDB je baza podataka, Express je programski okvir za Node.js server, React je JavaScript programski okvir klijentske strane i Node.js je JavaScript web server tj. izvršno okruženje za JavaScript. MERN arhitektura omogućava izradu troslojne arhitekture (*front-end*, *back-end*, baza podataka) korištenjem JavaScript i JSON-a. [7]

2.5.React

React je JavaScript biblioteka koju je 2013. godine objavio Facebook za izradu modularnog korisničkog sučelja (*front-end* dio aplikacije). React se izvršava na strani klijenta. Popularnost Reacta zasjenila je popularnost svih drugih *front-end* razvojnih okvira baziranih na JavaScriptu. React omogućava olakšanu izradu dinamičkih aplikacija jer zahtijeva manje kodiranja i pruža više funkcionalnosti za razliku od samog JavaScript-a gdje kodiranje može postati kompleksno.

React je poznat po tome što je vrlo brzo izvršava u usporedbi s ostalim *front-end* razvojnim okvirima, zbog čega je najkorišteniji *front-end* razvojni okvir baziran na

JavaScriptu. Razlog za visoku učinkovitost je u suštini značajka virtualnog DOM-a (engl. *Document Object Model*) koji uspoređuje prijašnja stanja komponenti pomoću algoritma. Uspoređuje se virtualni DOM i DOM preglednika te se zatim ažuriraju samo one stavke u stvarnom DOM-u koje su promijenjene, umjesto ažuriranja svih komponenti kao što to čine ostale aplikacije. Komponente su građevni blokovi svake React aplikacije. Jedna se aplikacija sastoji od više komponenti od kojih svaka ima svoju logiku i kontrolu te se mogu više puta koristiti u cijeloj aplikaciji, što smanjuje vrijeme razvoja.

Osim web, postoji i mogućnost izrade mobilnih aplikacija uz pomoć React Native biblioteke koja je izvedena iz samog React-a. Također postoje i namjenski alati za jednostavno otklanjanje pogrešaka koje je Facebook objavio kao proširenje za Chrome, čime je proces otklanjanja pogrešaka React web aplikacija brži i lakši.

React kombinira osnovne HTML i JavaScript koncepte s korisnim dodacima, što olakšava učenje. [8][9]

2.6. Node.js

Node.js je JavaScript izvršno okruženje otvorenog kôda i pokreće V8 JavaScript engine, što predstavlja jezgru Google Chrome-a.

Node.js pokreće se u samo jednom procesu bez kreiranja novih niti za svaki zahtjev. Također pruža set asinkronih I/O (engl. *Input/Output*) primitiva (što znači da su operacije poput čitanja i pisanja bajtova prema mrežnim konekcijama i od njih asinkrone i bilo koja operacija više razine mora biti izgrađena pomoću tih istih osnovnih operacija) u standardnoj biblioteci koje sprječavaju blokiranje JavaScript kôda.

Knjižnice u Node.js-u napisane su korištenjem neblokirajućih paradigmi. Kada se izvršava I/O (engl. *Input/Output*) operacija kao što je čitanje s mreže ili pristupanje bazi podataka ili sustavu datoteka, Node.js, umjesto da blokira nit i zaustavlja procesorske cikluse čekajući, nastavlja operaciju kada dobije povratnu informaciju od procesora. To Node.js-u omogućava rukovanje tisućama istovremenih veza s jednim poslužiteljem bez tereta istovremenog upravljanja nitima, što može predstavljati veliki gubitak vremena.

Velika prednost je što milijuni front-end programera koji pišu JavaScript za preglednike sada mogu pisati kod s poslužiteljske strane uz dodatak pisanja kôda s klijentske strane bez učenja kompletno novog jezika.

Node.js je platforma niskog nivoa. Kako bi se olakšalo pisanje kôda postoje tisuće biblioteka i okvira koje su zajednice izgradile za Node.js. Neki od njih su: AdonisJS, Egg.js, Express (okvir koji će se koristiti za izradu završnog rada), Fastify, FeatherJS, Gatsby, Hapi, Next.js. [10]

2.7. Express.js

Express.js je minimalan i fleksibilan web aplikacijski okvir koji pruža mogućnosti za web i mobilne aplikacije. Temelji se na jezgri Node.js http modula i Connect komponenti te se koristi kao međuprogram (engl. *middleware*). Express.js radi tako što ima ulaznu točku tj. glavnu datoteku i u toj se datoteci provode sljedeći koraci:

1. Uključuju se komponente treće strane i izrađeni moduli kao što su kontroleri, modeli, pomagači.
2. Konfiguriraju se postavke Express.js aplikacije kao što je predložak i ekstenzija svojih datoteka.
3. Definira se međuoprema kao što je upravljanje greškama, folder sa statičkim datotekama, kolačići i drugi parseri.
4. Definiranje ruta.
5. Spajanje na bazu podataka.
6. Pokretanje aplikacije.

Kada Express.js aplikacija radi, sluša zahtjeve. Svaki dolazeći zahtjev procesira se prema definiranom nizu međuopreme i rute počinju silazno. Važno je da rute počinju silazno jer to omogućava kontrolu toka izvršavanja. [11][12]

2.8. NoSQL

NoSQL ili *not only* SQL baze podataka su nerelacijske baze podataka i pohranjuju podatke drugačije od relacijskih tablica. Dolaze u različitim tipovima na temelju njihovih modela podataka. Imaju fleksibilne sheme i skaliraju se s velikom količinom podataka i opterećenjem korisnika. NoSQL baze podataka prvotno su izašle kasnih 2000-ih radi drastičnog smanjenja cijena pohrane podataka. NoSQL baze podataka omogućuju spremanje velike količine nestrukturiranih podataka, što doprinosi fleksiblinosti. Svaka NoSQL baza podataka ima svoje jedinstvene značajke. Na visokom nivou, mnogo NoSQL baza ima sljedeće značajke: fleksibilne sheme, horizontalno skaliranje, brže upite i lakše korištenje za programere. Neke od glavnih vrsta NoSQL baza podataka su dokumenti, grafovi, wide-column, key-value.

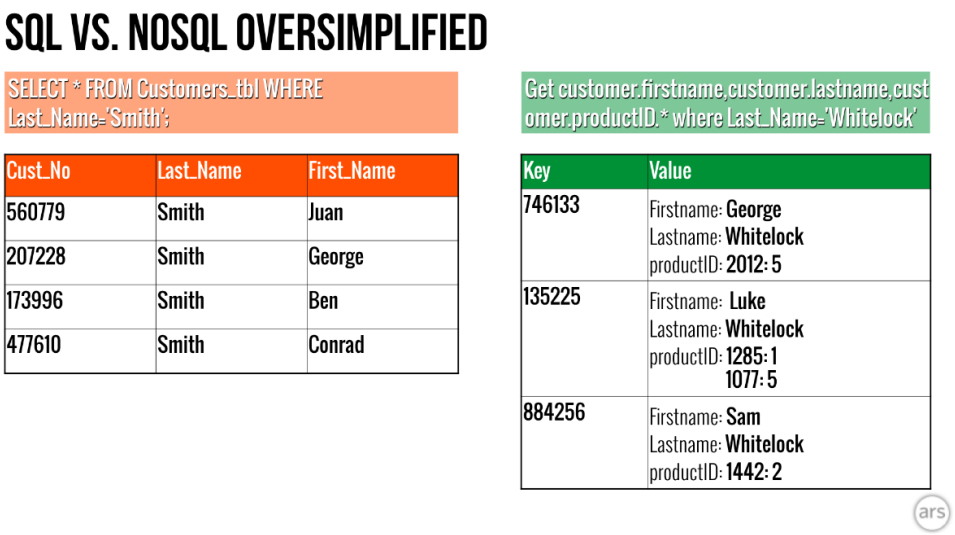
Dokumentne baze podataka spremaju podatke u dokumente slične JSON objektima, a svaki objekt sadrži par polja i vrijednosti. Vrijednosti mogu biti različitih tipova kao što su stringovi, brojevi, bool vrijednosti (true ili false), niz (*engl. Array*) ili objekti.

Grafičke baze podataka pohranjuju podatke u čvorove i rubove. Čvorovi obično pohranjuju informacije o ljudima, mjestima i stvarima, dok rubovi pohranjuju informacije o odnosima između čvorova.

Wide-column baza podataka organizira pohranu podataka u fleksibilne stupce koji su u suštini raširenu u više servera ili čvorova baza podataka koristeći višedimenzionalno mapiranje za referenciranje podataka po stupcu, retku i vremenskoj oznaci.

Key-value baze podataka su jednostavnija vrsta baze podataka u kojoj svaka stavka sadrži vrijednosti i ključeve. [13]

Za razliku od NoSQL baza podataka, relacijske baze podataka koriste strukturirani jezik upita, imaju unaprijed definiranu shemu i bazirane su na tabelama. Jedna se vrijednost pohranjuje u jednoj ćeliji. Slika 2 prikazuje razliku između relacijskih i nerelacijskih baza podataka.



Slika 2. Razlika između SQL i NoSQL baze podataka

Izvor: https://cdn.arstechnica.net/wp-content/uploads/2016/03/sqlvnosql.png (06.02.2022)

2.9.Firebase

Google Firebase je rješenje za razvoj mobilnih i web aplikacija. Pruža niz alata koji pomažu u razvoju aplikacija. Sadrži NoSQL baze podataka te pohranjuje podatke i informacije u JSON formatu. Firebase pruža programerima gotova rješenja ovjeravanja autentičnosti korisnika, usluge poslužitelja web aplikacija sa svim sigurnosnim certifikatima, pohranu podataka, integrirani Google Analytics. [14] Ovjeravanje korisnika omogućuje prijavu u aplikaciju preko postojećih računa kao što su Facebook, Twitter, Google ili Github te ima sveobuhvatnu sigurnost koju podržava Google. [15]

2.10. Cloudinary

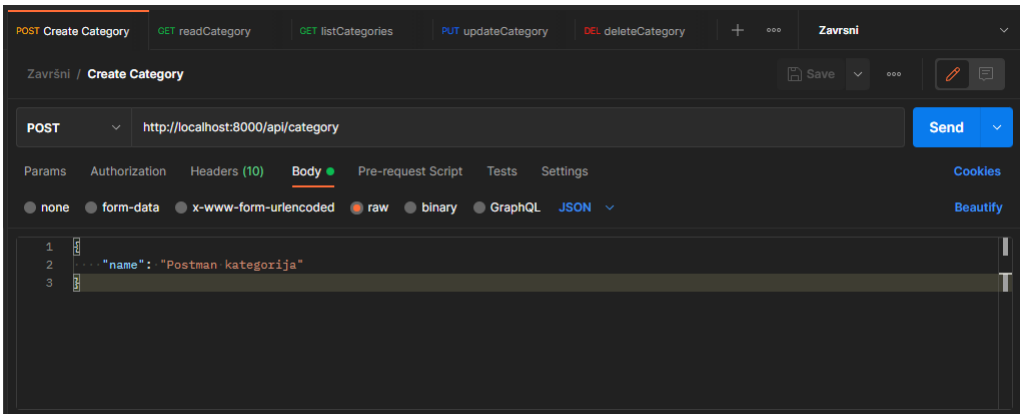
Cloudinary je *end-to-end* rješenje za upravljanje slikama i videozapisima za web i mobilne aplikacije. Pokriva sve od prijenosa slika i videa, pohranjivanja, manipulacije te optimizacije do isporuke. Slike se pohranjuju u oblaku i on omogućuje njihovu manipulaciju bez instalacije drugih programa. Mediji se isporučuju putem mreže za brzu isporuku sadržaja (engl: *Content Delivery Network „CND“*). Cloudinary nudi sveobuhvatna sučelja aplikacijskih programa te mogućnosti administracije preko Cloudinary konzole. Administracija se može integrirati s web ili mobilnom aplikacijom. Administrator preko

Cloudinary konzole upravlja i manipulira sa svim slikama koje su prenesene preko web aplikacije. [16]

3. STRUKTURA APLIKACIJE I ULOGE

3.1. Struktura

Aplikacija je strukturirana na dva glavna dijela: poslužiteljski dio (engl. *Backend*) i klijentski dio (engl. *Frontend*). Poslužiteljski dio aplikacije sastoji se od konfiguracijskih datoteka za Firebase, Cloudinary i MongoDB sustave te inicijalizacijskih datoteka koje se spajaju na bazu podataka, Firebase sustav. Sadrži modele po kojima je definirana shema za košaricu, narudžbu, korisnika, proizvod, kategoriju i kupon te se prema njima stvara model koji se poziva kod upisa u nerelacijsku bazu podataka. Osim modela, poslužiteljski dio aplikacije također je zadužen za rute koje definiraju korisničke zahtjeve prema aplikaciji. Mogu biti *GET, POST, PUT, DELETE* te sadrže poveznicu koja se poziva u klijentskom dijelu aplikacije. Kontroleri su također definirani u poslužiteljskom dijelu aplikacije. Kontroler metode obrađuju zahtjeve koje šalje klijentski dio aplikacije i vrše akcije na samoj bazi podataka. Provjera ispravnosti kontrolera i ruta radi se preko Postman aplikacije kao na Slici 3 ili preko klijentskog dijela aplikacije. Za sve rute koje manipuliraju glavnim podacima koji se prikazuju na stranici potrebno je da korisnik bude logiran kao administrator.



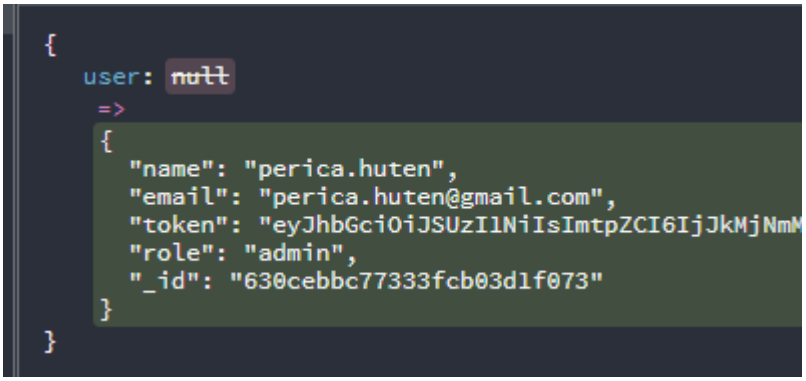
Slika 3. Postman zahtjev za izradu kategorije

Izvor: Autor

Klijentski dio aplikacije je ono što administrator i korisnik aplikacije vide i koriste. Klijentski dio aplikacije sadrži klijentske konfiguracije za Firebase i Cloudinary servise, React komponente, stranice (engl. *Pages*) koje se izravno prikazuju na aplikaciji, funkcije s API pozivima prema poslužiteljskom dijelu aplikacije i Redux reduktora za postavljanje globalnog stanja u aplikaciji.

Firebase konfiguracija u klijentskom dijelu aplikacije sadrži glavne ključeve i autorizacijske domene za Firebase servis koji su potrebni za inicijalizaciju Firebase-a. Cloudinary konfiguracija definira ime oblaka (engl. *Cloud*) koji se koristi za prijenos slika i konfiguriranje medija urednika (engl. *Media editor*) koji se koristi za personalizaciju proizvoda.

React komponente u klijentskom dijelu aplikacije rade podjelu sučelja na neovisne dijelove kôda koji se koriste više puta na stranicama aplikacije. Stranice vraćaju HTML kod koji React prikazuje korisniku aplikacije. Stranice aplikacije osim React komponenti sadrže metode koje se pozivaju na stranici prilikom učitavanja ili interakcije korisnika sa stranicom. Te metode koriste funkcije s API pozivima prema poslužiteljskom dijelu aplikacije za dohvaćanje informacija iz baze podataka. Ovisno o vrsti metode ponekad je potrebno spremiti informaciju u globalno stanje za što se koriste Redux reduktori. Npr. stranica za prijavu korisnika u aplikaciju poziva metodu koja sadrži reduktor koji uzima informacije prijavljenog korisnika te ga postavlja u globalno stanje. Primjer postavljanja globalnog stanja može se vidjeti na Slici 4.



Slika 4. Korisnički Redux reduktor

Izvor: Autor

3.2. Uloge

Aplikacija je podijeljena na 3 korisničke uloge: korisnik aplikacije koji nije prijavljen (gost), prijavljeni korisnik i administrator.

Korisnik koji nije prijavljen u aplikaciju ima mogućnost pregleda proizvoda, dodavanja proizvoda u košaricu koja se sprema lokalno na računalu. Također ima mogućnost uređivanja proizvoda koji su dodani u korisničku košaricu sa slikom koja se prenosi na Cloudinary servis. Ako korisnik želi kupiti proizvod pomoću kartice ili pouzećem, mora se prijaviti u aplikaciju pomoću e-mail računa. Prijava se odvija putem Firebase sustava.

Korisnik koji je prijavljen u aplikaciju ima mogućnost kupovine proizvoda karticom ili mogućnost plaćanja pouzećem. Košarica prijavljenog korisnika sprema se u bazu podataka na dulje čuvanje. Registrirani korisnik ima mogućnost izmjene svoje lozinke za prijavu te u slučaju da zaboravi lozinku može putem aplikacije kod prijave zatražiti automatiziranu poruku za zaboravljenu lozinku. Kod pregledavanja proizvoda korisnik ima mogućnost ocjenjivanja proizvoda ocjenom od 1 do 5, što daje mogućnost drugim korisnicima da filtriraju proizvode po dodijeljenim ocjenama. Prijavljeni korisnik ima popis želja te može dodati proizvode na tu listu koja se prikazuje na korisničkoj nadzornoj ploči. Na nadzornoj ploči također se nalazi i povijest narudžbi pomoću kojih korisnik može pratiti trenutni status narudžbe i preuzeti račun narudžbe u PDF formatu.

Administrator aplikacije ima najviše ovlasti od svih korisnika, sve prijašnje navedene mogućnosti standardnog korisnika i više. Administrator stranice zadužen je za kreiranje i upravljanje proizvodima, dodaje i upravlja s kategorijama i potkategorijama, može dodavati kupone za popust na stranici i, najvažnije, administrator vidi potpunu povijest narudžbi svih korisnika te mijenja status narudžbe koji korisnik može vidjeti na svojoj nadzornoj ploči.

4. INICIJALIZACIJA APLIKACIJE

Aplikacija koristi Visual Studio Code uređivač teksta te je potrebno kreirati klijentski i poslužiteljski dio aplikacije unutar iste datoteke. Pokretanjem komande „npx create-react-app client“ inicijalizira se prednji dio aplikacije. Potrebno je ručno kreirati datoteku „server“ pomoću Visual Studio Code upravitelja datoteka i ručno se pozicionirati u server datoteku s

„cd server“ komandom. Komandom „npm init -y“ kreira se početna konfiguracija za server s „package.json“ datotekom unutar koje će se prikazati sve potrebne informacije o projektu i instalirani moduli. Moduli koje aplikacija koristi za poslužiteljski dio su:

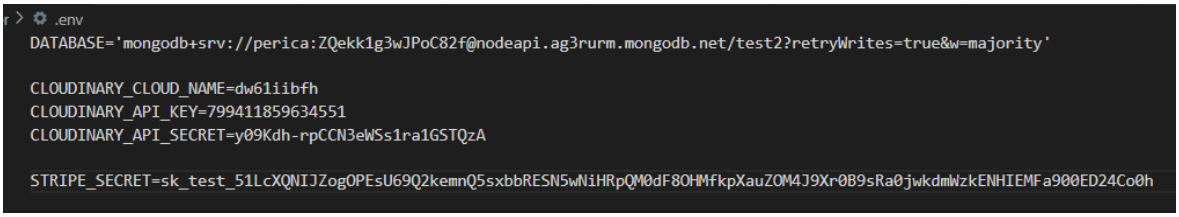
- body-parser – raščlanjuje dolazne zahtjeve u međuprogramu i sprema podatke u req.body svojstvo
- cloudinary – omogućava brzu i jednostavnu integraciju aplikacije s Cloudinary web servisom
- cors – omogućava dijeljenje resurse s različitim izvorima (engl. *Cross Origin Resource Sharing*)
- dotenv – modul koji učitava varijable okruženja iz .env datoteke te ih pohranjuje u process.env
- express – glavni web okvir za node.js
- firebase-admin – pruža sve potrebne alate za razvijanje aplikacije pomoću firebase servisa; koristi se za dohvaćanje podataka korisnika i inicijalizaciju servisa
- Mongoose – alat za MongoDB bazu podataka, služi za spajanje s bazom podataka i pomaže kod kreiranja mongoose modela
- Morgan – međuprogram koji služi kao zapisnik za HTTP zahtjeve
- Nodemon – služi za ponovno pokretanje aplikacije u slučaju izmjena datoteka kod spremanja
- Slugify – biblioteka koja stvara slug koji se može koristiti u linkovima, npr: „bijela majica“ -> „bijela-majica“
- Stripe – biblioteka za spajanje na Stripe servis za plaćanje
- Uniqueid – služi za generiranje unikatnih identifikatora

Veći dio modula na korisničkom dijelu aplikacije koristi se za dizajn, no neki su moduli potrebni za funkcionalnost aplikacije. Primarno za rad s Cloudinary servisom, ispis PDF

dokumenata, Firebase autentifikacija, Axios modul koji dodaje HTTP klijent temeljen na obećanju (engl. *promise*) za preglednik.

4.1. Podešavanje konfiguracijskih datoteka i spajanje na bazu podataka

Budući da aplikacija koristi više vanjskih servisa kao što su MongoDB baza podataka, Stripe servis za plaćanje, Cloudinary servis za prijenos i prikaz slika i Firebase servis za autentifikaciju korisnika, potrebno je prilagoditi aplikaciju prema njihovim potrebama. Unutar .env datoteke su svi potrebni ključevi za pristup bazi, Cloudinary servisu i Stripe servisu kao što se može vidjeti na Slici 5.



Slika 5. Poslužiteljeva .env datoteka

Izvor: autor

Potrebno je konfigurirati i inicijalizirati Firebase na strani poslužitelja kako bi se omogućilo slanje e-mailova korisnicima prilikom registracije ili u slučaju da zaborave lozinku. Korištenjem Firebase sustava nemoguće je kreirati profil bez pravog e-mail profila jer je potrebno potvrditi registraciju korisnika koja dolazi mailom.

Spajanje na bazu podataka odvija se unutar glavne datoteke na strani poslužitelja. Potrebno je spajanje na bazu jer se sve potrebne informacije o proizvodima i korisnicima dohvaćaju iz baze podataka. Kod za spajanje na bazu može se vidjeti na Slici 6.

```
// mongoose baza
mongoose
  .connect(process.env.DATABASE, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("BAZA SPOJENA"))
  .catch((err) => console.log("GREŠKA KOD SPAJANJA", err));
```

Slika 6. Spajanje na bazu unutar Server.js datoteke

Izvor: autor

String potreban pri *.connect* metodi dohvaća se iz *.env* datoteke pod DATABASE imenom, što se može vidjeti na Slici 5 te se dohvaća pomoću *dotenv* modula koji pohranjuje sve elemente iz *.env* datoteke u *process.env* objekt. Izgled kolekcije mongoDB baze podataka može se vidjeti na slici broj 7.

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
carts	1	207B	207B	36KB	1	36KB	36KB
categories	3	345B	115B	36KB	2	72KB	36KB
coupons	1	118B	118B	20KB	2	40KB	20KB
orders	2	2.03KB	1.02KB	36KB	1	36KB	36KB
products	4	1.92KB	492B	36KB	3	108KB	36KB
subs	2	238B	119B	36KB	2	72KB	36KB
users	4	815B	204B	36KB	2	72KB	36KB

Slika 7. Kolekcija baze podataka

Izvor: autor

5. IMPLEMENTACIJA POSLUŽITELJSKE STRANE APLIKACIJE

5.1. Modeli

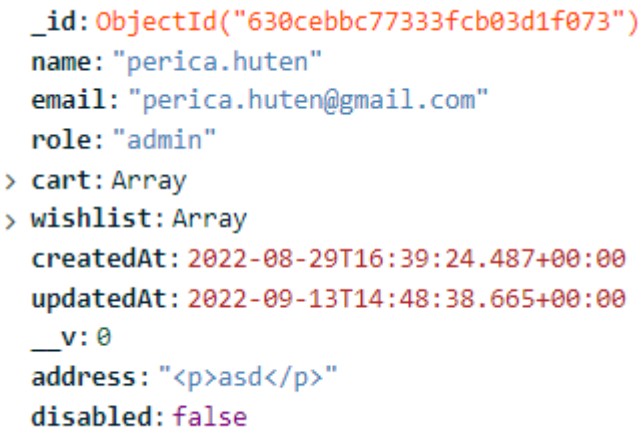
Modeli u aplikaciji prikazuju kako će podaci izgledati u samoj bazi podataka. Kod stvaranja modela izrađuje se mongoose shema koja se može vidjeti na Kôdu 1.

```
const mongoose = require("mongoose");

const categorySchema = new mongoose.Schema(
  {
    name: {
      type: String,
      trim: true,
      required: "Ime je potrebno",
      minlength: [2, "Prekratko"],
      maxlength: [32, "Predugo"],
    },
    slug: {
      type: String,
      unique: true,
      lowercase: true,
      index: true,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Category", categorySchema);
```

Kôd 1. Model za korisnika

Izvor: autor

Model se definira kao objekt. Prilikom pozivanja modela u kontroleru kreira se novi objekt koji se zatim prosljeđuje bazi podataka u JSON obliku. Moguće je napraviti validaciju unutar samog modela, što se može vidjeti u kôdu 1 na imenu korisnika (engl. *name*) pod svojstvima *required*, *maxlength* i *minlength*. Model postoji za svaki objekt koji se upisuje u bazu podataka, pa čak i za korisnike jer se autentifikacija radi putem Firebase servisa koji dohvaća i upisuje informacije u bazu podataka. Primjer korisnika u bazi podataka može se vidjeti na Slici 8.



Slika 8. Korisnik u bazi podataka

Izvor. autor

5.2. Rute

Poslužiteljski dio s korisničkim dijelom aplikacije komunicira pomoću ruta. Svaka ruta ima definirani zahtjev koji može biti za kreiranje, čitanje, ažuriranje ili brisanje podataka (engl. *POST*, *GET*, *PUT*, *DELETE*), put koji se poziva u korisničkom dijelu aplikacije, dodatni parametri za provjeru npr. je li korisnik prijavljen ili ima ulogu administratora i kontroler metodu koja kreira objekte na temelju modela i prosljeđuje podatke od korisničke strane prema bazi podataka.

```
router.post("/category", authCheck, adminCheck, create);
router.get("/category/:slug", read);
router.put("/category/:slug", authCheck, adminCheck, update);
router.delete("/category/:slug", authCheck, adminCheck, remove);
```

Kôd 2. Rute CRUD metode kategorija

Izvor: autor

5.3.Kontroleri

Kontroleri dobivaju podatke od klijentske strane aplikacije u *req.body* objektu te stvaraju novi objekt ako je kontroler namijenjen za izmjenu ili kreiranje novih podataka unutar baze podataka. Kontroleri koriste kolekcijske metode za mongo bazu podataka kao što su

findOne, findOneAndDelete, findOneAndUpdate i druge. Nakon dohvaćanja podataka kontroler vraća poruku korisničkoj strani aplikacije putem *res* objekta i pokušava uhvatiti greške unutar *try-catch* izjave te vraća status ovisno o uhvaćenoj grešci. Primjer kontroler metode za kreiranje kategorije može se vidjeti u Kôdu 3.

```
exports.create = async (req, res) => {
  try {
    const { name } = req.body;
    res.json(await new Category({ name, slug: slugify(name) }).save());
  } catch (err) {
    console.log(err);
    res.status(400).send("Stvaranje kategorije neuspješno");

  }
};
```

Kôd 3. Kontroler metoda za kreiranje kategorije

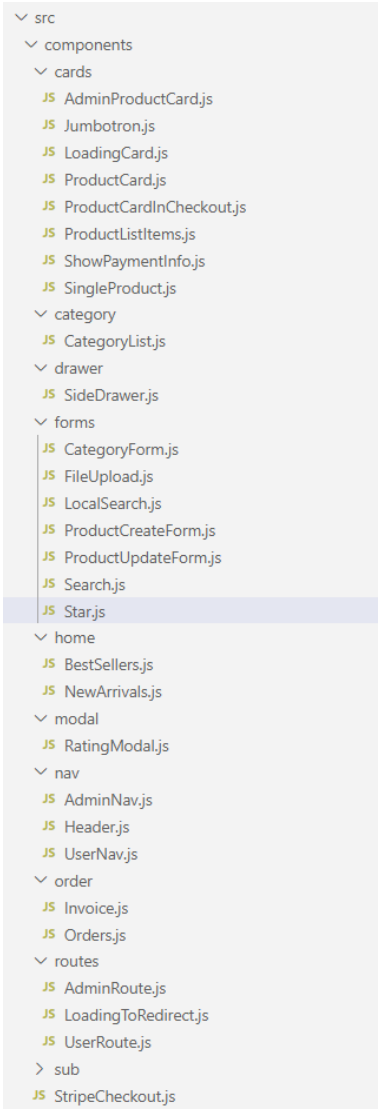
Izvor: autor

6. IMPLEMENTACIJA KLIJENTSKE STRANE APLIKACIJE

Klijentska strana (engl. *frontend*) aplikacije napravljena je od React komponenti koje se koriste u stranicama te prikazuju u aplikaciji, funkcija koje komuniciraju s klijentskim dijelom aplikacije putem HTTP zahtjeva, glavnih stranica u kojima su sadržane određene funkcije za prikaz i manipulaciju podataka te React reduktora (engl. *reducer*)

6.1. Komponente

React komponente su višekratno upotrebljivi i neovisni dijelovi kôda koji vraćaju HTML kôd i koriste se u cijeloj aplikaciji za prikaz proizvoda na više načina, listu kategorija, košaricu, forme za kreiranje i ažuriranje podataka, navigaciju te prikaz određenih podataka. Popis svih komponenti se može vidjeti na sljedećoj slici.



Slika 9. Popis komponenti klijentske strane aplikacije

Izvor: autor

React komponente su iskoristive jer mogu kod pozivanja zahtijevati objekt. Komponenta za prikaz proizvoda zahtijeva objekt proizvod te dohvaća sve potrebne informacije za taj proizvod iz baze podataka i vraća HTML kôd koji se prikazuje na stranici. Unutar te komponente također je definirana funkcija za dodavanje proizvoda u košaricu. HTML kod koji vraća komponenta za proizvod može se vidjeti u Kôdu 4.

```
return (
    <>
      {product && product.ratings && product.ratings.length > 0 ? (
        showAverage(product)
      ) : (
        <div className="text-center pt-1 pb-3">Nema ocjena</div>
      )}

      <Card
        cover={
          <img
            src={images && images.length ? images[0].url : DefaultImg}
            style={{ height: "150px", objectFit: "cover" }}
            className="p-1"
          />
        }
        actions={[
          <Link to={`/product/${slug}`}>
            <EyeOutlined className="text-info" /> <br /> Vidi proizvod
          </Link>,
          <Tooltip title={tooltip}>
            <a onClick={handleAddToCart} disabled={product.quantity < 1}>
              <ShoppingCartOutlined className="text-info" /> <br />
              {product.quantity < 1 ? "Rasprodano" : "Dodaj u košaricu"}
            </a>
          </Tooltip>,
        ]}
      >
        <Meta
          title={`${title} - ${price} kn`}
          description={`${description && description.substring(0, 40)}...`}
        />
      </Card>
    </>
  );
```

Kôd 4. Povratni kod za komponentu proizvoda

Izvor: autor

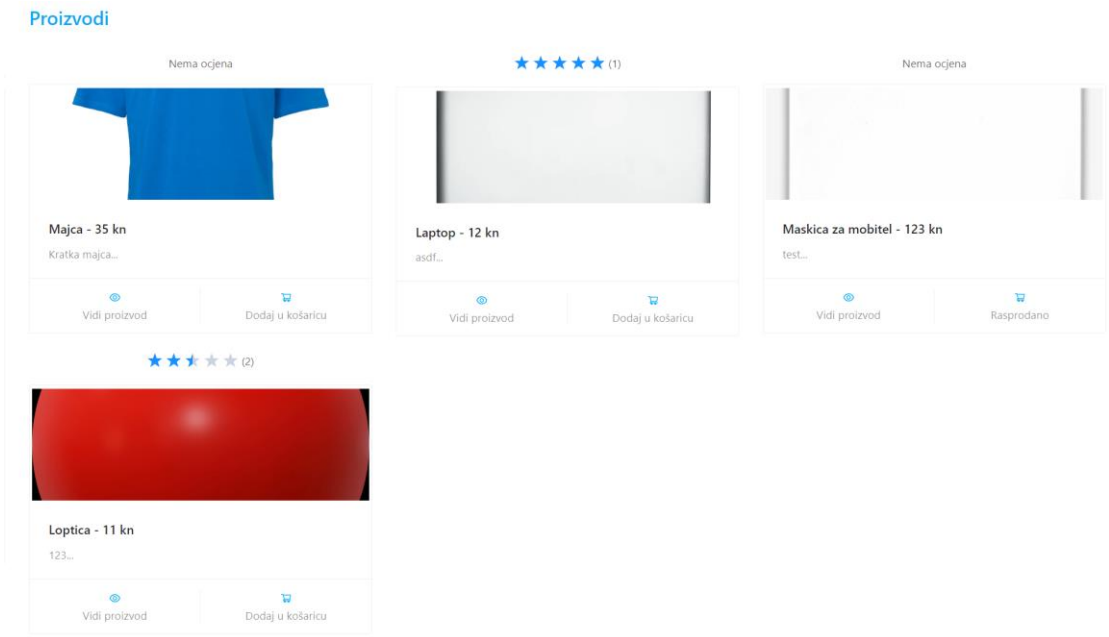
Povratni kôd sadrži još jednu React komponentu Card koja je uvozna komponenta s *ANTDesign* modula koji sadrži već napravljene komponente za ljepši vizualni izgled aplikacije.

Komponente su višekratno iskoristive te se mogu pozivati više puta za različite proizvode. U aplikaciji se komponenta poziva unutar *.map* funkcije koja prolazi kroz niz proizvoda (engl. *Array*) i poziva komponentu za svaki proizvod kao što je prikazano na Kôdu 5. Konačan izgled komponenti na stranici može se vidjeti na Slici 10.

```
<div className="col-md-9 pt-2">
      {loading ? (
            <h4 className="text-danger">Učitavanje...</h4>
      ) : (
            <h4 className="text-info">Proizvodi</h4>
      )}

      {products.length < 1 && <p>Proizvodi nisu pronađeni</p>}

      <div className="row pb-5">
            {products.map((p) => (
                  <div key={p._id} className="col-md-4 mt-3">
                        <ProductCard product={p} />
                  </div>
            ))}
      </div>
</div>
```

Kôd 5. Korištenje komponente unutar HTML kôda stranice

Izvor. autor



Slika 10. Izgled komponenti na stranici

Izvor: autor

6.2.Funkcije

Funkcije klijentske strane aplikacije zadužene su za komunikaciju s poslužiteljskom stranom. Ovisno o tipu zahtjeva mogu zahtijevati podatke kao što su autorizacijski žeton (engl. *token)*, ime proizvoda, identifikacijski broj proizvoda i mnoge druge informacije. Za dohvaćanje zahtjeva koristi se Axios s popratnim metodama kao što su *.get, .put, .post, .delete.* Vrsta metode ovisi o ruti napisanoj u poslužiteljskom dijelu aplikacije na koju se Axios zahtjev nadovezuje. Primjer Axios funkcije i rute na koju se ta funkcija nadovezuje može se vidjeti na Slici 11. Axios dohvaća glavni link za zahtjev putem .env datoteke, dok se kategorija i autorizacijski žeton prosljeđuju prilikom pozivanja funkcije.

```
export const createCategory = async (category, authtoken) =>
  await axios.post(`${process.env.REACT_APP_API}/category`, category, {
    headers: {
      authtoken,
    },
  });

router.post("/category", authCheck, adminCheck, create);
```

Slika 11. Funkcija za kreiranje kategorije i ruta za kreiranje kategorije

Izvor: autor

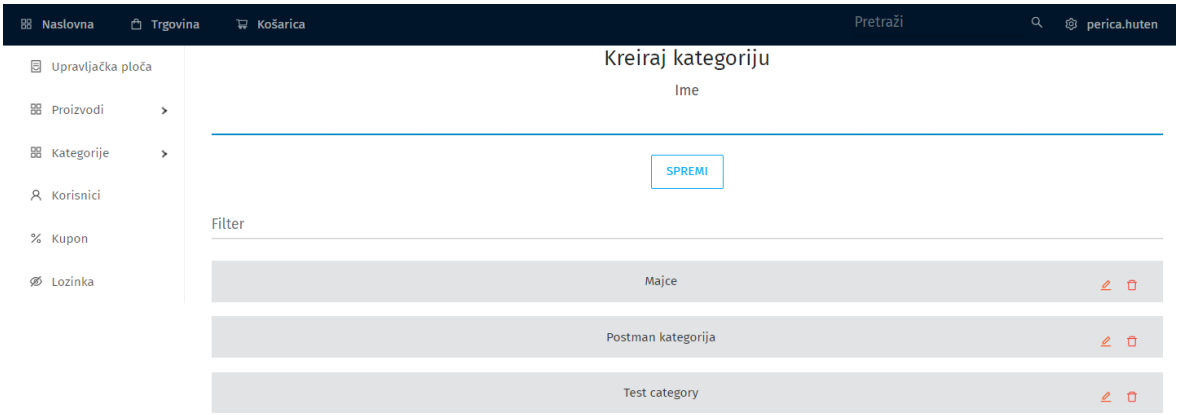
6.3. Stranice (engl. *Pages*)

Nakon kreiranja, funkcije se pozivaju u metodama napisanim na stranicama aplikacije. Na stranicama za kreiranje i izmjenu podatka koristi se kuka stanja (engl. *State Hook*) koja sprema informacije upisane u formu. Pritiskom na gumb za spremanje forme pokreće se *handleSubmit* metoda koja poziva funkciju za kreiranje ili ažuriranje kojoj se zatim prosljeđuju prethodno spremljene informacije u stanju. U Kôdu 6 može se vidjeti *handleSubmit* metoda za kreiranje kategorije, a na Slici 12 izgled stranice za kreiranje kategorije.

```
const handleSubmit = (e) => {
  e.preventDefault();
  // console.log(name);
  setLoading(true);
  createCategory({ name }, user.token)
    .then((res) => {
      // console.log(res)
      setLoading(false);
      setName("");
      toast.success(`"${res.data.name}" is created`);
      loadCategories();
    })
    .catch((err) => {
      console.log(err);
      setLoading(false);
      if (err.response.status === 400) toast.error(err.response.data);
    });
};
```

Kôd 6. *handleSubmit* metoda za kreiranje kategorije

Izvor: autor



Slika 12. Stranica za kreiranje kategorija

Izvor: autor

Stranica za kategorije sadrži i listu kategorija koja se automatski ažurira kod kreiranja nove kategorije uz pomoć *loadCategories()* funkcije napisane u *handleSubmit* metodi.

Budući da React radi na način da izmjenjuje samo potrebne informacije na aktivnoj stranici, potrebno je koristiti *useEffect* funkciju. *useEffect* omogućava aplikaciji da izvršava kod prilikom otvaranja stranice, što je slično *componentDidMount* i *componentDidUpdate*

funkcijama. Uz pomoć *useEffect* funkcije radi se zahtjev prema bazi podataka za dohvaćanje informacija koje se prikazuju na stranici. Primjer se može vidjeti u Kôdu 7.

```
useEffect(() => {
  loadUserOrders();
}, []);

const loadUserOrders = () =>
  getUserOrders(user.token).then((res) => {
    console.log(JSON.stringify(res.data, null, 4));
    setOrders(res.data);
  });
```

Kôd 7. Učitavanje povijesti narudžbi kod otvaranja stranice

Izvor: autor

useEffect poziva *loadUserOrders* metodu koja radi zahtjev prema bazi podataka za dohvaćanje liste korisničkih narudžbi.

6.4.Reduktori (engl. *Reducers*)

Redux reduktor kreira globalno stanje koje pomaže u dohvaćanju određenih informacija kao što su korisnički podaci i korisnički žeton za autentifikaciju. Za korištenje React Redux-a potrebna je izmjena korijenske React index.js datoteke prema sljedećem kôdu:

```
const store = createStore(rootReducer, composeWithDevTools());

ReactDOM.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>,
  document.getElementById("root")
);
```

Kôd 8. Index.js datoteka s Redux store state spremnikom

Izvor: autor

Aplikacija koristi reduktore za globalno spremanje stanja košarice, narudžbi s plaćanjem po pouzeću, kupona, prikaza košarice kod dodavanja proizvoda, pretrage proizvoda i

korisničkih informacija. Reduktor za spremanje korisničkih informacija može se vidjeti u Kôdu 9.

```
export const userReducer = (state = null, action) => {
  switch (action.type) {
    case "LOGGED_IN_USER":
      return action.payload;
    case "LOGOUT":
      return action.payload;
    default:
      return state;
  }
};
```

Kôd 9. Korisnički reduktor s 2 slučaja za prijavu i odjavu

Izvor: autor

Zadano stanje reduktora postavljeno je na *null* vrijednost te se izmjenjuje samo kada je pozvana *dispatch* funkcija. Određuje se tip *dispatch* funkcije, što se referencira na određeni case u reduktoru i podaci u korisnom teretu (engl. *payload*) koji će se spremiti u globalno stanje kao što je prikazano na Kôdu 10.

```
createOrUpdateUser(idTokenResult.token)
  .then((res) => {
    dispatch({
      type: "LOGGED_IN_USER",
      payload: {
        name: res.data.name,
        email: res.data.email,
        token: idTokenResult.token,
        role: res.data.role,
        _id: res.data._id,
      },
    });
    roleBasedRedirect(res);
  })
  .catch((err) => console.log(err));
```

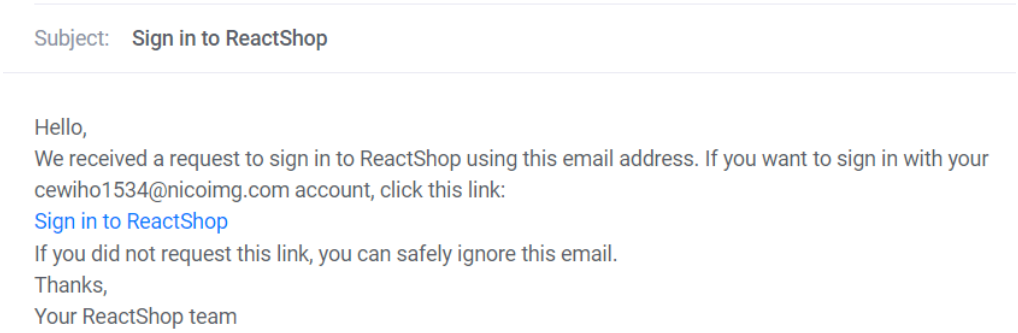
Kôd 10. Korištenje reduktora unutar Firebase funkcije

Izvor: autor

7. REGISTRACIJA I PRIJAVA KORISNIKA

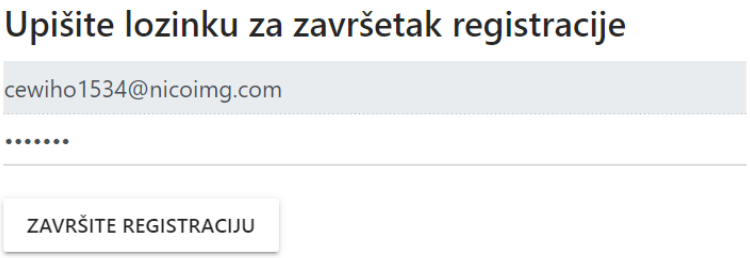
Registracija korisnika putem Firebase sustava radi se preko korisničkog dijela aplikacije na stranici za registraciju. Korisnik za prvi dio registracije upisuje samo e-mail profila. U

slučaju neispravnog upisa e-maila sustav obavještava korisnika porukama o grešci prilikom upisa. Nakon ispravnog upisa aplikacija poziva *sendSignInLinkToEmail* metodu iz firebase modula te šalje korisniku e-mail koji se može vidjeti na Slici 13. Mail sadrži poveznicu koja otvara stranicu za upis lozinke, kao što se vidi na Slici 14.



Slika 13. Email za registraciju

Izvor: autor



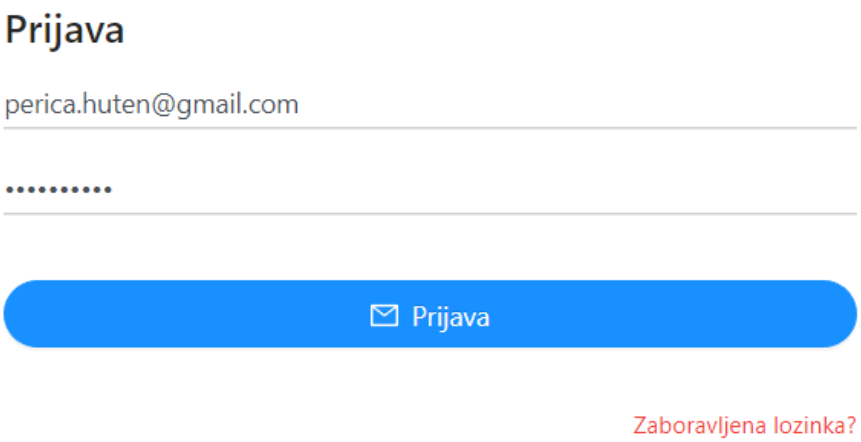
Slika 14. Završetak registracije

Izvor: autor

Nakon završetka registracije korisnik se upisuje u bazu podataka. Baza podataka ne sadrži lozinke korisnika jer su one kriptirane i spremljene unutar Firebase-a te nema potrebe za spremanjem lozinke na dva mjesta.

Za prijavu korisnici koriste e-mail adresu i lozinku od registracije na aplikaciju. Aplikacija kod prijave prosljeđuje zahtjev Firebase sustavu da se ažurira korisnik s novim identifikacijskim žetonom te se informacije prosljeđuju putem reduktora u globalno stanje.

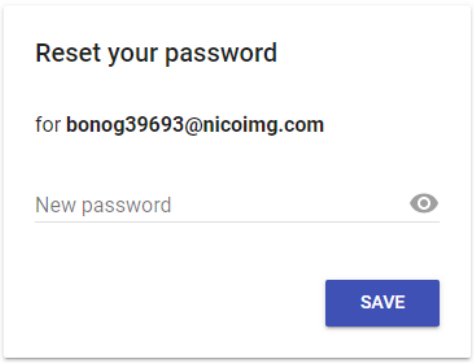
Prilikom odjave informacije se brišu iz globalnog stanja. Forma za prijavu može se vidjeti na Slici broj 15.



Slika 15. Forma za prijavu

Izvor: autor

Korisnik može zatražiti izmjenu lozinke u slučaju da ju zaboravi. Kod upisa e-maila i slanja zahtjeva korisnik dobiva poveznicu na e-mail, putem koje izmjenjuje lozinku za prijavu na aplikaciju kao što se vidi na Slici broj 16.



Slika 16. Izmjena zaboravljene lozinke

Izvor: autor

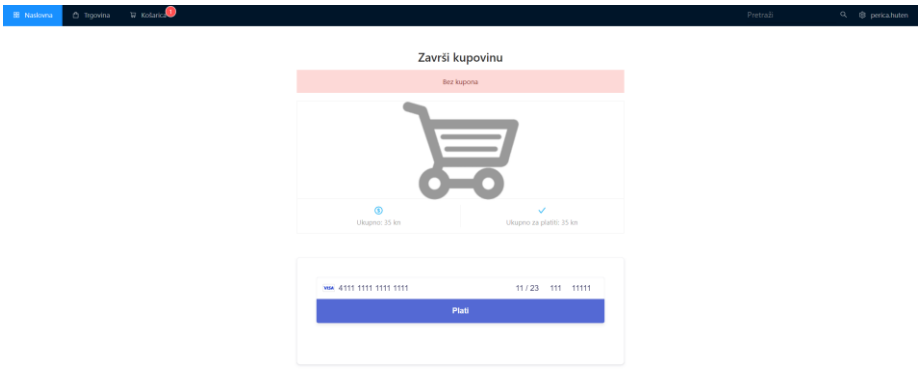
8. PERSONALIZACIJA I KUPOVINA PROIZVODA

Nakon odabira željenog proizvoda korisnik dobiva mogućnost personalizacije proizvoda. Personalizacija proizvoda odvija se putem *Cloudinary* medija urednika. Korisnik može personalizirati svaki proizvod u košarici. Sučelje za personalizaciju proizvoda može se vidjeti na Slici 17.

Slika 17. Personalizacija proizvoda

Izvor: autor

Nakon opcionalne personalizacije proizvoda korisnik završava narudžbu. Narudžba može biti plaćanje pouzećem, pri čemu korisnik upisuje adresu dostave i mogući kupon za popust, ili plaćanje karticom. Kod plaćanja karticom korisnik unosi adresu i mogući kupon i kreće na plaćanje koje se provodi u aplikaciji putem Stripe sustava, što se može vidjeti na Slici 18.



Slika 18. Plaćanje

Izvor: autor

Nakon uspješnog plaćanja aplikacija stvara novu narudžbu i pohranjuje ju u bazu podataka prema sljedećem kôdu. Stripe sustav zatim potvrđuje i pohranjuje naplatu u svoju bazu podataka.

```
createOrder(payload, user.token).then((res) => {
  if (res.data.ok) {
    // pražnjenje košarice iz lokalnog spremišta
    if (typeof window !== "undefined") localStorage.removeItem("cart");
    // pražnjenje košarice iz globalnog stanja
    dispatch({
      type: "ADD_TO_CART",
      payload: [],
    });
    // kupon se resetira ako je bio zadan
    dispatch({
      type: "COUPON_APPLIED",
      payload: false,
    });
    // košarica se briše iz baze podataka
    emptyUserCart(user.token);
  }
});
```

Slika 19. Pohranjivanje narudžbe u bazu podataka

Izvor: autor

9. ZAKLJUČAK

Aplikacija je izrađena s MERN stog tehnologijama i tehnologijama trećih strana kao što su Firebase, Cloudinary i Stripe za lakše upravljanje podacima i lakšu izvedbu aplikacije.

Poslužiteljski dio aplikacije izrađen je pomoću Express.js i Node.js web aplikacijskog okvira. Korištena je nerelacijska MongoDB baza podataka

Klijentski dio aplikacije izrađen je pomoću React.js biblioteke uz korištenje Redux-a za spremanje globalnih stanja, Firebase za prijavu i registraciju korisnika, Cloudinary koji omogućava prijenos, manipulaciju i personalizaciju slika prema korisničkim zahtjevima.

Prednost korištenja MERN stoga za izradu web aplikacije je pisanje kôda u JavaScript programskom jeziku za sve četiri tehnologije stoga.

10.POPIS LITERATURE

[1] Deloitte - Mass-to-order: The rise of mass personalisation

Dostupno na: https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/consumer-business/ch-en-consumer-business-made-to-order-consumer-review.pdf (03.02.2022)

[2] Salesforce – State of Marketing

Dostupno na: https://c1.sfdcstatic.com/content/dam/web/en_us/www/assets/pdf/datasheets/salesforce-research-fourth-annual-state-of-marketing.pdf (03.02.2022)

[3] © 2022 Microsoft – Visual Studio Code FAQ

Dostupno na: https://code.visualstudio.com/docs/supporting/faq (03.02.2022)

[4] © 2022 Microsoft - IntelliSense

Dostupno na: https://code.visualstudio.com/docs/editor/intellisense (04.02.2022)

[5] © 2005-2022 Mozilla and individual contributors - HTML

Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTML (04.02.2022)

[6] © 2005-2022 Mozilla and individual contributors - JavaScript

Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (05.02.2022)

[7] 2021 MongoDB, Inc.

Dostupno na: https://www.mongodb.com/mern-stack (05.02.2022)

[8] Chinmayee Deshpande – What is ReactJS

Dostupno na: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs (06.02.2022)

[9] S Aggarwal - International Journal of Recent Research Aspects - ReactJS

Dostupno na: http://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf (06.02.2022)

[10] © OpenJS Foundation – Node.js

Dostupno na: https://nodejs.dev/learn (06.02.2022)

[11] Azat Mardan (May 28, 2014). Express.js Guide: The Comprehensive Book on Express.js

Dostupno na: https://books.google.hr/books?id=5eGRAwAAQBAJ&lpg=PP6&ots=nkpdx-clKK&dq=express.js&lr&pg=PA14#v=onepage&q=express.js&f=false (06.02.2022)

[12] 2017 StrongLoop, IBM, and other expressjs.com contributors - ExpressJS

Dostupno na: https://expressjs.com/ (06.02.2022)

[13] Lauren Schaefer, MongoDB Developer Advocate. - What is NoSQL

Dostupno na: https://www.mongodb.com/nosql-explained (06.02.2022)

[14] What is Google Firebase and how to use it for application development

Dostupno na: https://www.ashutec.com/blog/what-is-google-firebase-and-how-to-use-it-for-application-development-d4c84d9d0207 (09.09.2022)

[15] Firebase Google

Dostupno na: https://firebase.google.com/products/auth (09.09.2022)

11.POPIS SLIKA

12.POPIS KÔDOVA