UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

PROJEKAT

iz predmeta Razvoj softvera za embeded sisteme

Internet prodavnica

Student:

Katarina Ninković

Profesor:

prof. dr Teodorović Predrag

Novi Sad, 2022. god.

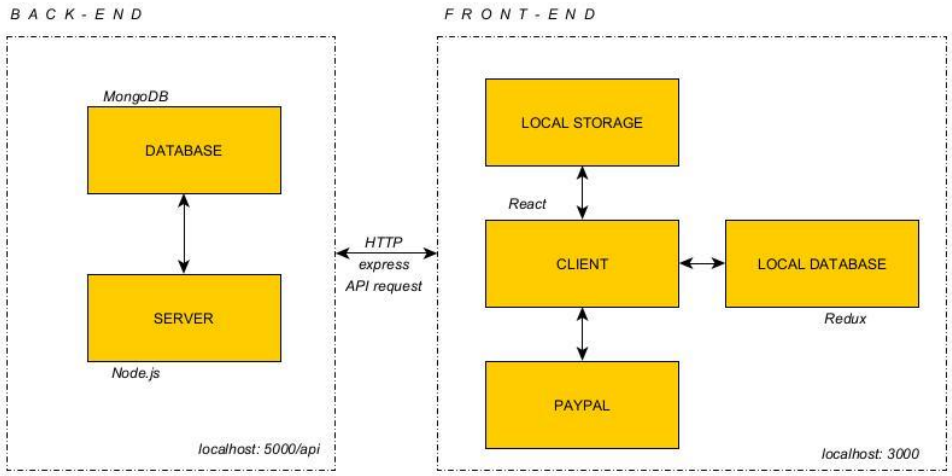
Sadržaj

1 Uvod 3
2 Back-end 4
2.1 Node.js 4
2.2 MongoDB 6
2.3 Express.js 8
2.4 Kontroleri 9
3 Front-end 10
3.1 Virtualni DOM 11
3.2 Elementi i komponente 12
3.3 Redux 14
4 Zaključak 16
5 Reference 17

1 Uvod

U današnje vreme proces *online* kupovine ili kupovine preko interneta postao je veoma prihvaćen i zastupljen u modernom društvu. Dinamika života današnjeg čoveka često ostavlja bez mogućnosti da prolazi kroz mnogobrojne radnje u potrazi za onim što mu je potrebno, što pored napornog obilaženja, uključuje i velik gubitak slobodnog vremena, te ovakav vid kupovine predstavlja olakšavajuću okolnost. Ova činjenica je bila podsticaj za pokretanje ovog projekta, realizacije internet prodavnice (eng. *Web Shop*), gde se proizvodi mogu kupovati iz udobnosti svog doma ili bilo koje druge lokacije na svetu. Korisnik je u mogućnosti da pregleda sve proizvode, odabere one odgovarajuće i „pošalje ih u korpu“, odakle će dalje moći da izvrši kupovinu.

Sistem koji čini ovaj projekat se može podeliti u dva osnovna bloka. Prvi blok predstavlja serversku stranu, kolokvijalno nazvanu *Back-end*, koja uključuje serversku aplikaciju baziranu na *Node.js* platformi i njoj specifičnom *javascript* programskom jeziku, i bazu podataka, koja je u ovom slučaju *MongoDB*. Drugi blok predstavlja klijentsku stranu iliti *Front-end* i sadrži *React.js* korisnički interfejs, koji je takođe baziran na *javascript* programskom jeziku, ali i na *HTML* i *CSS* jezicima za kreiranje i dizajniranje *web* stranica. Pored toga, na „frontu“ se koriste *Local Storage* za skladištenje podataka u pretraživaču, te lokalna baza podataka, čija se stanja kontrolišu pomoću *Redux*-a i *PayPal* kao *third party* platni sistem.



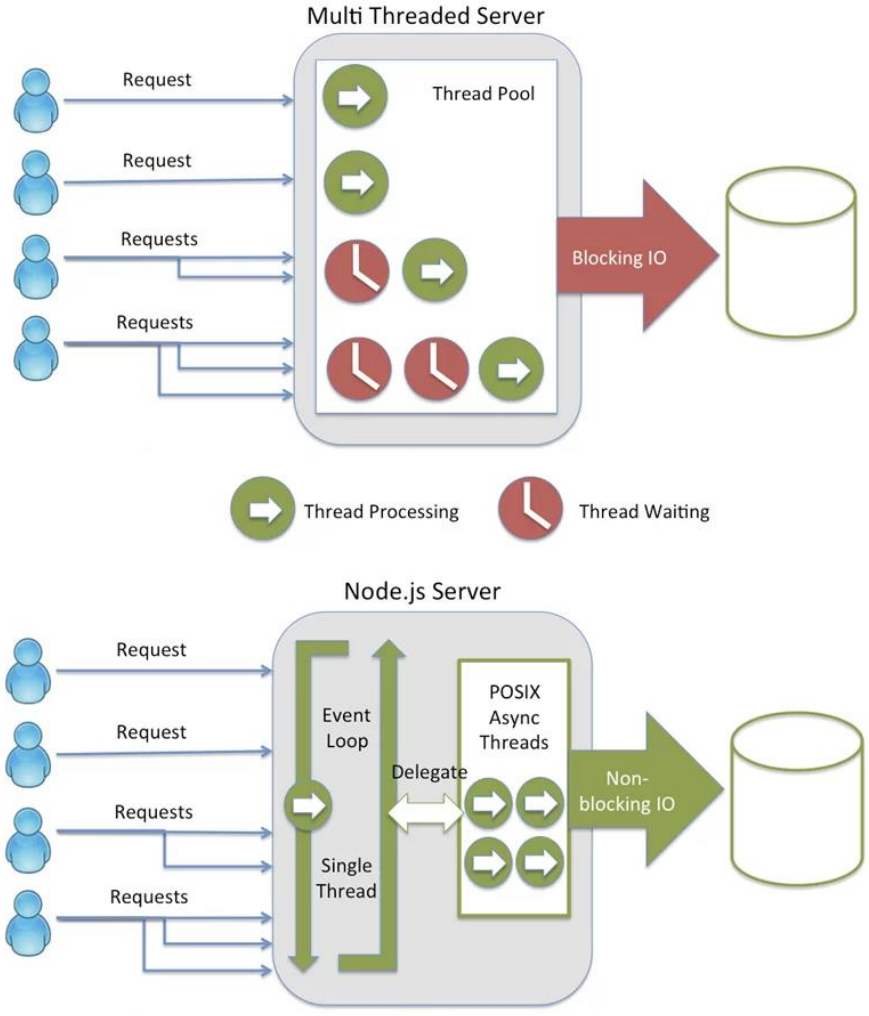
Slika 1. Blok šema celokupnog sistema

2 Back-end

Kao što je već spomenuto u uvodnom delu, u implementaciji sistema korišćen je *javascript* programski jezik, kako na *Front-end*, tako i na *Back-end* delu. *Javascript* je zavisan programski jezik, koji se ranije mogao koristiti isključivo na klijentskoj strani, tj. na pretraživaču (eng. *Browser*). Međutim, danas ovaj programksi jezik moguće je koristiti i na serverskoj strani zahvaljujući *Node.js* platformi.

2.1 Node.js

Node.js predstavlja *javascript* okruženje izvršavanja (eng. *Runtime Environment*), koji radi na *Google V8 Engine*-u, te obezbeđuje jednostavan, brz i asinhroni model programiranja za izradu *web* aplikacija. Obradu zahteva radi u jednom procesu, bez kreiranja nove niti (eng. *Thread*) za svaki zahtev. Na sledećoj ilustraciji (*slika 2.*) upoređeno je ponašanje standardnog višenitnog servera sa *Node.js* prilikom obrade novih zahteva. Može se uočiti da prvi pristup iziskuje čekanje za neke zahteve, za razliku od *Node*-a, čiji program ne čeka da se zahtev izvrši već nastavlja dalje na izvršenje sledeće linije koda.



Slika 2. Poređenje Multi-threading vs. Node.js servera

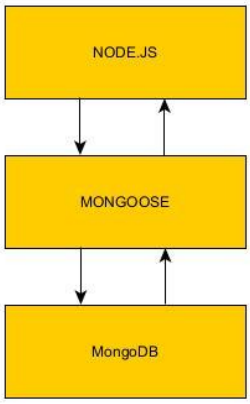
Deo koji dolazi sa instalacijom *Node*-a i koji igra bitnu ulogu jeste *Node Package Manager* (*npm*) koji upravlja *Node.js* paketima u aplikaciji. To je program komandne linije pomoću kojeg se vrši instalacija, ažuriranje i deinstalacija *Node.js* paketa. Takođe, predstavlja i mesto za skladištenje i pronalaženje paketa.

Node.js obezbeđuje i modularnost što podrazumeva njegovu nadogradnju i proširivanje njegovih mogućnosti. Instaliranje novih modula se obavlja pomoću *npm* koji nove pakete automatski dodaje u listu zavinsoti (eng. *Dependency*) u fajl *package* sa *JSON* ekstenzijom. Paketi koji su korišćeni su:

- *Mongoose* - za rad sa *MongoDB*
- *Dotenv* - za učitavanje promenljivih iz *env* datoteke u procesu
- *Express* - za rutiranje servera i obradu *HTTP* zahteva
- *BCrypts* - za enkriptovanje šifre
- *JSONWebToken* - za generisanje tokena

2.2 MongoDB

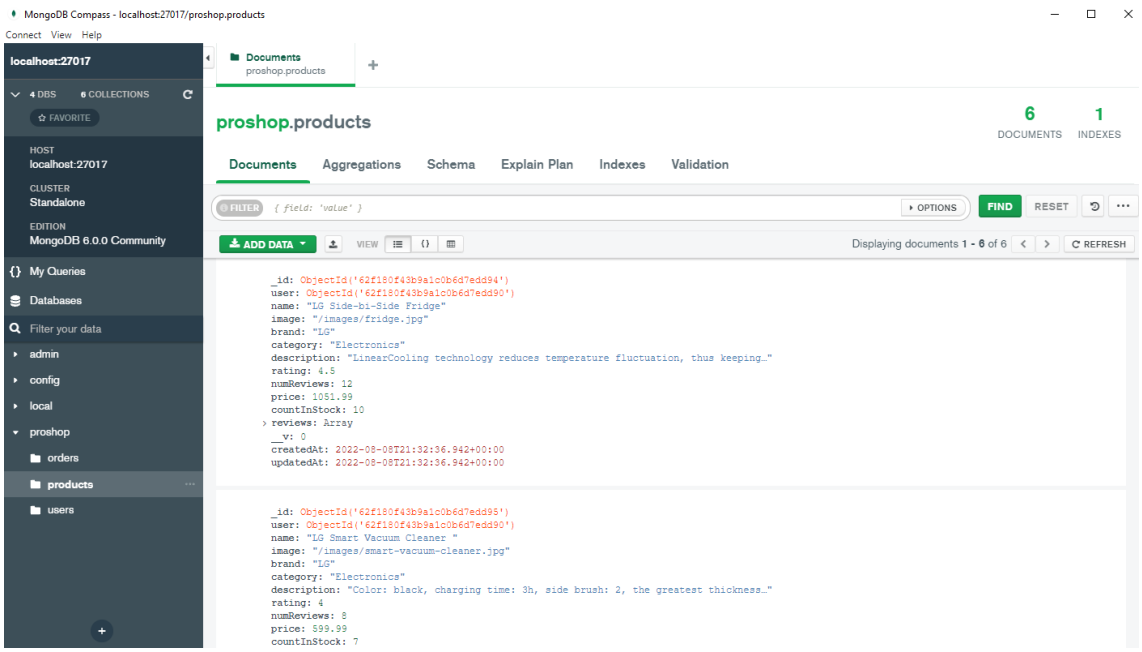
Mongoose je *npm* paket koji se koristi za modeliranje podataka objekata (eng. *Object Data Modeling - ODM*) za *MongoDB* i *Node.js*. Koristi se za prevođenje između objekata u kodu i predstavljanja istih u *MongoDB*.



Slika 3. Mongoose kao sprega između MongoDB i Node.js

Za bazu podataka korišćen je *MongoDB*. S obzirom da je ovo baza podata bez šeme (*NoSQL*) mogu se skladištiti *JSON* dokumenti u njoj, gde struktura ovih dokumenata može da varira, što predstavlja prednost u odnosu na *SQL* jer ubrzava razvoj aplikacije i smanjuje složenost primene.

Bazu je moguće pratiti i kontrolisati uz pomoć alata koji se zove *MongoDB Compass*. Naime, ovaj alat se povezuje sa bazom, na način na koji to radi i server, te je potrebno proslediti *IP* adresu, port i ime baze. Kada *MongoDB Compass* uspešno pristupi bazi, korisnik dobija grafički prikaz podataka, gde je moguće ručno modifikovati bazu, ako za tim postoji potreba. Alat je veoma koristan u fazi razvoja, a sigurno ima značaj i u produkciji.



Slika 4. Pristupanje bazi pomoću MongoDB Compass

Za potrebe ovog projekta definisana su tri tipa *MongoDB* modela. Reč je o uopštenim modelima koji su neophodni za rad internet prodavnice i to:

- *Product* je model koji je predviđen za plasiranje artikala koje internet prodavnica nudi. Sastoji se od elemenata kao što su: ime, opis, cena, količina (koliko artikala ima na stanju) i druga potrebna polja.
- *User* je model uz čiju pomoć se vrši evidencija korisnika. Sastoji se od klasičnih polja kao što su korisničko ime, *e-mail* adresa i lozinka.
- *Order* je model koji je zadužen za kolekciju realizovanih porudžbina. Sastoji se od liste naručenih proizvoda, adrese dostave, načina plaćanja, rezultata plaćanja, ukupne cene i drugo.

2.3 Express.js

Iako se *Node.js* koristi za kreiranje aplikacije na serverskoj strani, on ne uključuje rukovođenje zahtevima i *HTTP* metodama koje klijentska aplikacija šalje, pa se u tom slučaju koriste *Node*-ove ugrađene biblioteke poput *Express* paketa. *Express* predstavlja *Node.js* framework koji se koristi za izgradnju *API web* aplikacija.

Komuniciranje izmedju servera i baze podataka vrši se pomoću modela u *Express*-u. Gde serverska strana traži od modela da izvrši operacije sa bazom podataka. Metode koje su korišćene nad bazom su čitanje (*GET*), kreiranje (*POST*), i ažuriranje (*PUT*).

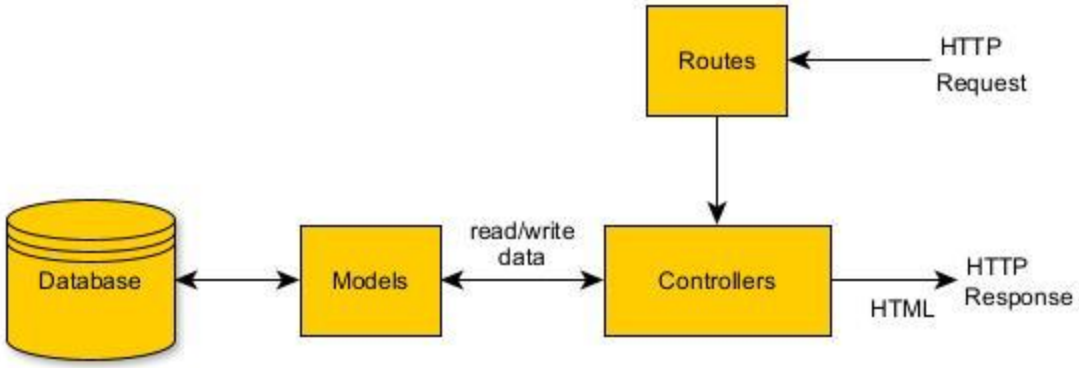
Nakon što server pošalje zahtev za čitanje podataka iz baze, potrebno je i dobiti te podatke kao odgovor. To obezbeđuju putanje ili rute (eng. *Routes*) koje su kreirane za određenu krajnju tačku (eng. *Endpoint*) koja je u fazi razvoja *http://localhost:5000/api*, dok bi u produkciji krajnja tačka bila naziv *web* stranice, što bi se realizovalo preko registracije i *host*-ovanja domena. Ukratko, rute služe za prosleđivanje zahteva odgovarajućim kontrolerima, koji dalje obavljaju svu potrebnu funkcionalnost.

Tabela 1. Postojeće rute i odgovarajući HTTP zahtevi

Rute	HTTP Metode	REQ	RES	Pristup	Opis
http://localhost:5000/api/products	GET		products	javni	Preuzimanje svih proizvoda
http://localhost:5000/api/products/:id	GET	id	product	javni	Preuzimanje jednog proizvoda
http://localhost:5000/api/users	POST	name, email, password	id, name, email, token (user)	javni	Registracija novog korisnika
http://localhost:5000/api/users/login	POST	email, password	id, name, email, token (user)	javni	Autorizacija korisnika i dobijanje tokena
http://localhost:5000/api/users/profile	GET	id	id, name, email (user)	privatni	Preuzimanje korisničkog profila
http://localhost:5000/api/users/profile	PUT	id, name, email, password	id, name, email, token (user)	privatni	Ažuriranje korisničkog profila
http://localhost:5000/api/orders	POST	orderItems, shippingAddress, paymentMethod, itemsPrice, taxPrice, shippingPrice, totalPrice	orderItems, user.id shippingAddress, paymentMethod, itemsPrice, taxPrice, shippingPrice, totalPrice (order)	privatni	Kreiranje nove porudžbe
http://localhost:5000/api/orders/:id	GET	id	order	privatni	Preizimanje porudžbe pomoću id
http://localhost:5000/api/orders/:id/pay	PUT	user id id, status, update_time, email from PayPal response		privatni	Ažuriranje porudžbe nakon plaćanja
http://localhost:5000/api/orders/myorders	GET	id	orders	privatni	Pregled svih porudžbi ako ste prijavljeni

2.4 Kontroleri

Kontroleri funkcionišu tako što preko modela šalju zahteve ka bazi, kreiraju određene upite u zavisnosti od toga za šta su im potrebni ti podaci i ukoliko ispunjavaju uslove onda kao odgovor dobijaju željene podatke.



Slika 5. Blok dijagram back-end strane

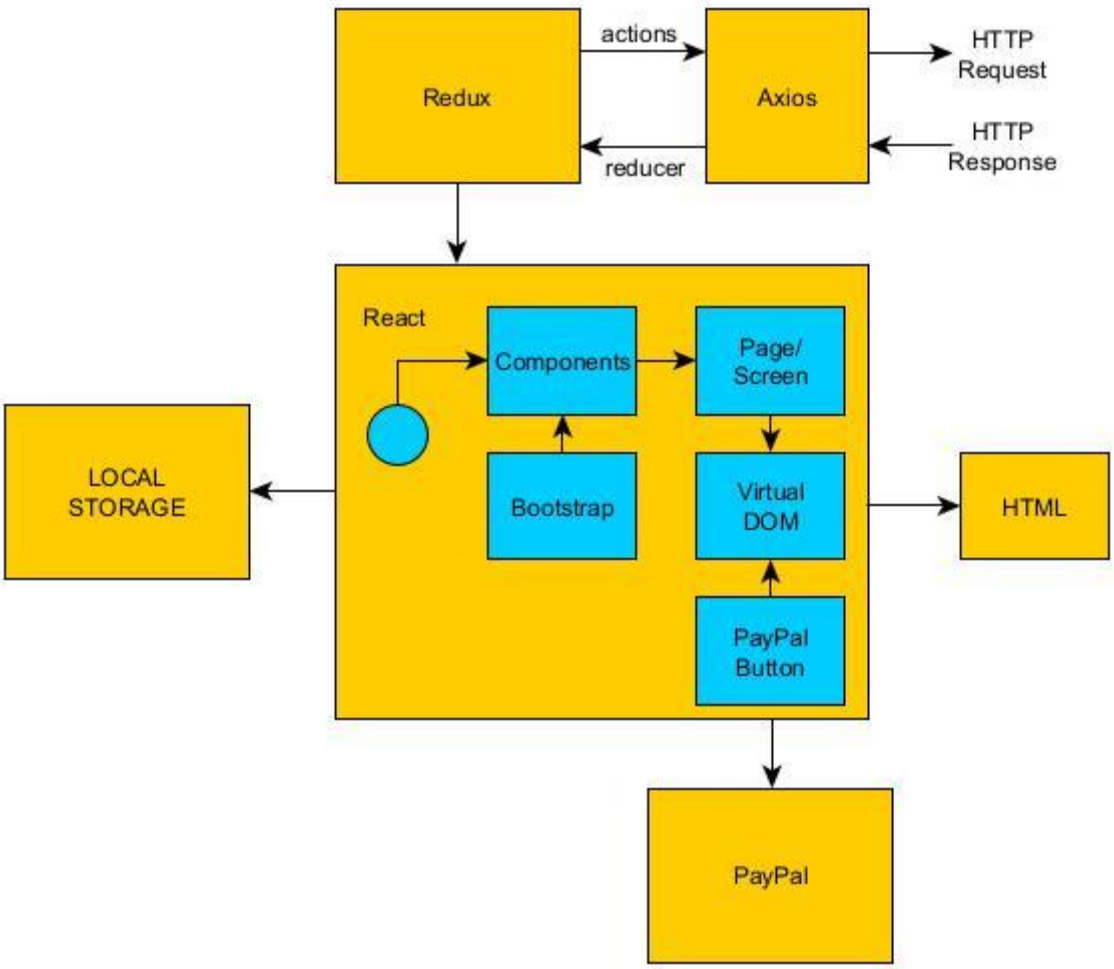
Ukoliko bi se uzeo primer prijave (eng. *Sign in*) korisnika na *web* stranicu, prvi korak bi bio unos podataka (*e-mail* adresa i lozinka) na klijentskoj strani, nakon čega sledi pakovanje unetih podataka u *HTTP* zahtev koji server preuzima i potom prosleđuje odgovarajućem kontroleru. Kontroler dalje za ulogu ima proveru da li je korisnik evidentiran u bazi tako što šalje zahtev ka bazi, tražeći da pročita podatke i izvrši određena ispitivanja kao što je u ovom slučaju postojanje i poklapanje unete *e-mail* adrese i šifre. Ukoliko korisnik postoji kao odgovor od baze dobijamo podatke vezane za tog korisnika.

Neki korisnički podaci su zaštićeni, kao na primer profil korisnika. Kako bismo mogli pristupiti, odnosno videti podatke tog korisnika potrebno je da imamo *JWT* (*JSON Web Token*). Kreiranje *JWT* tokena vrši se uz pomoć *JSONWebToken npm* paketa. Prilikom generisanja koristi se funkcija koja za prvi argument očekuje nekakav jedinstveni *ID*. Poželjno je da to bude podatak koji nije od velike važnosti, kao što bi, na primer, bio broj računa, koji treba da ostane tajna. Kao drugi argument stavlja se tajna sekvenca, odnosno kreira se promenljiva koja je poznata vlasniku (kreatoru) servera i stavljena je u zaštićenom *env* fajlu. Za učitavanje te promenljive iz *env* datoteke koristi se *Dotenv* paket. Treći argument je opcion i predstavlja vreme isteka tokena.

3 Front-end

Za razvoj korisničkog interfejsa korišćena je *React* biblioteka sa *ES6 javascript* verzijom, koji nam omogućava jednostavno i lako pisanje koda. Kao i na *back-end* strani, za kreiranje projekta i instalaciju paketa treće strane koristi se *npm* menadžer za instalaciju paketa. Paketi koji su uključeni na *front-end*-u su:

- *React* - za korišćenje svih *React* funkcionalnosti
- *Axios* - za slanje *HTTP* zahteva
- *PayPalButton* - za ubacivanje dugmeta za prelazak na *PayPal* prozor
- *Bootstrap* - za uvlačenje stilizovanih komponenti
- *Redux* - za kontrolu lokalne baze podataka



Slika 6. Blok dijagram front-end strane

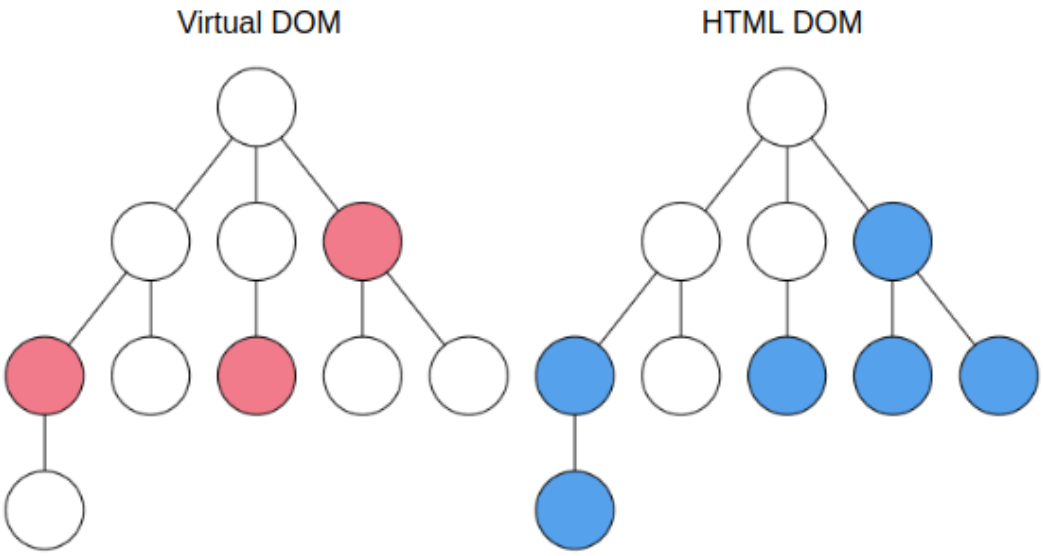
Prednosti *React*-a su olakšano kreiranje dinamički renderovanih *web* aplikacija i mogućnost kreiranja virtualnog *DOM* (*Document Object Model*) zahvaljujući kojem su poboljšane performanse i brže kreiranje aplikacije.

3.1 Virtualni DOM

React kreira *javascript* objekte za svaki *HTML* element pomoću kojih se vrši kontrola nad *DOM* modelom. Objekti mogu biti elementi ili komponente (složeni elementi) i svaki od njih može da uključuje neke druge objekte čime obrazuju strukturu "stabla" gde je svaki čvor jedan objekat. Na taj način se kreira virtualni *DOM* kao identična kopija realnog modela, čija je uloga upoređivanje i ažuriranje istog, koji se koristi za renderovanje stranice.

Prilikom promene stanja nekog objekta virtualni *DOM* ažurira samo taj objekat dok ostale ne dira, za razliku od realnog modela koji bi ažurirao celo stablo čime predstavlja mnogo sporiju manipulaciju objektima. Dakle, renderovanje objekata u *HTML DOM* modelu obezbeđeno je pomoću virtualnog.

Na sledećoj ilustraciji (*Slika 6.*) prikazan je proces interakcije između virtualnog i *HTML* modela. Virtualni *DOM*, kao kopija realnog, vrlo jednostavno može da odredi lokaciju objekta koji je potrebno ažurirati. Nakon promene stanja nekog objekta virtualni model se ažurira, zatim upoređuje svoje prethodno stanje kako bi video gde se promena desila i nakon toga ažurira samo te objekte u *HTML DOM*-u i sve ostale objekte koji su mu podređeni.



Slika 7. Prikaz virtualnog DOM-a prilikom promene objekta i prikaz renderovanja istih u realnom DOM-u

3.2 Elementi i komponente

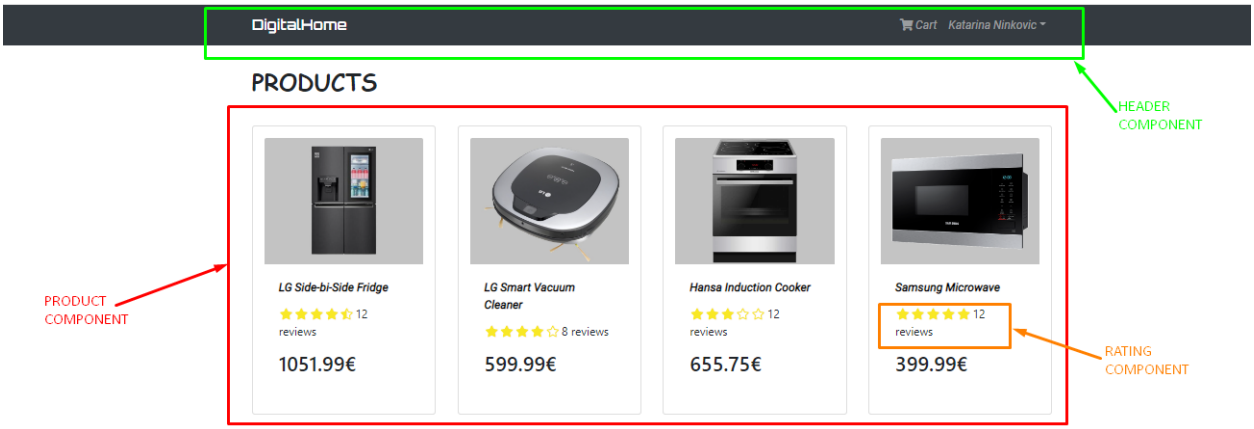
React elementi se koriste za opisivanje izgleda *HTML* elemenata, koji se prikazuju na pretraživaču. Za opis tih elemenata koristi se *javascript* sintaksno proširenje (eng. *Javascript Syntax Extension - JSX*) koji nam obezbeđuje pisanje *HTML* strukture i *javascript* kod u istoj datoteci. Kako bismo mogli da upravljamo tim elementima koristimo *React* komponente.

Slika 8. JSX kao kombinacija javascript i HTML

React deli korisnički interfejs na brojne komponente koje predstavljaju gradivne blokove, čime je otklanjanje grešaka olakšano. Mogućnosti koje komponente obezbeđuju su:

- Jedna komponenta se može koristiti u više oblasti aplikacije, čime ubrzava proces razvoja
- Komponente mogu sadržati i druge komponente
- Komponenta može da primi svojstva (eng. *Properties*) iliti *props* koju prosleđuje njen roditelj.

U okviru ovog projekta deljiva osobina *React* komponenti je u velikoj upotrebi zbog postojanja više stranica koje moraju uključiti iste *HTML* elemente i komponente. Time je, očuvan i identičan stil na svim stranicama sajta, ali i bitna funkcionalnost. Pre svega, ovo se odnosni na navigacioni meni iliti *navbar* koji je uzet sa jednog od popularnih *Bootstrap* šablona.



Slika 9. Prikaz komponenata na početnoj stranici

Tabela 2.Postojeće front-end rute i odgovarajući prikazi

Rute	Glavna komponenta	Opis
http://localhost:3000	HomeScreen	Prikaz svih artikala
http://localhost:3000/register	RegisterScreen	Registracija korisnika
http://localhost:3000/login	LoginScreen	Prijavljivanje korisnika
http://localhost:3000/profile	ProfileScreen	Korisnički prozor
http://localhost:3000/product/:id	ProductScreen	Prozor specifičnog artikla
http://localhost:3000/cart/:id?	CartScreen	Prikaz korpe
http://localhost:3000/shipping	ShippingScreen	Unos ličnih podataka za dostavu
http://localhost:3000/payment	PaymentScreen	Odabir načina plaćanja
http://localhost:3000/placeorder	PlaceOrderScreen	Prikaz realizovanih porudžbina
http://localhost:3000/order/:id	OrderScreen	Plaćanje porudžbine

Tabele 2. i *3.* prikazuju pobrojane sve rute na *front-end* strani, te njihove glavne komponente (jedinstvene *DOM* modele po stranicama), kao i komponente koje dele. Može se uočiti da neke rute na kraju imaju deo formata "*:x*", što znači da te rute nisu fiksne.

Tabela 3. Postojeće komponente i njihova relacija sa stranicama (rutama)

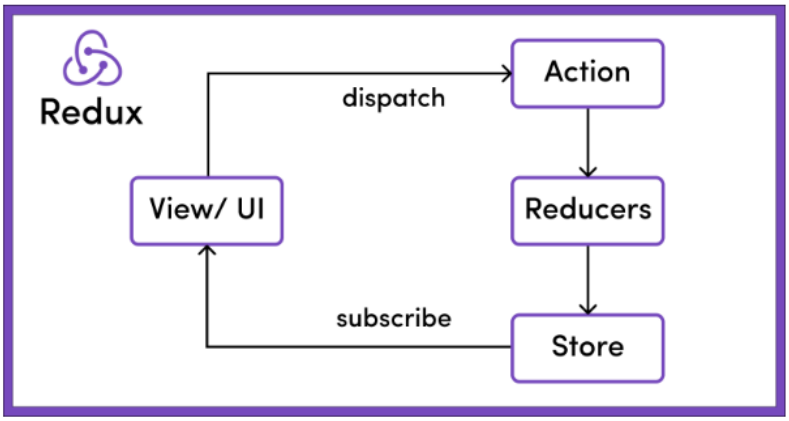
Komponenta	Rute	Opis
Header	http://localhost:3000 http://localhost:3000/register http://localhost:3000/login http://localhost:3000/profile http://localhost:3000/product/:id http://localhost:3000/cart/:id? http://localhost:3000/shipping http://localhost:3000/payment http://localhost:3000/placeorder http://localhost:3000/order/:id	Zaglavlje svake stranice, uključuje navigacioni meni
Footer		Podnožje svake stranice, uključuje *copyright* oznaku
Product	http://localhost:3000	Pojedinačni artikal
Message	http://localhost:3000 http://localhost:3000/register http://localhost:3000/login http://localhost:3000/profile http://localhost:3000/product/:id http://localhost:3000/cart/:id? http://localhost:3000/placeorder http://localhost:3000/order/:id	Obaveštenja (uspeh, greška ili upozorenje)
Loader	http://localhost:3000 http://localhost:3000/register http://localhost:3000/login http://localhost:3000/profile http://localhost:3000/product/:id http://localhost:3000/order/:id	Učitavanja ili čekanja odgovora
Rating	http://localhost:3000/product/:id	Prosečna ocena artikla
FormContainer	http://localhost:3000/register http://localhost:3000/login http://localhost:3000/shipping http://localhost:3000/payment	Forma za unos podataka
ChechoutSteps	http://localhost:3000/shipping http://localhost:3000/payment http://localhost:3000/placeorder	Koraci za realizaciju kupovine

3.3 Redux

Potreba za *Redux*-om može se objasniti kroz primer akcije dodavanja artikala u korpu. Svaki put kada bi korisnik dodao proizvod u korpu aplikacija mora obraditi tu radnju i staviti taj proizvod u objekat korpu. Ista priča je i sa brisanjem neželjenih proizvoda iz korpe. Ovo je moguće održavati, međutim aplikacija može vremenom postati veća i usled toga otežati održavanje stanja. Kako bi se izbegla takva situacija koristi se *Redux*.

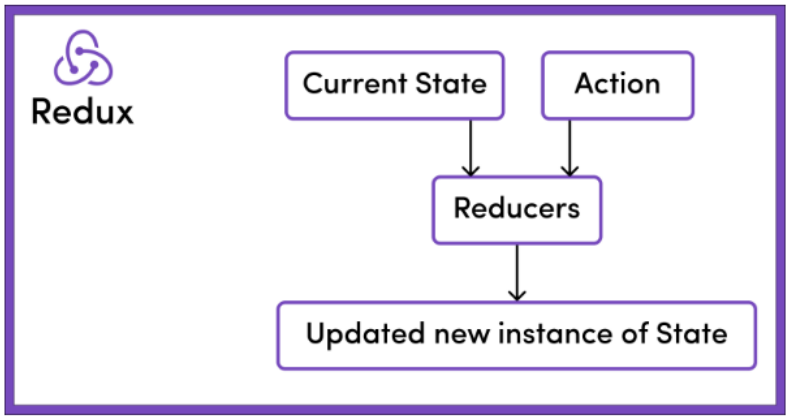
Redux je biblioteka koja uspešno rukuje višestrukim stanjima iz više komponenti i može se koristiti sa poznatim *framework*-om kao što je *React*, *Angular* ili *Vue*.

Stanja u *Redux*-u predstavljaju *javascript* objekte koji se mogu samo čitati (eng. *Read-only*). Kako bi se moglo uticati na promenu stanja i njegovu manipulaciji koriste se akcije i reduktori (eng. *Reducer*), gde se uz pomoć akcija opisuju željene izmene a pomoću reduktora te ismene izvršavaju.



Slika 10. Relacije između Redux komponenti

Zadatak reduktora je da prihvati akciju i trenutno stanje aplikacije, pa potom vrati novo ažurirano stanje. Reduktorske metode se ne pozivaju direktno nego uz pomoć specijalne funkcije *dispatch* iz *Redux* biblioteke. *Dispatch* potom aktivira sve reduktore, ali se izvršavaju samo oni koji se poklapaju sa prosleđenom akcijom. Potrebno je imati na umu da je stanje *read-only*, jer reduktori prvo prave kopiju celog trenutnog stanja, izvrše izmene i vraćaju novu instancu stanja.



Slika 11. Tok ažuriranja stanja na Redux-u

Tabela 4. Lista reduktora i modela stanja

Slice	Reduktori	Model stanja	Opis
Cart	cartAddItem, cartRemoveItem, cartSaveShippingAddress, cartSavePaymentMethod	cartItems, shippingAddress, paymentMethod	Rukovođenje korpom
Order	orderCreateRequest, orderCreateSuccess, orderCreateFail	order, loading, success, error	Rukovođenje porudžbinom
Product	productListRequest, productListSuccess, productListFail	products, loading, error	Uvid u listu artikala
User	userLoginRequest, userLoginSuccess, userLoginFail, userLogout	userLoginRequest, userLoginSuccess, userLoginFail, userLogout	Rukovođenje korisničkim podacima

4 Zaključak

Može se zaključiti da sistem čiji je cilj bio primena klijent-server komunikacije sa praktičnom primenom uspešno realizovan. Projekat je uključio razvijanje modernog *web* sajta sa poslednjim tehnologijama za razvijanje istog. Obezbeđeno je paralelno posluživanje više klijenata istovremeno, kao i dodavanje različitih artikala.

Neka od eventualnih poboljšanja bila bi svakako unapređivanje korisničkog interfejsa, kategorizacija proizvoda, pretraga proizvoda, kao i *SEO* (*search-engine* optimizacija), mogućnost ostavljanja recenzija na proizvode, dodavanje admin korisnika sa *CRUD* (*Create, Read, Update, Delete*) stranicom za artikle, podrška za *Android* i *iOS* kroz mobilne aplikacije itd.

5 Reference

[1] https://www.popwebdesign.net/popart_blog/2015/06/sta-je-node-js/ - internet članak o *Node.js*, stranica *Popwebdesign*, pristupljeno: septembar 2022.

[2] https://matfuvit.github.io/UVIT/vezbe/knjiga/Poglavlja/NodeJS/ - *Node.js*, stranica *Matfuvit*, pristupljeno: septembar 2022.

[3] https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/ – *Mongoose and MongoDB*, stranica *Freecodecamp*, pristupljeno: septembar 2022.

[4] https://medium.com/@mzarkovicm/lagani-uvod-u-react-a4d07034a604 - internet članak o *React*-u, stranica *Medium*, pristupljeno: septembar 2022.

[5] https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/ - internet članak o *Redux*-u, stranica *Freecodecamp*, pristupljeno: septembar 2022.