Izrada web trgovine upotrebom MEVN stack-a

Filipović, Filip

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Filipović

IZRADA WEB TRGOVINE UPOTREBOM MEVN STACK-A

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Filipović

Matični broj: 45922/17–R

Studij: Informacijsko i programsko inženjerstvo

IZRADA WEB TRGOVINE UPOTREBOM MEVN STACK-A

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, rujan 2022.

Filip Filipović

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema diplomskog rada je razvoj web aplikacije odnosno web trgovine koristeći tehnologije MEVN stack-a. U uvodu rada stoji opis web aplikacija, nakon čega slijedi detaljniji opis spomenutih tehnologija te kratka usporedba sa konkurentima. Naglasak će biti na odabranim tehnologijama, SEO analizi kroz aplikacije na jednoj stranici i razlici između izvršavanja aplikacije na klijentskoj strani i strani servera. Praktični dio sastoji se od potpuno funkcionalne web aplikacije koja ima mogućnost kreiranja korisničkih računa, spremanja košarice, izrade narudžbi i plaćanje istih. Osim običnog korisnika, aplikacija ima i administratorski modul u kojem administrator može na jednostavan način upravljati proizvodima i korisnicima te filtrirati i pretraživati narudžbe. Svaki aspekt aplikacije poslužitelja i njegovih mogućnosti i putanja detaljno je opisan u ovome dijelu. Isto vrijedi i za klijentsku aplikaciju gdje je opisana implementacija modernih funkcionalnosti svake web stranice. Na samom kraju rada slijedi zaključak.

Ključne riječi: REST, SEO, SSR, MEVN stack, web trgovina

Sadržaj

1. Uvod

Mobiteli, tableti i računala sastavni su dio dana gotovo svake suvremene osobe. Količina dostupnih informacija, načini komunikacije koji omogućuju razgovore u realnom vremenu između osoba diljem cijelog svijeta, zabava i sve drugo što se može pronaći na internetu uvelike je olakšalo život ljudima. Stoga, većina vremena koja se provodi na malim i velikim ekranima zapravo označava surfanje webom kroz internet preglednike.

Baš iz navedenih razloga, web stranice i aplikacije koje se posjećuju bi trebale biti što brže, prilagođene za sve vrste ekrana s obzirom da veći dio prometa dolazi sa mobilnih uređaja i oku privlačne odnosno u skladu sa današnjim standardima kako bi zainteresirale i zadržale korisnika koji ih posjećuje. Iako za većinu web aplikacija postoje i mobilne aplikacije, internetski preglednici unaprijed su instalirani na većini uređaja te pristup web aplikacijama ne zahtijeva nikakva dodatna preuzimanja i instalacije.

U današnje vrijeme, popularni okviri za JavaScript jezik omogućili su aplikacije na jednoj stranici što omogućuje aplikaciji da komunicira s korisnikom te na dinamičan način prepisuje trenutnu web stranicu sa novim podacima koje vrati poslužitelj za razliku od zadane metode web preglednika koji učitava cijele nove stranice iste aplikacije. Cilj takvih aplikacija je bolje korisničko iskustvo jer se može prikazati indikator učitavanja novih podataka što omogućuje brže prijelaze između stranica iste aplikacije. U takvim aplikacijama se nikad ne događa osvježavanje stranice na razini preglednika, već se sav sadržaj dinamički briše i učitava se novi.

Primjer takve aplikacije je upravo i praktični dio ovog rada, web trgovina koja kada se jednom učita, dinamički mijenja sav sadržaj te se ponovno učitavanje cijele aplikacije događa jedino kada korisnik to zatraži.

2. MEVN stack i korišteni alati

U modernom web programiranju ME(V)N *stack* dosta je čest pojam koji se odnosi na izradu aplikacija. S obzirom da većina aplikacija zahtijeva nekakvu vrstu podataka, iste je potrebno spremiti u bazu podataka koja, radi sigurnosti, komunicira sa serverom. Nakon što server dohvati podatke, tada ih može vratiti klijentu ili korisniku aplikacije. Upravo se ta sva logika može napraviti preko MEVN stack-a, gdje svako slovo označava akronim za pojedinu tehnologiju, a to su redom:

- MongoDB,
- Express,
- Vue.js – koji često može biti zamijenjen sa konkurentskim tehnologijama tipa Angular ili React,
- Node.js.

Osim MEVN stack-a koji predstavlja tehnologije potrebne za izradu aplikacija, korišteni su i drugi alati koji recimo pojednostavljuju rad sa bazom, služe za verzioniranje programskog koda i slično. Iz tih razloga, radi jednostavnijeg i boljeg programerskog iskustva, korišteni alati jesu:

- Visual Studio Code – uređivač teksta odnosno programskog koda sa većim brojem integracija poput terminala, ugrađenog alata za rad sa Git-om i drugima,
- MongoDBCompass – alat za spajanje na bazu i sa jednostavnim radom s podacima na bazi, službeni alat MongoDB-a,
- Insomnia – API platforma za razvojne programere koja služi za dizajniranje, izgradnju, testiranje i ponavljanje API poziva, u slučaju izrade praktičnog dijela, alat se koristi za testiranje API putanja na serveru,
- Google Chrome/Firefox – internetski preglednici korišteni za testiranje aplikacije odnosno front-end dijela aplikacije. S obzirom da nisu sve funkcionalnosti JavaScript-a podržane kroz sve preglednike i njihove verzije, uvijek je poželjno testirati aplikaciju na više različitih preglednika i sustava kako bi se dobila bolja povratna informacija.

Kako su sve tehnologije MEVN *stack*-a (osim baze podataka) bazirane na JavaScript-u, cijela klijentska i poslužiteljska strana napisane su u TypeScript-u, strogom sintaktičkom nadskupu JavaScripta koji omogućuje tipiziranje varijabli radi lakšeg čitanja i održavanja programskog koda. Dodatni plus je već ugrađeni *IntelliSense* u Visual Studio Code koji zahvaljujući TypeScript-u može prepoznati objekte te automatski ponuditi svojstva koja određeni objekt ima prilikom pisanja programskog koda.

2.1. MongoDB

MongoDB trenutno je najpoznatiji program baze podataka orijentiran na dokumente, što ga čini klasificiranim kao NoSQL programom baze podataka. Za spremanje podataka koristi dokumente slične JSON obliku sa shemama koje definiraju izgled kolekcija. [1] Pogodan je za web programiranje jer može spremati objekte koji izgledaju jednako kao i objekti koji se pripreme na klijentskoj strani.

Prvo izdanje veže se trinaest godina unazad što MongoDB čini relativno mladom tehnologijom za baze podataka. Razvijen je od strane MongoDB Inc., a napisan je u programskim jezicima C++, JavaScript i Python. Postoji u tri izdanja, MongoDB Community Server, MongoDB Enterprise Server te MongoDB Atlas koji je izrazito popularan zbog toga što označava bazu u oblaku. Više o samom izgledu baze, kolekcija i dokumenata biti će objašnjeno u opisu implementacije.

2.2. Express

Express predstavlja back-end okvir dizajniran za razvijanje web aplikacija i API-ja za Node.js. Nazvan je de facto standardnim serverskim okvirem za Node.js. Autor okvira opisao ga je kao poslužiteljem nadahnutim Sinatrom (besplatnom bibliotekom za web aplikacije napisanom u Ruby programskom jeziku) što znači da je relativno minimalan s mnogo dostupnih dodataka. Kreirao ga je T.J. Holowaychuk ne tako davne 2010. godine, a do sada je prema podacima preuzimanja sa npmjs.com stranice, najpopularniji okvir za razvoj API-ja. [2] Jedna od glavnih značajki Expressa je to što je jednostavan za pokrenuti osnovni program odnosno osnovnu API točku iz koje proizlaze cijeli programi i moduli.

```
import express from "express";

const app = express();

app.get("/", (req, res) => {
  res.send("Hello world.");
});

app.listen(5000, () => {
  console.log("Server running on port 5000.");
});
```

Primjer 1. Osnovni primjer Express servera

2.3. Vue.js

Vue.js (ili samo Vue) je okvir za JavaScript programski jezik sa glavnom zadaćom jednostavne izrade korisničkih sučelja i web aplikacija na jednoj stranici. Kreirao ga je Evan You prije osam godina, u svrhu korištenja u internim projektima jer mu se osobno nije svidio niti jedan postojeći okvir u to vrijeme. Evan je prije razvoja okvira radio za Google i u nekoliko projekata koristio Google-ov Angular, iz kojeg je uzeo stvari koje su mu se svidjele i napravio okvir po svojim željama i potrebama da bude što minimalniji. Ubrzo nakon prve stabilne verzije, ljudi su počeli primjećivati njegov rad te je u roku nekoliko mjeseci uspio formirati tim kojemu je cilj napraviti najbolji okvir za JavaScript. [3]

Vue se nadovezuje na standardni HTML, CSS i JavaScript i pruža deklarativni model programiranja temeljen na komponentama koji pomaže da se učinkovitije razviju korisnička sučelja, bila ona jednostavna ili složena. Za razliku od konkurencije, Vue se može vrlo lagano priključiti na bilo koju HTML datoteku odnosno njen element.

```
// main.js
import { createApp } from "vue"

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount("#app")

// index.html
<div id="app">
  <button @click="count++">
    Brojač: {{ count }}
  </button>
</div>
```

Primjer 2. Gumb sa prikazom broja klikova koristeći Vue

Gornji primjer rezultira gumbom koji u sebi sadrži tekst „Brojač: x“ gdje x označava broj klikova na gumb. Sa gornjeg primjera uočljive su dvije temeljene značajke Vue-a, deklarativno iscrtavanje i reaktivnost. Deklarativno iscrtavanje govori da Vue proširuje standardni HTML sintaksom predloška koja omogućuje deklarativno opisivanje HTML izraza na temelju stanja JavaScript-a. Reaktivnost se primjećuje na tome da Vue automatski prati promjene stanja JavaScript-a i učinkovito ažurira DOM kada se promjene dogode. I jedno i drugo načelo vidljivo je na tekstu gumba „Brojač: {{ *count* }}“ gdje *count* predstavlja vrijednost koja se nalazi u JavaScript datoteci, a zbog reaktivnosti je nakon svakog klika vidljiva promijenjena vrijednost varijable. [4]

```
// main.js
const button = document.getElementById("button");
let count = 0;

const increment = () => {
    count++;
    button.innerText = `Brojač: ${count}`;
}

// index.html
<div>
  <button id="button" onclick="increment()">
    Brojač: 0
  </button>
</div>
```

Primjer 3. Gumb sa prikazom broja klikova koristeći JavaScript

Na gornjem primjeru nalazi se identičan rezultat kao i sa početnog primjera koristeći Vue, samo ovaj puta se koristi klasični JavaScript bez ikakvog okvira. Iako je HTML dio gotovo identičan, main.js datoteka nešto se razlikuje. U ovom slučaju, cijeli tekst gumba mijenja se koristeći JavaScript, čak i dio u kojem se nalazi statični dio teksta koji se nikad ne mijenja, „Brojač: “. Iako izgledaju jednako jednostavno, kada bi uzeli više stvari u obzir poput automatskog gledanja na promjene vrijednosti varijable *count*, došli bi do zaključka da koristeći okvir poput Vue-a imamo jednostavniji, čitljiviji i održiviji kod nego koristeći klasični JavaScript.

2.3.1. Nuxt

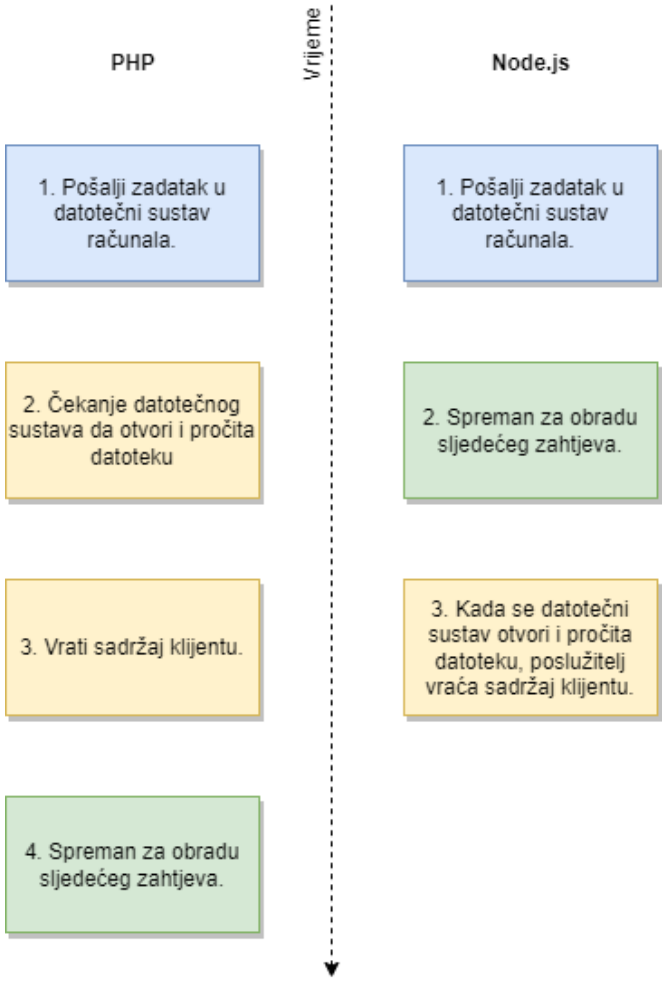
S obzirom da je Vue-u jedna od glavnih značajki izrada aplikacija na jednoj stranici koja se u učitava na klijentskoj strani, dolazimo do problema sa SEO-om. Laički rečeno, Google SEO bazira se na pregledavanju izvora stranice te čitanja <a> elemenata te se na temelju *href* vrijednosti kreće po aplikaciji i pregledava podatke. Kada se otvori izvor stranice koja je napravljena u Vue-u, dobiva se prazna stranica sa jednim elementom jer se ostatak stranice učitava na klijentskoj strani.

Tu u igru dolazi Nuxt, razvojni okvir temeljen na Vue-u koji omogućuje učitavanje stranice na serveru tako da klijent dobije sve potrebne podatke i HTML prilikom inicijalnog učitavanja stranice na njegovoj strani. Iz tog razloga, na izvoru stranice vidljivi su podaci koji su izloženi iscrtavanju na serverskoj strani (eng. *Server side rendering*). Osim SEO-a, postoje i druge značajke koje automatski dolaze sa instalacijom Nuxt-a poput optimiziranog učitavanja slika, jednostavnijeg usmjeravanja na aplikaciji, *plugin*-ova, globalnog stanja i drugih. [5]

2.4. Node.js

Node.js je back-end okruženje otvorenog koda koje radi na JavaScript V8 Engine-u i izvršava JavaScript kod izvan web preglednika odnosno omogućuje programiranje na strani poslužitelja, a dizajniran je za izgradnju skalabilnih aplikacija. Baziran je na JavaScript-u, skriptnom programskom jeziku koji omogućuje implementaciju složenih funkcionalnosti na web aplikacijama sa izvršavanjem u web pregledniku na strani korisnika te ga bez Node.js-a nije moguće izvršiti izvan web preglednika. Razvio ga je Ryan Dahl 2009. godine, te je označio revoluciju JavaScripta jer je bio jednostavniji nego njegov prethodnik, Netscape-ov LiveWire Pro Web.

Node.js koristi asinkrono programiranje što omogućuje ne blokirajuće izvršavanje zahtjeva koje korisnik šalje. Uzmimo za primjer zadatak otvaranja datoteke na poslužitelju i vraćanje sadržaja klijentu u PHP-u i Node.js-u. Zahvaljujući asinkronom programiranju, Node.js nakon primanja zahtijeva ne mora čekati rezultat zadatka već može krenuti na obradu sljedećeg zahtjeva s obzirom da se radi o programiranju na jednoj dretvi. [6]



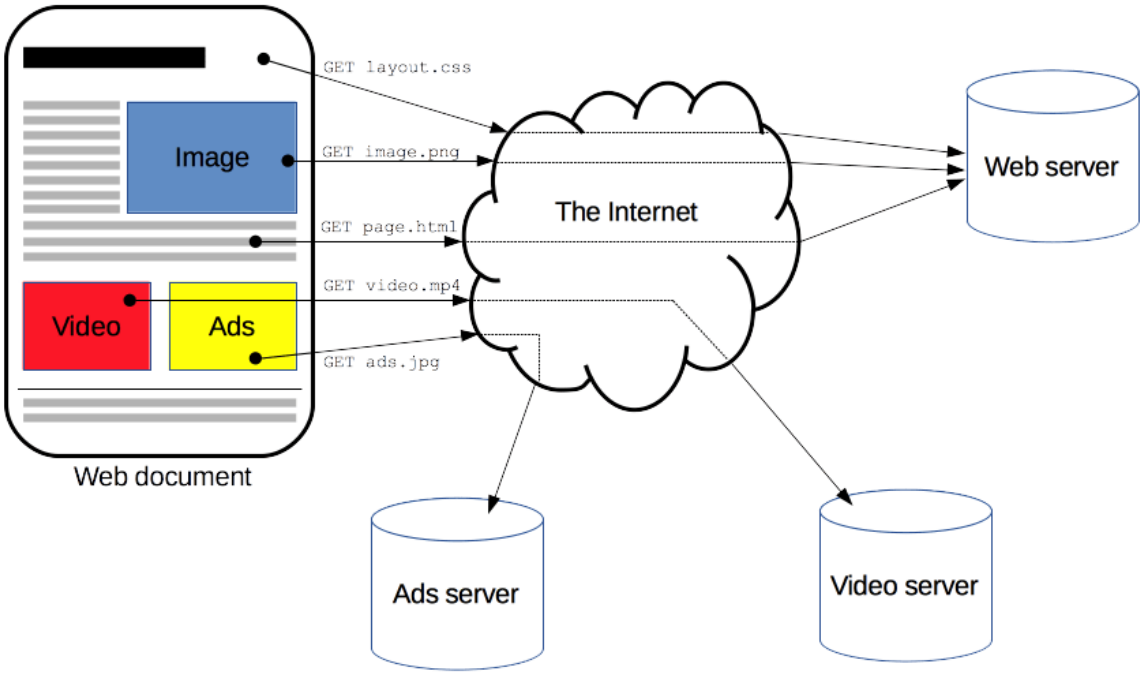
Slika 1. Primjer obrade korisničkog zahtjeva u PHP-u i Node.js-u

3. HTTP

HTTP (eng. *Hypertext Transfer Protocol*) glavna je i najčešća metoda prijenosa informacija na web-u. Dizajniran ranih 1990-ih, HTTP je proširiv protokol koji se s vremenom razvijao. To je protokol aplikacijskog sloja koji se šalje preko TCP-a ili preko TLS kriptirane TCP veze, iako bi se teoretski mogao koristiti bilo koji pouzdani transportni protokol. Zbog svoje proširivosti, ne koristi se samo za dohvaćanje HTML datoteka, već i za slike i videozapise ili za objavu sadržaja na poslužiteljima, poput rezultata HTML obrazaca. HTTP se također koristi za dohvaćanje dijelova dokumenata za ažuriranje web stranica i aplikacija na zahtjev. [7]

Drugim riječima, HTTP korisniku omogućuje prikaz web stranice te služi za komunikaciju sa serverom u obliku pitanje – odgovor. Proširuje web aplikaciju do te granice da korisnik može unijeti podatke, poslati serveru ispunjene podatke te ih spremiti i u drugom trenutku ili na drugom uređaju dohvatiti ranije unesene podatke.

HTTP/1.1 verzija, koja je i dalje aktualna, sadrži poruke koje su čitljive ljudima, gdje novija verzija, HTTP/2 sadrži poruke ugrađene u binarnu strukturu što omogućuje optimizacije poput kompresije zaglavlja i slično. [8]



Slika 2. Ilustracija rada HTTP-a [7]

Postoje dvije vrste HTTP poruka, a to su zahtjevi i odgovori od kojih svaka ima svoj format. Svaki HTTP zahtjev sastoji se od sljedećih elemenata:

- HTTP metoda koja definira koju operaciju klijent želi izvesti. Korištene metode jesu sljedeće:
  - GET – korištena isključivo za dohvat resursa, zahtjevi koji su bazirani na GET metodi ne rade nikakve izmjene na bazi podataka,
  - POST – koristi se kod slanja entiteta poslužitelju, korištena za dodavanje novih resursa u bazu ili autentifikaciju korisnika,
  - PUT – služi za ažuriranje postojećih resursa,
  - DELETE – metoda koja briše navedeni resurs,
- putanja na koju se zahtjev šalje. Može sadržavati ID resursa i/ili podatke o filtriranju rezultata poput tekuće stranice za paginaciju i izraza za pretraživanje i slično,
- verzija HTTP-a,
- neobavezna zaglavlja koja prenose dodatne informacije za poslužitelje poput korisničkog agenta i ostalih, u praktičnom dijelu primarno služe za slanje autorizacijskog tokena na server,
- tijelo zahtjeva, koristi se za slanje podataka o zapisima koji se dodaju ili ažuriraju u bazi.

HTTP odgovor relativno je sličnog formata kao i zahtjev, a sadrži sljedeće elemente:

- verziju HTTP-a,
- statusni kod koji ukazuje uspješnost zahtjeva, odnosno neuspješnost zahtjeva i objašnjenje, grupirani u pet kategorija odnosno klasa:
  - informativni odgovori (100 – 199),
  - uspješni odgovori (200 – 299),
  - poruke preusmjeravanja (300 – 399),
  - odgovori pogreške na strani klijenta (400 – 499),
  - odgovori pogreške na strani poslužitelja (500 – 599)
- statusna poruka koja je usko povezana sa statusnim kodom,
- zaglavlja koja sadrže dodatne informacije o odgovoru,
- tijelo odgovora koje sadrži novokreirani resurs, opis greške ili bilo što drugo.

Iako se tehnički cijela aplikacija može napraviti koristeći isključivo POST metode koje uvijek vraćaju statusni kod 200, korištenjem unaprijed definiranih metoda i statusnih kodova dobijemo kvalitetnije i robusnije aplikacije i mogućnosti kojima recimo GET metoda ima pristup, dok POST metoda nema.

3.1. Aplikacijsko programsko sučelje (API)

API (eng. *Aplication Programming Interface*) je softverski posrednik koji omogućuje da dvije aplikacije međusobno komuniciraju i razmjenjuju podatke. Svaki put kada korisnik koristi aplikaciju, klijentska strana kontaktira poslužitelja (koristeći HTTP zahtjev) koji vrati potrebne podatke za prikaz. Nekim jednostavnijim jezikom rečeno, poslužitelj za praktični dio koji je napisan u Node.js-u predstavlja API. Kada bi željena aplikacija zahtijevala podatke o vremenskoj prognozi, umjesto izrade vlastite prognoze, što bi bilo skupo i vremenski zahtjevno, programer može jednostavno pozvati jedan takav servis tipa *OpenWeatherMap*, poslati podatke o koordinatama za koje želi prognozu i dobiti natrag objekt sa podacima o temperaturi, vlažnosti zraka i svemu popratnom.

U suštini, upotreba API-ja omogućava programerima korištenje rada drugih programera štedeći trud i vrijeme koje je potrebno da se napravi neki složeni program (npr. dobivanje prognoze) koristeći iste standarde.

3.1.1. REST

REST ili *REpresentational State Transfer* je arhitektonski stil za pružanje standarda između računalnih sustava na webu čime se olakšava međusobna komunikacija sustava. Sustavi usklađeni REST-om, koji se često nazivaju i *RESTful* sustavi, karakterizirani su time što su bez stanja te odvajaju brige između klijenta i poslužitelja. [9] U prijevodu, klijent nakon što pošalje zahtjev čeka odgovor od poslužitelja koji može bez znanja klijenta kontaktirati API za prognozu, dohvatiti podatke o prognozi te podatke koje on ni ne posjeduje – vratiti klijentu.

Implementacija klijenta i poslužitelja u REST arhitektonskom stilu mogu se obaviti neovisno jedna o drugoj. To znači da se programski kod na strani poslužitelja može promijeniti u bilo kojem trenutku bez utjecaja na rad klijenta i obratno. Sve dok svaka strana zna koji format poruka treba poslati drugoj, one mogu biti modularne i odvojene. Osim toga, odvajanje omogućuje svakoj komponenti mogućnost samostalnog razvoja. [10] Slična implementacija opisanoj bila bi implementacija mikroservisne arhitekture koja je unazad nekoliko godina sastavni dio svake *enterprise* aplikacije. REST tehnologija općenito ima prednost u odnosu na druge slične tehnologije jer koristi relativno nisku razinu propusnosti što je čini prikladnom za učinkovitije korištenje interneta.

U praktičnom dijelu koristit ćemo tri glavne putanje koje će sa svojim podrutama biti dovoljne za obradu svih funkcionalnosti aplikacije.

4. Usporedba tehnologija

Količina jezika, okvira i biblioteka koje danas postoje u svijetu web programiranja vjerojatno je nadmašila sva očekivanja programera ranih 2000-ih. Danas se, koristeći gotovo svaki jezik može napraviti poslužiteljska strana REST servisa kao i klijentska strana web aplikacije. Bez obzira na količinu svih jezika i načina, uvijek se izdvaja njih nekoliko koji su postali današnji standard web programiranja. Stoga, u sljedećim odlomcima slijedi kratka usporedba korištenih tehnologija sa njihovim konkurentima.

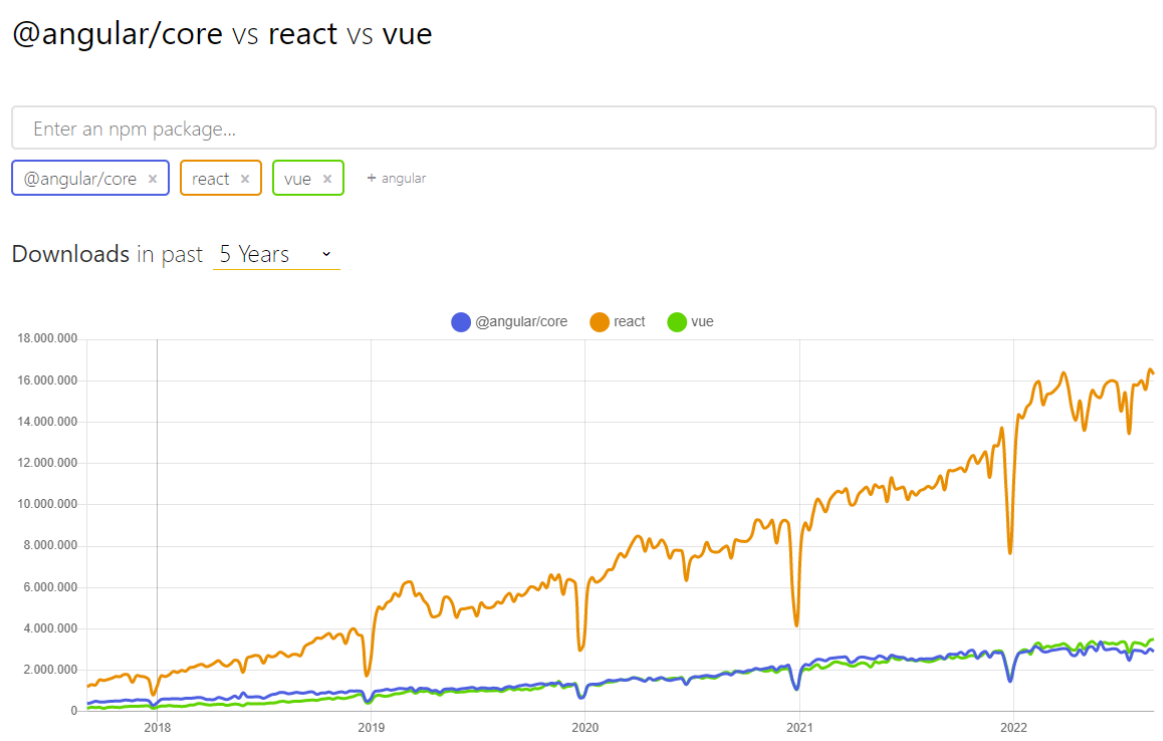
4.1. Font-end tehnologije

Pojavom novih okvira i biblioteka za JavaScript, usporedba modernih JavaScript tehnologija postala je dosta česta tema internetskih blogova. Tako dakle razlikujemo tri glavne takve tehnologije: Angular, React i Vue. Svaka od navedenih tehnologija ima svoje prednosti i mane što ih čini dobrim konkurentima jedna drugoj.

Angular je okvir za JavaScript otvorenog koda temeljen na TypeScript-u kojeg vodi tim za Angular u Google-u. Radi se o potpuno prerađenom prethodniku imena AngularJS. [11] Kao i ostale dvije tehnologije, bazira se na izgradnji komponenata koje čine web aplikaciju. Omogućuje MVC arhitekturu što ga čini pogodnim za programere koji dolaze sa znanjem Jave i sličnih jezika. Za razliku od druge dvije tehnologije, Angular ne zahtijeva dodatke za samostalan i normalan rad, već u sebi sadrži sve potrebno za rad aplikacije, što ga može činiti nefleksibilnim. U te sve dodatke dolazi do izražaja i učitavanje stranica na razini servera za koje React i Vue trebaju zasebne biblioteke što može uzročiti dodatno učenje tehnologija. Također, za Angular se smatra da je najteži za naučiti, što nekim programerima može dati razlog za razmatranje drugih tehnologija.

S druge strane, React je najpopularnija tehnologija (od tri navedene) koju je razvila Meta (bivši Facebook). Za razliku od Angulara, React se dobrim dijelom oslanja na dodatke koje je napravila zajednica programera koji koriste React. Samim time, navigiranje po aplikaciji, a da se pritom ne krši koncept aplikacija na jednoj stranici, nije moguć bez dodatka – što možda nekima predstavlja manu, a nekima prednost. Za pisanje React-a koristi se posebna sintaksa naziva JSX, te se smatra najjednostavnijim od sve tri tehnologije. [11] Različit je od ostale dvije po tome što je React samo biblioteka, a ne cijeli okvir, što ga ne čini manje sposobnim za izradu aplikacija koje moraju biti visoko interaktivne. Od verzije 16.8 React koristi funkcijske komponente i *hooks*-e, što je okarakteriziralo samu tehnologiju. [12]

Vue, koji je već opisan u prvom dijelu, nije razvijen od strane mega korporacije već od bivšeg zaposlenika Google-a kojem se koncept Angulara nije u potpunosti svidio. Dobra prednost Vue-a je što sadrži najdetaljniju i najintuitivniju dokumentaciju od sve tri tehnologije, no veličina zajednice i broj dodataka nije opsežan kao kod Angular-a ili React-a. Vue osim što je okvir, može se ubaciti u bilo koju postojeću stranicu i njen HTML, što je prikazano na samom početku rada. Prvenstveno se bavi MVVM arhitekturom, a kao i Angular, koristi klasični HTML te direktive koje proširuju svaki element.



Slika 3. Graf popularnosti sve tri tehnologije prema https://npmtrends.com/

Zaključak bi bio da ne postoji jedna tehnologija koja je bolja od druge dvije, već je sve stvar preferencije i potreba projekta koji se izrađuje.

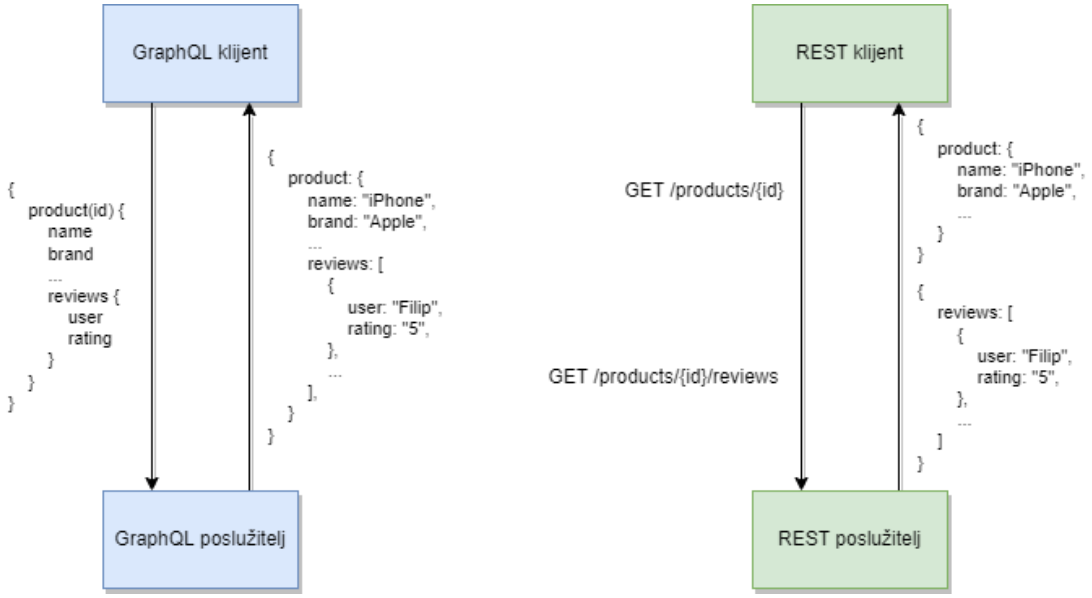
4.2. Back-end tehnologije

Usporedba back-end tehnologija u ovome slučaju vršit će se kao usporedba različitih arhitektura API-ja koji se mogu izraditi koristeći JavaScript odnosno Node.js. Opće je poznato da je JavaScript na strani poslužitelja jedan od sporijih jezika pa usporedba s drugim jezicima nema puno smisla osim toga što se Node.js smatra jednostavnijim od ostatka s obzirom da se radi o JavaScript-u odnosno TypeScript-u.

Jedna takva usporedba bila bi između trenutno dva najpopularnija načina implementacije API-ja, a to su REST i GraphQL. Jedna od glavnih razlika između njih je

arhitektura na temelju koje rade, gdje REST predstavlja arhitekturu vođenu poslužiteljem, dok se GraphQL temelji na arhitekturi vođenom klijentom. Recimo da dolazimo na stranicu proizvoda neke web trgovine i na njoj se nalaze informacije o proizvodu i recenzije istog tog proizvoda. U teoriji, REST servis zahtijevao bi poziv dvije rute gdje bi prva dohvaćala informacije o proizvodu, a druga listu recenzija vezanih za taj proizvod. Prvi problem kod takvog pristupa je dohvaćanje podataka s dvije rute umjesto s jedne, a drugi potencijalno pretjerano dohvaćanje podataka. Zamislimo da poslužitelj na putanji dohvaćanja proizvoda vrati podatke koji se ne prikazuju na stranici poput ID-a proizvoda, datuma kreiranja i posljednjeg ažuriranja, potrebna je veća propusnost s obzirom da se radi o većoj količini podataka što na kraju krajeva može utjecati na brzinu izvršavanja zahtijeva.

GraphQL za isti primjer zahtijeva podatke na malo drugačiji način. Kako je on temeljen na arhitekturi vođenom klijentom, potrebni podaci definiraju se na klijentskoj strani u obliku tzv. grafova. To nam omogućuje da sve potrebne podatke dohvatimo samo jednim zahtjevom što rješava problem pretjeranog dohvaćanja podataka. Uz to, ukoliko mijenjamo strukturu stranice na klijentskoj strani, dovoljno je definirati dodatno polje za dohvat (pod pretpostavkom da se model sastoji od tog podatka) pa promjene na strani poslužitelja nisu potrebne.



Slika 4. Prikaz rada GraphQL-a i REST-a

No, nije ni GraphQL bez mane. Jedan od problema s kojim se susreće je izvješće grešaka. Naime, svaki poziv na GraphQL server vraća statusni kod 200 bez obzira na njegovu uspješnost. Isto tako, GraphQL nema mogućnost izrade putanje za prijenos datoteka bez dodataka niti ima ugrađeni sustav predmemoriranja. Isto tako, ukoliko netko s treće strane želi koristiti naš API, tada njihova aplikacija mora imati implementiran GraphQL kako bi uopće mogala stvarati pozive. No unatoč prednostima i nedostatcima, na kraju se kao i kod tehnologija na klijentu, sve svodi na stvar preferencije i potreba projekta.

5. Praktični dio

Praktični dio sastoji se od web aplikacije koja simulira web trgovinu sa svim funkcionalnostima modernih web trgovina, od registracije i prijave korisnika, spremanja košarice na korisnički račun (što omogućava pregledavanje košarice na više uređaja) do administratorskog modula koji služi za unos i izmjenu proizvoda i korisnika te pregled narudžbi. Osim osnovnih funkcionalnosti, implementirano je i plaćanje pomoću Stripe klijenta. Korišten je novi mehanizam Stripe-ovog plaćanja po uzoru na *low code* platformu. Radi se o unaprijed izgrađenoj stranici za plaćanje koja je podignuta na Stripe-ovom serveru, a može se na jednostavan način koristiti za jednokratne kupnje ili pretplate, garantira jednostavnost i ono najbitnije, sigurnost. Kao što je i ranije spomenuto, Vue sam po sebi ne nudi dobar SEO, stoga je cijeli front-end napravljen u Nuxt-u što omogućuje učitavanje aplikacije na strani poslužitelja gdje se učitavaju proizvodi te se cijeli HTML šalje na klijenta. Iz istog razloga, ako se otvori izvor početne stranice ili stranice određenog proizvoda, vidljivi su svi podaci kao i sva moguća navigacija po stranici.

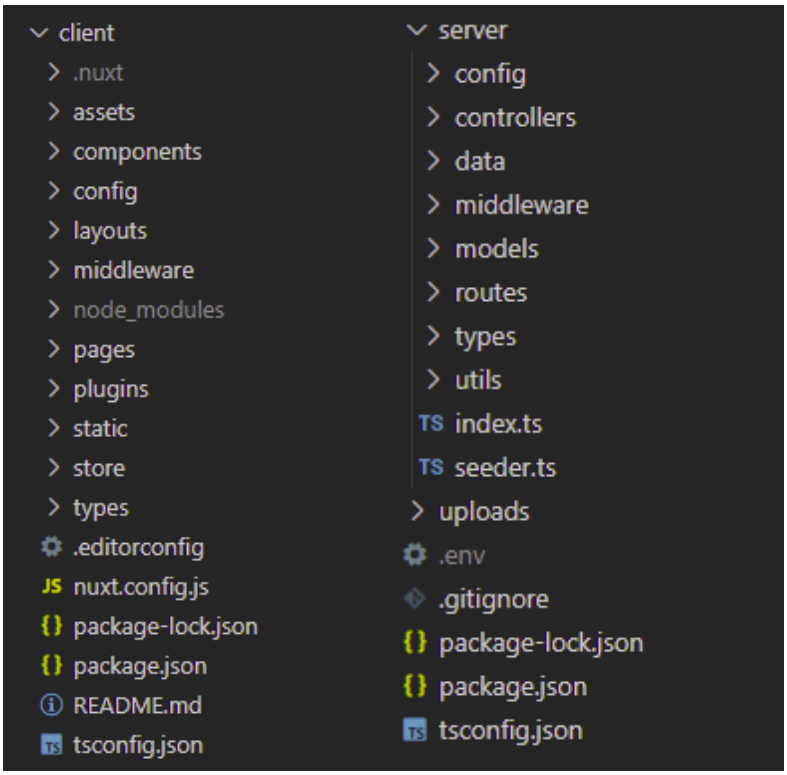
Cijeli tijek aplikacije napravljen je da bude što jednostavniji za korisnika, odnosno aplikacija je dizajnirana da bude minimalna i lako dostupna svim dobnim skupinama korisnika. Registracija zahtijeva minimalne podatke koji su bitni za kasnije korake tipa dostavu. Nakon uspješne registracije, korisnik je automatski prijavljen te može početi koristiti svaki aspekt aplikacije koji mu je ponuđen. U ovom trenutku korisnik može uređivati profil, pregledavati sve svoje narudžbe, dodavati proizvode u košaricu (ili ih brisati iz iste), pokrenuti proces unosa narudžbe gdje je dužan ispuniti podatke o adresi dostave, odabrati način plaćanja i finalno pregledati narudžbu te ukoliko je sve u redu, potvrditi narudžbu i poći do ekrana detalja narudžbe. Inicijalno narudžba nije niti plaćena, niti označena kao dostavljenom, stoga je korisnik dužan za početak pokrenuti proces plaćanja. Nakon plaćanja, bilo uspješnog ili neuspješnog, korisnik se vraća na istu stranicu sa povratnom informacijom o plaćanju te u ovom trenutku, ukoliko je plaćanje uspješno, slijedi čekanje i proces logistike koji nije obrađen u ovom radu. Isto tako, korisnik može ostaviti po jednu recenziju na proizvod kako bi drugi korisnici dobili bolji dojam i iskustvo drugih korisnika za određeni proizvod.

Za razliku od običnog korisnika, administrator ima dodatni modul upravljanja podacima. Radi se o tri zasebne stranice koje su izgledom međusobno inspirirane, a svaka prikazuje tablični prikaz drugih podataka. Tako imamo stranicu za upravljanje narudžbama, proizvodima i korisnicima. Sva tri resursa možemo obrisati preko aplikacije, dok proizvode i korisnike možemo dodavati i uređivati – stavljati novu cijenu, promijeniti opis ili bilo koje drugo svojstvo proizvoda, dodavati te oduzimati administratorska prava drugim korisnicima.

5.1. Priprema projekta i projektna struktura

Cijela izrada aplikacije kreće inicijalnim postavljanjem projekta, razdvajanja strane poslužitelja i klijenta. Za potrebe rada, projekt je napravljen kao *monorepo*, jedan repozitorij koji u sebi sadrži i back-end i front-end. Iako je dobra praksa odvajati kod na više repozitorija i modula, radi jednostavnosti je sve stavljeno u jedan repozitorij.

Stoga, sukladno gornjem tekstu, početak izgleda sa izradom dva glavna foldera po imenima *client* i *server*. U *client* folderu nalazi se cijeli kod koji je isključivo vezan za klijenta, odnosno Nuxt aplikacija. Zahvaljujući *npm*-u, upravitelju paketa za JavaScript, Nuxt aplikacija, kao i konkurentske tehnologije, u projekt se ubacuju sa svega jednom linijom koda. U ovom slučaju, radi se o liniji koja glasi *npx create-nuxt-app <ime-projekta>*, gdje ime projekta može biti točka što znači da će *npm* instalirati aplikaciju u direktoriju u kojem se trenutno nalazi. Nuxt inicijalizacija je jednostavan proces kroz nekoliko koraka gdje se bira hoće li projekt automatski generirati datoteku za konfiguraciju TypeScript-a, hoće li inicijalno instalirati dodatne pakete tipa Axios koji služi za jednostavno slanje zahtjeva prema API-ju i slično. S obzirom da je *npm* upravitelj paketa, svaki paket koji se instalira skida se lokalno na računalo ili server. Paketi i njihove verzije mogu se upravljati kroz datoteku imena *package.json*, a podaci svakog paketa nalaze se u folderu *node_modules* koji se sastoji od nekoliko tisuća datoteka i mapa potrebnim za normalan rad aplikacije.



Slika 5. Struktura cijele aplikacije

Sa gornje slike vidljiva je struktura cijele aplikacije koja se sastoji od direktorija klijenta i servera, a uključuje sljedeće mape i datoteke (FE označava direktorij na klijentu, BE označava poslužitelja, a ako stavka nema oznaku, obuhvaća obje strane):

- *assets/static* direktoriji (FE) – sadrže statičke elemente aplikacije, tipa globalnu *.css* datoteku, *favicon* i slike,
- *components/pages* direktorij (FE) – sadrži sve komponente i stranice korištene u aplikaciji. Radi se o komponentama koje se koriste na više mjesta u aplikaciji time prateći dizajn komponenta za višekratnu upotrebu,
- *config* direktorij – direktorij sa datotekama potrebnim za lokalizaciju FE aplikacije i datoteci za spajanje na bazu te Stripe klijentom,
- *layouts* direktorij (FE) – sadrži dvije datoteke, stranicu 404 te *default.vue* koja služi za izradu rasporeda elementa kroz sve stranice (tipa zaglavlje i podnožje),
- *middleware* direktorij – direktorij sa *middleware* datotekama koje služe kako ne bismo pisali na svaku stranicu ili putanju kod ako korisnik nema pristup istoj (autorizacijski *middleware*, admin *middleware*, *middleware* za greške i sl.),
- *plugins* direktorij (FE) – specifičan direktorij za Nuxt, omogućuje ubrizgivanje dodataka u *this* varijablu,
- *store* direktorij (FE) – direktorij sa svim datotekama za globalna stanja, sadrži stanja o korisniku, košarici i slično,
- *types* direktorij – pomoćni direktorij sa tipiziranim objektima (TypeScript), proširuje Express-ov *Request* objekt sa *user* objektom,
- *controllers/routes* direktorij (BE) – *routes* sadrži popis svih ruta na našoj REST aplikaciji od kojih svaka ruta konzumira određenu metodu iz *controllers* foldera,
- *data* direktorij i *seeder.ts* datoteka (BE) – datoteka koja služi kao skripta za popunjavanje baze sa osnovnim podacima, koristi podatke iz *data* direktorija,
- *models* direktorij (BE) – sadrži modele za bazu,
- *utils* direktorij (BE) – pomoćni direktorij sa datotekom za generiranje tokena,
- *uploads direktorij* (BE) – spremište slika proizvoda koji se prenose kroz FE,
- *index.ts* (BE) – inicijalna datoteka koja se pokreće prilikom pokretanja servera, objedinjuje sve rute, *middleware*-e i pakete u jednu aplikaciju,
- *.env* (BE) – sadrži povjerljive informacije, tipa podatke za spajanje na bazu, ključ za Stripe klijenta i druge, radi se o varijablama okoline za projekt,
- *nuxt.config.js* (FE) – konfiguracijska datoteka za Nuxt aplikaciju,
- *tsconfig.json*, *package.json* i *package-lock.json* – konfiguracijska datoteka za TypeScript i datoteke koje stvara *npm*, sadrži podatke o projektu, skripte i popis paketa sa korištenim verzijama.

5.2. Baza podataka

Kao što je spomenuto ranije, tehnologija koja je korištena za bazu podataka jest MongoDB. MongoDB za razliku od SQL baza podataka koristi tzv. kolekcije umjesto tablica te ne zahtijeva strogo definiranje modela na razini baze. U suštini, možemo poslati bilo koji objekt na bazu na što će MongoDB automatski kreirati novu kolekciju ako ona ne postoji i ubaciti podatke. Na strani poslužitelja korišten je paket *mongoose*, najpoznatiji paket za modeliranje MongoDB objekata za Node.js.

Ako razmišljamo na način web trgovine, potrebni su nam proizvodi bez kojih web trgovina ne bi bila trgovina. Osim proizvoda, potrebno je napraviti model narudžbi kako bi korisnici mogli naručivati proizvode iz trgovine, a kako bi korisnici mogli pregledavati prošle narudžbe, potrebno je napraviti i model korisnika na kojeg možemo spremati košaricu kako bi korisnik na bilo kojem uređaju i u bilo koje vrijeme mogao nastaviti kupovinu koju je započeo. Na proizvod se veže model recenzije s obzirom da je jedna recenzija vezana za jedan proizvod koja može, a i ne mora postojati. Zaključak je izrada tri modela: proizvodi, korisnici te narudžbe. Svako polje u definiranju modela ima obavezan atribut koji označava tip polja, te dosta često i atribut koji označava obaveznost polja. Osim ova dva najčešća polja, u modelima se može pronaći i polje zadane vrijednosti ako se ne pošalje niti jedna druga. Polja koja označavaju referencu na drugi model koriste *ref* atribut polja koji ukazuje na model na koji se veže polje.

S obzirom da su modeli sa svim poljima poprilično veliki, a korišten je TypeScript za tipiziranje shema, na sljedećoj stranici vidljivi su svi tipovi podataka koji se spremaju u bazu. Svako sučelje odgovara jednoj shemi u *mongoose*-u, a određena polja poput ocjene i broja recenzija proizvoda imaju zadanu vrijednost 0, što i ima logike. Tako dakle svaki korisnik ima osnovne podatke o kontaktu, broj bodova lojalnosti, zastavicu koja upućuje je li korisnik administrator ili ne te podatke o košarici. Podaci o košarici uključuju referencu na proizvod koja je uvijek aktualna trenutnim podacima te količinu tog proizvoda. Proizvod s druge strane sadrži osnovne informacije koje bi svaki proizvod trebao sadržavati i listu recenzija koje se vežu za određenog korisnika gdje jedan korisnik može isključivo napisati jednu recenziju po proizvodu. Tako ograničenje nije na razini baze, već na razini poslužitelja. Zadnji model tiče se narudžaba koje predstavljaju najveći objekt. Svaka narudžba striktno je povezana sa korisnikom, a osim poveznice sa korisnikom sadrži i podatke o proizvodima u vremenu kreiranja narudžbe, podatke o adresi dostave, sve cjenovne razrede koji ulaze u takvu narudžbu (cijena bez poreza, porez, dostava te ukupna cijena narudžbe) te podatke o plaćanju koji nisu dostupni klijentu već se koriste za provjere na strani poslužitelja. Naravno, datum plaćanja i dostave prikazuju se na klijentskoj strani s obzirom da se na temelju njih ne može naštetiti sigurnosti.

```
interface OrderItem {
  name: string;
  quantity: number;
  image: string;
  price: number;
}

interface Order {
  user: Schema.Types.ObjectId;
  orderItems: OrderItem[];
  shippingAddress: {
    address: string;
    city: string;
    postalCode: string;
    country: string;
  };
  paymentMethod: string;
  paymentSession?: string;
  paymentResult?: {
    id?: string;
    status?: string;
    customer_name?: string;
    email_address?: string;
  };
  cartPrice: number;
  taxPrice: number;
  shippingPrice: number;
  totalPrice: number;
  isPaid: boolean;
  paidAt?: number;
  isDelivered: boolean;
  deliveredAt?: number;
}
```

```
interface CartProduct {
  product: Schema.Types.ObjectId;
  quantity: number;
}

interface User {
  name: string;
  email: string;
  password: string;
  isAdmin: boolean;
  loyaltyPoints: number;
  cart: CartProduct[];
  checkPassword: (password: string)
=> Promise<boolean>;
}
```

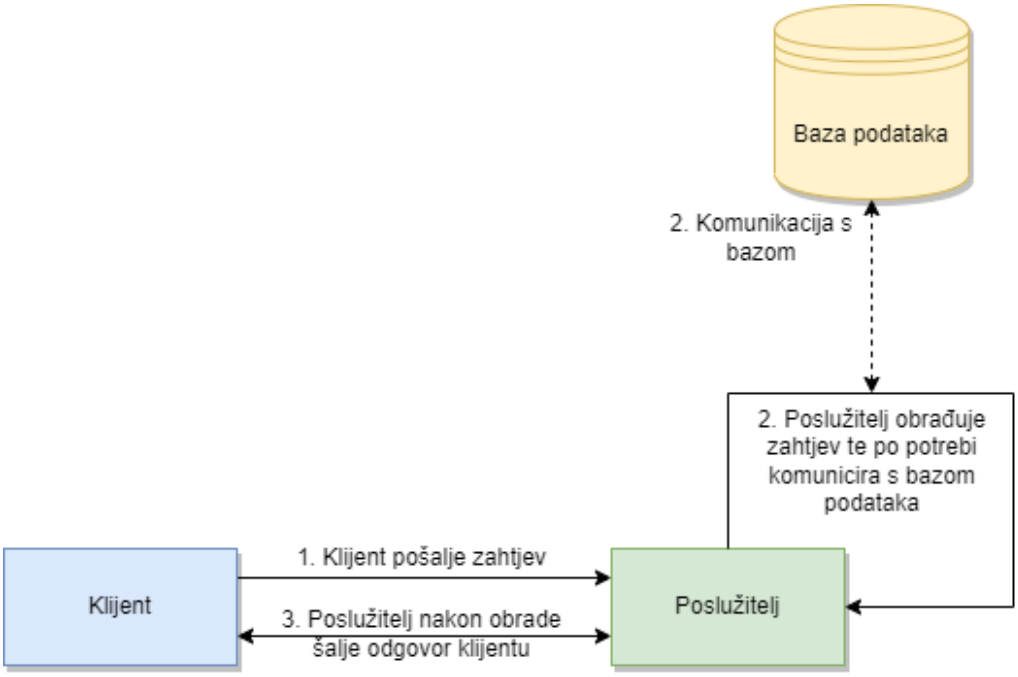
```
interface Review {
  rating: number;
  comment: string;
  user: Schema.Types.ObjectId;
}

interface Product {
  name: string;
  image: string;
  brand: string;
  category: string;
  description: string;
  reviews: Review[];
  rating: number;
  reviewCount: number;
  price: number;
  stockCount: number;
}
```

Primjer 4. Modeli resursa u bazi podataka

Uz osnovna polja i atribute, korisnik za svoj model veže i jednu metodu koja se zahvaljujući *mongoose*-u može definirati. Radi se o jednostavnoj metodi provjere zaporke, odnosno metodi koja prima zaporku u tekstualnom i čitljivom obliku te vraća *bool* vrijednost o točnosti iste. S obzirom da su zaporke kriptirane, nije moguće jednostavno usporediti dva *string*-a, no kako se za kriptiranje koristi *bcrypt* algoritam, istoimeni paket veže za sebe metodu *compare* koja uspoređuje unesenu lozinku sa onom kriptiranom u bazi. Uz tu metodu, model korisnika ima i slušača na spremanje korisnika koji sluša na promjenu zaporke, te ukoliko se ista promijenila, na razini modela kriptira zaporku te sprema u bazu.

5.3. Implementacija poslužitelja

Kao što je već spomenuto, aplikacija na strani poslužitelja se bazira na REST servisu rada što znači da ima definirane rute koje primaju i obrađuju zahtjev te na kraju vraćaju odgovor. Koristeći načela i pravila HTTP-a, poslužitelj na temelju putanje točno zna koji zahtjev obrađuje dok klijent s druge strane na temelju odgovora i statusnog koda zna kakav je ishod zahtjeva koji je poslao.



Slika 6. Jednostavni prikaz REST servisa

No, kako poslužitelj na temelju putanje zahtjeva zna što korisnik traži, iste te putanje potrebno je definirati na serveru. Sukladno tome, na strani poslužitelja implementirane su četiri glavne putanje, a to su:

- /api/products – glavna putanja za rad sa proizvodima,
- /api/users – putanja za upravljanje korisnicima,
- /api/orders – služi za rad sa narudžbama,
- /api/upload – putanja koja ima samo jednu rutu, a služi za spremanje slika proizvoda na poslužitelj.

Svaka od navedenih glavnih ruta sadrži po jednu ili više podruta koje u cijelom sklopu služe za normalno funkcioniranje cijele aplikacije na strani klijenta. Osim ruta, na strani poslužitelja je napravljeno i upravljanje greškama koje mogu proizaći iz baze na koje server nema utjecaj, te grešaka koje zahtjevi mogu vratiti zbog nedovoljno prava ili jednostavno nepostojanja tj. nemogućnosti obrade zahtjeva.

5.3.1. Middleware

Middleware se može opisati kao softverska tehnologija koja omogućuje modularnu vezu distribuiranog softvera odnosno dio softvera koji se izvršava između komunikacije dvije aplikacije. [13] U slučaju praktičnog dijela, *middleware* na strani poslužitelja ima dvije glavne uloge – predstavlja autorizacijski most te upravlja greškama. Posebno je koristan u zaštićivanju određenih putanja na strani poslužitelja s obzirom da npr. narudžbe korisnika „x“ ne bi smjele i trebale biti dostupne korisniku „y“ ili javno. Stoga, autorizacijski *middleware* predstavlja dio softvera koji se izvršava prije obrade zahtijeva te je zaslužan za blokiranje pristupa ukoliko klijent taj pristup nema.

Prije autorizacijskog *middleware*-a dolazi dio softvera koji služi za bolje upravljanje greškama. Upravljanje greškama bitan je dio procesa razvoja aplikacije jer za greške koje automatski proizlaze iz baze ili drugih izvora, Express nije dužan vratiti povratnu informaciju o njima. Zato se iz tog razloga koristi metoda koja se nalazi na samom kraju poslužitelja, te ukoliko poslužitelj sa obradom zahtjeva dođe do nje, poslužitelj je siguran da će vratiti nekakvu povratnu informaciju. Ovakav *middleware* omogućuje programeru da kroz kod izbaci novu grešku koju će *middleware* pokupiti i obraditi, umjesto da svaki puta piše isti predložak za pisanje grešaka.

Ukoliko poslužitelj prilikom obrade zahtjeva ne pronađe odgovarajuću putanju, pred sami kraj dolazi do *middleware*-a pod imenom *notFound*. Radi se o *middleware*-u koji će izbaciti grešku sa statusnim kodom 404 koju će gore opisani *middleware* pokupiti.

```
const notFound = (req: Request, res: Response, next: NextFunction) => {
  const error = new Error(`API route not found - ${req.originalUrl}`);

  res.status(404);

  next(error);
};

const errorHandler = (
  err: any,
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const statusCode = res.statusCode === 200 ? 500 : res.statusCode;

  res.status(statusCode).json({
    message: err.message,
    stack: process.env.NODE_ENV === "production" ? null : err.stack,
  });
};
```

Primjer 5. *Middleware*-ovi za upravljanje greškama

Autorizacija na aplikaciji napravljena je koristeći JSON web tokene, jednu od najpopularnijih metoda autorizacije web programiranja današnjice. JSON web token predloženi je internetski standard za stvaranje podataka s izbornim potpisom i/ili opcionalnom enkripcijom. Za Node.js postoji istoimeni paket koji olakšava stvaranje, čitanje i uništavanje tokena.

```
const protect = asyncHandler(
  async (req: Request, res: Response, next: NextFunction) => {
    const accessToken = req.cookies.Bearer;

    if (accessToken) {
      try {
        if (process.env.JWT_SECRET) {
          const decoded = jwt.verify(accessToken, process.env.JWT_SECRET);

          req.user = await UserModel.findById(
            (decoded as UserPayload).id
          ).select("-password");

          if (req.user) {
            next();
          } else {
            res.status(404);
            throw new Error("User not found.");
          }
        } else {
          console.error("JWT_SECRET missing from .env file.");
          process.exit(1);
        }
      } catch (error) {
        res.status(401);
        throw new Error("Not authorized, invalid token.");
      }
    } else {
      res.status(401);
      throw new Error("Not authorized, no token.");
    }
  }
);
```

Primjer 6. Autorizacijski *middleware*

Zahvaljujući *middleware*-u, sve rute koje želimo zaštiti te dopustiti obradu zahtjeva samo za korisnike naše aplikacije, možemo napraviti koristeći ključnu riječ *protect* (što predstavlja ime metode koja označava *middleware*) prilikom definiranja same putanje. Express će prepoznati zatraženi *middleware* te će prije obrade zahtijeva prvo proći kroz njega. U slučaju sa gornjeg primjera, ukoliko na zaštićenu rutu dođe zahtjev koji u zaglavljima nema kolačić imena *Bearer* (standardizirana riječ za autorizacijski token), automatski će javiti grešku sa kodom 401 koji predstavlja nedostatak autorizacije. Ako se token nalazi u zaglavljima zahtjeva, tada JWT paket provjerava valjanost tokena na temelju potpisa koji je spremljen u *.env*

datoteci, te ukoliko metoda *verify* ustanovi da token nije valjan, poslužitelj vraća odgovor sa statusnim kodom 401. Ukoliko token ispada dobar, s obzirom da se je kreirao na temelju korisničkog ID-a, tada slijedi provjera postoji li korisnik u bazi. Ukoliko korisnik ne postoji, poslužitelj vraća grešku sa kodom 404 i porukom *„User not found“*. Na samom kraju, ako korisnik postoji u bazi, tada se objekt korisnika sprema u prošireni *request* objekt Express-a, a poziv metode next označuje uspješan prolaz kroz *middleware* te nastavlja na obradu zahtjeva.

No s obzirom da u aplikaciji postoji i administratorski modul, opisani autorizacijski *middleware* ne sadrži podatak o statusu odnosno ulozi korisnika pa je potrebno dodati još jedan *middleware* koji ovaj puta ima fokus na ulogu korisnika. Kako je *middleware* komadić softvera koji se izvršava u određenom trenutku prije obrade zahtjeva, moguće je nizati više takvih kodova sve dok svi osim zadnjeg imaju poziv metode *next* u bloku koda za koji postoji realna mogućnost izvršavanja. U takvom slučaju, za zaštitu administratorski putanja možemo koristiti autorizacijski *middleware* koji na samom kraju prošiti objekt *request* sa korisnikom iz baze. Stoga, administratorski *middleware* može na vrlo jednostavan način provjeriti radi li se o korisniku koji ima ulogu administratora ili ne.

```
const admin = (req: Request, res: Response, next: NextFunction) => {
  if (req.user.isAdmin) {
    next();
  } else {
    res.status(401);
    throw new Error("Not authorized, no admin rights.");
  }
};
```

Primjer 7. Administratorski middleware

5.3.2. API putanje

Nakon objašnjenih pomoćnih alata korištenih u izradi poslužitelja, slijedi opis putanja koje obrađuju klijentske zahtjeve te rade izmjene na bazi podataka. Radi lakše čitljivosti i održivosti programskog koda, rute su od svojih metoda odvojene u dva zasebna direktorija: *controllers* i *routes*. Direktorij *routes* sadrži sve definicije putanja, dok direktorij *controllers* sadrži metode koju svaka ruta obrađuje. Tako su i datoteke raspodijeljene da svaka u sebi sadrži opis putanje za koju se veže, npr. *orderControllers.ts* odnosno *orderRoutes.ts*.

U nastavku slijedi kratki opis svake rute u formatu „/api/ruta“ – Metoda (*middleware*) - opis. Dakle, u datoteci *orderRoutes.ts* koja označava putanje za narudžbe razlikujemo:

- „/api/orders“
  - GET (admin) – dohvaća sve narudžbe sa potencijalnom paginacijom,
  - POST (auth) – kreira novu narudžbu na temelju košarice,

- „/api/orders/my-orders“
    - GET (auth) – dohvaća narudžbe za trenutno prijavljenog korisnika,
- „/api/orders/{id}“
    - GET (auth) – dohvaća narudžbu po danom ID-u,
    - DELETE (admin) – briše narudžbu na temelju danog ID-a,
- „/api/orders/{id}/invoke“
    - POST (auth) – kreira sesiju za plaćanje preko Stripe-a,
- „/api/orders/{id}/deliver“
    - PUT (admin) – označava dostavu kao dostavljenu,
- „/api/orders/webhook“
    - POST (stripe) – putanja koju koristi Stripe za slanje podataka o provedenom plaćanju.

Druga datoteka sadrži definiciju putanja vezanih za rad sa proizvodima, pa tako razlikujemo:

- „/api/products/“
    - GET (-) – dohvaća proizvode sa potencijalnom paginacijom i filterom,
    - POST (admin) – kreira novi proizvod na temelju tijela zahtjeva,
- „/api/top“
    - GET (-) – dohvaća tri najbolje ocjenjena proizvoda za početnu stranicu,
- „/api/products/{id}“
    - GET (-) – dohvaća detalje proizvoda na temelju ID-a,
    - PUT (admin) – ažurira proizvod na temelju ID-a i tijela zahtjeva,
    - DELETE (admin) – briše proizvod sa odgovarajućim ID-em,
- „/api/products/{id}/review“
    - POST (auth) – unosi novu recenziju proizvoda za prijavljenog korisnika.

Treća datoteka odnosi se na putanju za pohranu fotografija te sadrži samo jednu metodu tipa POST na putanji „/api/uploads/“ koja je zaštićena administratorskim *middleware*-om. Obzirom da putanja služi za pohranu fotografija, prije spremanja na poslužitelj provjerava se tip poslane datoteke te poslužitelj vraća grešku ukoliko datoteka nije *jpg*, *jpeg* ili *png* formata.

Četvrta, ujedno i zadnja datoteka definira putanje za rad sa korisnicima, a to su:

- „/api/users/“
    - GET (admin) – dohvaća korisnike sa potencijalnom paginacijom i filtriranjem,

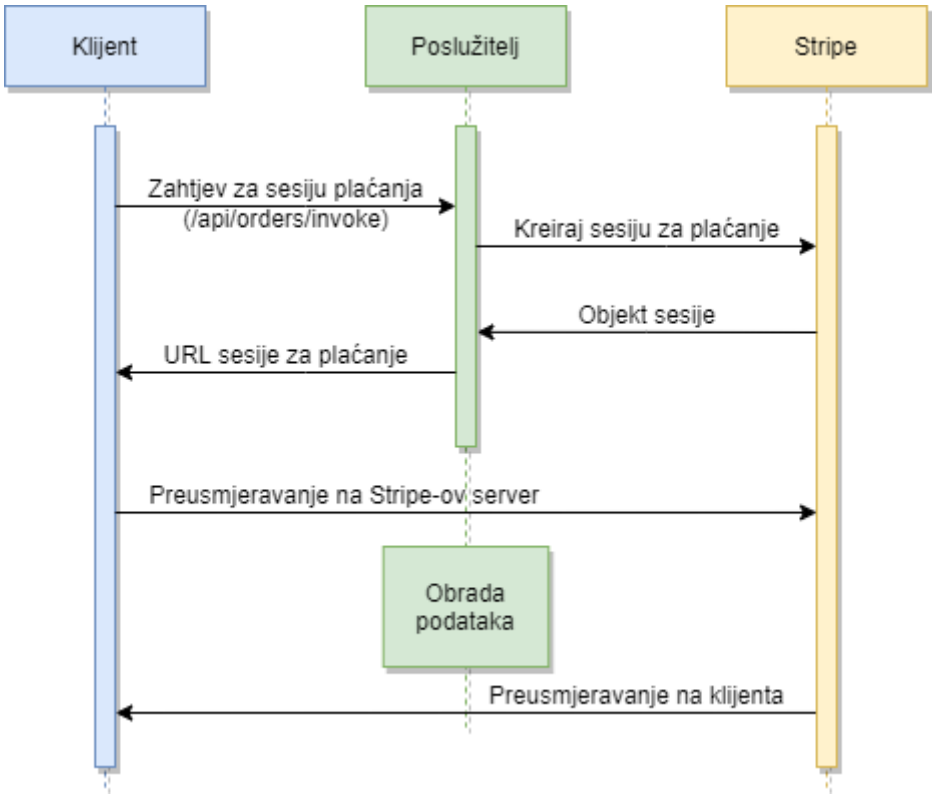
    - POST (-) – registrira novog korisnika na temelju poslanih podataka,
- „/api/users/sign-in“
    - POST (-) – kreira JWT token i prijavljuje korisnika ukoliko su poslani podaci valjani,
- „/api/users/log-out“
    - POST (auth) – poništava generirani token prilikom prijave ili registracije,
- „/api/users/validate-session“
    - POST (auth) – provjerava valjanost sesije prilikom učitavanja stranice,
- „/api/users/cart“
    - GET (auth) – vraća sadržaj košarice za prijavljenog korisnika,
    - POST (auth) – dodaje odnosno briše stvari iz košarice za prijavljenog korisnika,
- „/api/users/profile“
    - GET (auth) – dohvaća podatke o prijavljenom korisniku,
    - PUT (auth) – ažurira podatke za prijavljenog korisnika na temelju poslanih podataka,
- „/api/users/{id}“
    - GET (admin) – dohvaća detalje korisnika na temelju ID-a,
    - PUT (admin) – ažurira korisničke podatke na temelju tijela zahtjeva,
    - DELETE (admin) – briše korisnika iz baze na temelju ID-a.

5.3.3. Sigurnost aplikacije

Sigurnost web aplikacija dosta je škakljivo pitanje u web programiranju. Iako korisnik nema pristup poslužitelju i njegovim podacima, korisnik na klijentskoj strani može promijeniti određene varijable te na taj način „privremeno“ dobiti administratorska prava – doduše, samo na svojoj strani. Baš iz tog razloga je napravljen i ranije opisani *middleware* koji će provjeravati status sa onim u bazi, a ne sa onim kojeg klijent šalje.

S druge strane, implementacija plaćanja može isto tako biti škakljivo pitanje. Naime, prilikom kreiranja sesije za plaćanje, Stripe-ov API traži dva obavezna podatka, *succes_url* i *cancel_url* koji određuju lokaciju na koju Stripe preusmjeri u oba slučaja uspješnosti odnosno neuspješnosti plaćanja. Netom nakon generiranja sesije, ID novo generirane sesije spajamo sa narudžbom kako bi kasnije mogli prema istom ID-u vidjeti koja je narudžba u pitanju. U praktičnom dijelu, implementacija navodi lokaciju detalja narudžbe, što predstavlja isti ekran s kojeg je korisnik došao na plaćanje, samo što ovaj puta u parametrima pretraživanja ima dodatnu varijablu status koja ima vrijednost *success* ili *cancel*. Naravno, ta oznaka služi samo da dadne do znanja front-end aplikaciji koju povratnu informaciju korisniku treba prikazati, dok

se pravo plaćanje odvija preko putanje „api/orders/webhook“. Ta putanja je tu iz samo jednog razloga – da bi Stripe na nju slao podatke o generiranju novih sesija i plaćanjima. U ovom trenutku nas zanima jedino je li korisnik platio narudžbu stoga putanja osluškuje i radi obradu narudžbe samo ako je u pitanju tip događaja koji ima vrijednost *„checkout.session.completed“*. U slučaju da putanja za *webhook*-ove ne radi, uvijek postoji plan B koji manualno provjerava je li narudžba plaćena sa Stripe-ovim API-em preko ID-a sesije koja je povezana sa narudžbom. Nakon svih ovih koraka, sigurni smo da će korisničko iskustvo što se plaćanja tiče biti besprijekorno fluidno i da neće doći do problema.



Slika 7. Tijek plaćanja koristeći Stripe

Posljednja stvar što se sigurnosti tiče jesu korisničke sesije. U inicijalnoj verziji aplikacije, token koji poslužitelj pošalje prilikom prijave ili registracije spremao se u lokalnu pohranu internetskog preglednika, kao i košarica. No sve web stranice izložene su tzv. skriptanjem na više mjesta (eng. *Cross-Site Scripting*) preko kojeg se uz minimalno truda može izvući cijela lokalna pohrana ili kolačići te se na taj način oduzeti korisnička sesija. Jedno od rješenja je korištenje *HttpOnly* kolačića koji se generiraju na serveru, šalju klijentu koji ih nakon tog šalje sa svakim sljedećim zahtjevom.

```
const authUser = asyncHandler(async (req: Request, res: Response) => {
  const { email, password } = req.body;

  const user = await UserModel.findOne({ email });

  if (user && (await user.checkPassword(password))) {
    const userToken = generateToken(user._id);

    await user.populate("cart.product");

    res
      .cookie("Bearer", userToken, {
        httpOnly: true,
        expires: new Date(Date.now() + 1000 * 60 * 60 * 24 * 7),
        sameSite: "lax",
      })
      .json({
        user: {
          _id: user._id,
          name: user.name,
          email: user.email,
          isAdmin: user.isAdmin,
          loyaltyPoints: user.loyaltyPoints,
          token: userToken,
        },
        cart: user.cart,
      });
  } else {
    res.status(401);
    throw new Error("Invalid email or password.");
  }
});
```

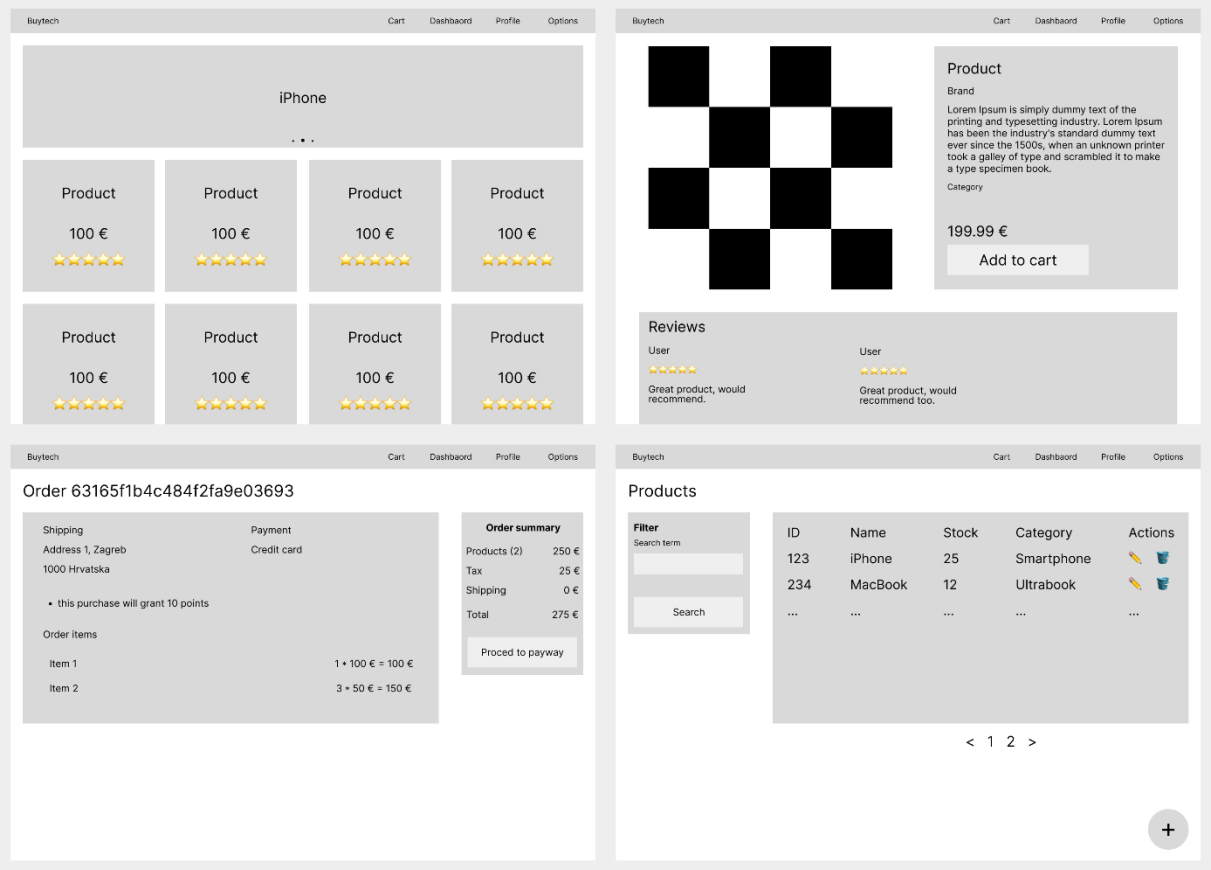
Primjer 8. Putanja za autentifikaciju korisnika

Na gornjem primjeru možemo vidjeti API putanju za autentifikaciju korisnika. Ranije spomenuto, metoda na modelu korisnika *checkPassword* provjerava unesenu zaporku te ukoliko je zaporka točna, generira JWT token na korisničkog ID-a i tajne koja je zapisana u *.env* datoteci. Nakon generiranja, poslužitelj odgovoru pridružuje zaglavlje kolačić imena *Bearer* sa *httpOnly* zastavicom postavljenom na *true* i trajanjem od tjedan dana.

Uz token, u lokalnu pohranu se u prvim verzijama aplikacije spremala i cijela košarica. Samim time, košarica nije bila vezana za korisnički račun već za internetski preglednik što znači da se kupovina nije mogla nastaviti na drugom uređaju te bi se košarica izgubila prilikom odjave. Osim gubljenja košarice, podaci su se dohvaćali direktno iz lokalne pohrane kojoj korisnik ima pristup i pravo ju mijenjati čime je došlo do mogućnosti promjena cijene te kupovine proizvoda po nerealnim cijenama. Takav propust jednostavno je riješiti spremanjem košarice odnosno ID-a proizvoda i količine u bazu podataka što dodaje dodatni plus korištenja košarice kroz više uređaja.

5.4. Implementacija klijenta

Prije ikakve izrade aplikacije odnosno front-end dijela, provedeno je malo vremena na izradu okvira stranice (eng. *wireframe*) glavnih stranica. Na samom početku ideja karusele činila se kao dobra opcija za početni ekran ispod čega slijedi lista proizvoda. Što se ekrana detalja proizvoda tiče, on bi trebao sadržavati sliku proizvoda, informacije o proizvodu i sekciju koja sadrži recenzije proizvoda. Ekran detalja narudžbe sadrži prikaz osnovnih informacija svaki narudžbe, adresu dostave, način plaćanja, listu proizvoda u narudžbi kao i ukupnu cijenu narudžbe. Za dobro korisničko iskustvo administratorskog modula odlično odgovara lista podataka koja u svojoj neposrednoj blizini sadrži karticu sa mogućim filterima iste.

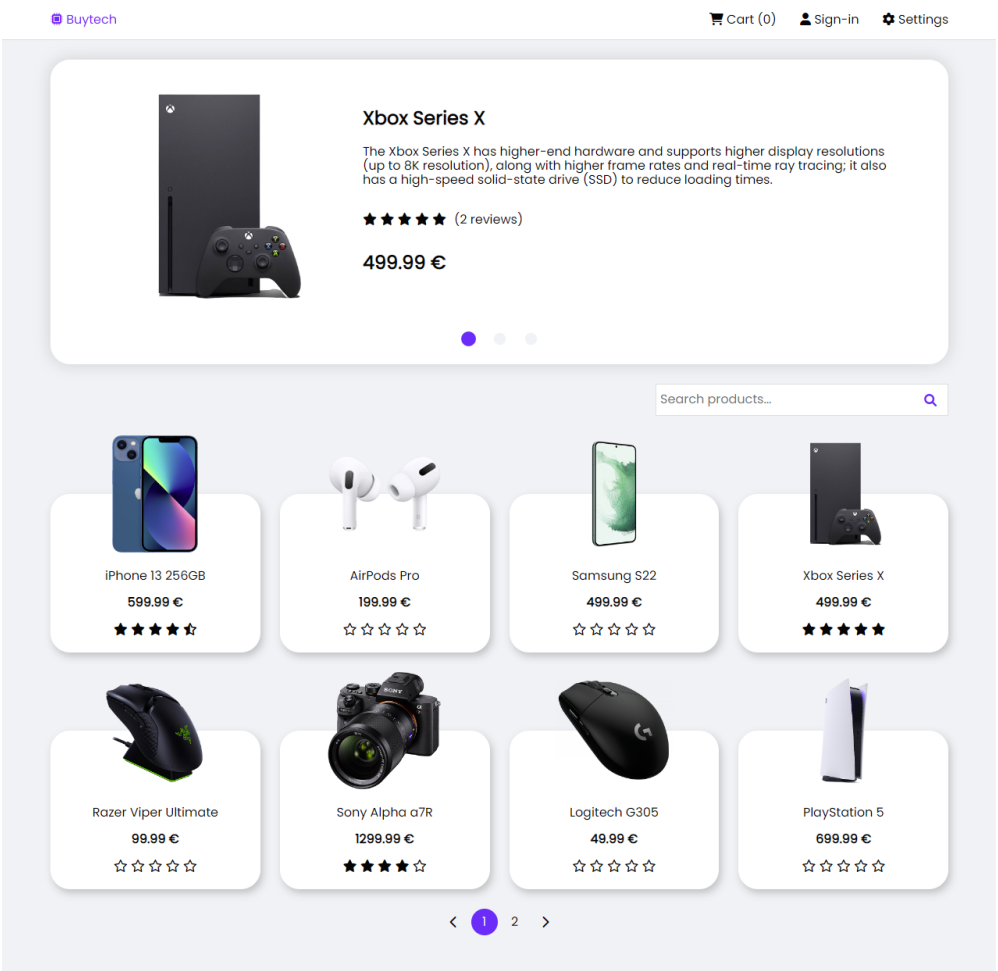


Slika 8. *Wireframe* glavna četiri ekrana

Što se samog dizajna tiče, on je rađen usputno, no generalna ideja izgleda aplikacije razrađena je kroz *wireframe* koji je dostojan izgledu finalne verzije aplikacije. Skice aplikacije inspirirane su minimalnim dizajnom koji je danas sastavni dio svake moderne stranice. S druge strane, iako dizajnom minimalno, svaka stranica sadrži sve potrebne podatke koje treba prikazivati, gdje recimo ekran detalja narudžbe sadrži podatke o narudžbi i proizvodima, ali nigdje se ne pojavljuje opis proizvoda koji je u ovom trenutku korisniku – nepotreban.

5.4.1. Početna stranica i općenito o aplikaciji

Klijentska aplikacija izrađena je kao aplikacija na jednoj stranici sa iscrtavanjem stranice na strani poslužitelja. Ideja aplikacije bila je izrada što više komponenata koje bi se mogle koristiti kroz više stranica ili drugih komponenata kako bi se smanjila količina programskog koda, ali sadržaj stranice ostao jednako bogat kao i na nekim većim stranicama. Takav način razmišljanja već je uočljiv na početnoj stranici u nekoliko navrata. Prvi takav primjer jest primjer komponente ocjene proizvoda koja, ovisno koja se ocjena pošalje kao *prop* (vrijednost koja se može poslati komponenti), toliko zvjezdica oboja. Radi jednostavnosti komponente, zvjezdice sa ukupnom ocjenom mogu biti u razmaku od 0.5, što znači da zvjezdica ili nije obojana, ili je polovično obojana ili je cijela obojana. Drugi primjer komponente je kartica proizvoda koja se ponavlja na početnoj stranici osam puta. Komponenta kartice proizvoda kao *prop* prima objekt proizvoda identičan onome sa baze te prikazuje tri osnovna podatka – fotografiju, naziv i cijenu proizvoda.



Slika 9. Početna stranica aplikacije

Na samom vrhu stranice nalazi se minimalan karusel od tri stupnja koji se svakih nekoliko sekundi prebaci na sljedeći stupanj, a sastoji se od tri najbolje ocijenjena proizvoda u cijeloj web trgovini. Ispod karusela nalazi se polje za unos koje na temelju upisanog pojma pretražuje proizvode po nazivu, brendu i kategoriji. Nakon polja za unos vidljiva je mreža sa karticama proizvoda, a na dnu komponenta paginacije koja se, kao i komponenta za ocjenu proizvoda, ponavlja kroz aplikaciju te ne sadrži nikakvo stanje.

```
<template>
  <nuxt-link :to="localePath(`/products/${product._id}`)">
    <div class="product-card">
      <div class="image-wrapper">
        <img
          :src="$utils.getImageUrl(product.image)"
          class="product-image" />
      </div>
      <p>{{ product.name }}</p>
      <p>
        <b>{{ product.price }} €</b>
      </p>
      <Rating :value="product.rating" />
    </div>
  </nuxt-link>
</template>

<script lang="ts">
import Vue, { PropType } from "vue";

import { Product } from "~/types/Product";

export default Vue.extend({
  name: "ProductCard",

  props: {
    product: Object as PropType<Product>,
  },
});
</script>
```

Primjer 9. Komponenta kartice proizvoda bez stilova

Gore spomenuta komponenta kartice proizvoda ne sadrži nikakvo stanje niti metode, stoga ju možemo nazvati isključivo prezentacijskom komponentom. Ona dakle uzima dobivene podatke, u ovom slučaju proizvod, te prikazuje sliku, naziv i cijenu proizvoda, skupa sa ocjenom koja je također prezentacijska komponenta. Naravno, ukoliko proizvod nema ocjene tada je proslijeđena varijabla jednaka 0 odnosno sve zvjezdice su prazne.

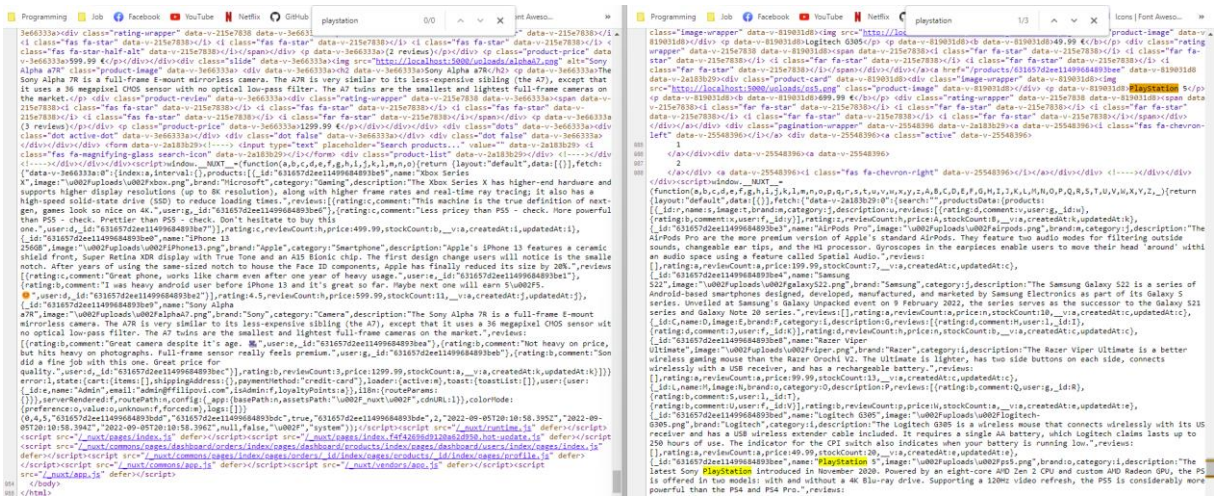
5.4.2.SSR, SEO i inicijalno učitavanje aplikacije

Kako se učitavanje stranice na strani poslužitelja spominje u više navrata, vrijeme je da opišemo kako i zašto radi te zašto je korišten Nuxt, a ne obični Vue. Ranije spomenuto, aplikacija koju pokreće Vue se cijela inicijalizira, pokreće i dohvaća podatke na strani klijenta što uzrokuje nedostatak informacija na izvoru stranice koja se koristi za SEO na svim pretraživačima. Laički rečeno, aplikacija vraća prazan izvor stranice, ali normalno prikazuje sve podatke. Upravo taj problem jedan je od glavnih problema aplikacija na jednoj stranici. U ovom trenutku u igru ulazi Nuxt, koji uz brojne dodatke za Vue, dodaje i dvije metode životnog ciklusa stranicama Vue aplikacije. To su metode pod nazivima *fetchData* i *fetch*, koje na kraju imaju isti produkt, no razlikuju se u tome što Nuxt neće navigirati na stranicu koja koristi *fetchData* sve dok se podaci ne dohvate, dok se koristeći *fetch* može prikazati indikator učitavanja podataka. Osim indikatora, *fetch* ima pristup *this* varijabli stranice, dok *fetchData* vraća dohvaćeni objekt i automatski ga sprema u željenu varijablu. Koristeći jednu od navedene dvije metode na stranici, Nuxt automatski prepoznaje da je u pitanju SSR (osim ako se u postavkama aplikacije ne odredi drugačije) te vraća ispunjen izvor stranice sa vidljivim podacima iste.

Indikator učitavanja stvara bolje korisničko iskustvo od blokiranja stranice i ne prikazivanja ničega, stoga je u praktičnom dijelu svugdje korištena *fetch* metoda koja se činila prikladnijom. Jedan od takvih primjera je upravo i početna stranica aplikacije koja sadrži listu proizvoda. Osim samih proizvoda koji se pojavljuju korištenjem SSR-a, bitno je napomenuti da Nuxt prepoznaje i sve poveznice među aplikacijom (ukoliko se koristi komponenta <*nuxt-link*>) te se i one pojavljuju na izvoru stranice što znači da jedan Google-ov program ima mogućnost „hodanja“ po stranici i dohvaćanja podataka za SEO.

```
async fetch() {
  let url = new URL(`${this.$axios.defaults.baseURL}/api/products`);

  const page = this.$route.query.page || 1;
  url.searchParams.set("page", page);

  const keyword = this.$route.query.keyword;
  if (keyword) {
    url.searchParams.set("keyword", keyword);
  } else {
    this.search = "";
  }

  const productsData = await this.$axios.$get(url.toString());

  this.productsData = productsData;
}
```

Primjer 10. *Fetch* metoda za dohvaćanje proizvoda



Slika 10. Primjer izvora stranice koristeći *fetch* metodu i običnog dohvaćanja podataka

Iako je glavna značajka učitavanja na strani poslužitelja upravo ispunjavanje izvora stranice, osim toga iskorištene su i druge stvari radi stvaranja boljeg korisničkog iskustva. Jedna od takvih prednosti je inicijalno ispunjavanje globalnog stanja aplikacije. Za početak, globalno stanje aplikacije predstavlja skup varijabli dostupnih kroz cijelu aplikaciju, za razliku od onog stanja koje je definirano na jednoj stranici i komponenti. U običnom JavaScript programiranju, jedan takav dodatak zove se Redux koji je jedan od najpopularnijih paketa u zajednici web programera. Za Vue postoji imenom malo drugačija varijanta imena Vuex.

Vuex služi za definiranje stanja aplikacije kroz trgovine (eng. *store*) koje se sastoje od funkcije koja vraća objekt stanja, a uz objekt stanja može imati objekt mutacija, akcija, i *getter*-a. Jedini način da se promijeni stanje u *store*-u je korištenjem mutacija koje su vro slične događajima. Akcije su slične mutacijama, ali umjesto promjene stanje one pozivaju mutacije te se mogu sastojati od proizvoljnih asinkronih operacija. Glavna uloga *getter*-a je vraćanje podataka no obično se koriste kod izračuna nekih kompleksnijih stanja, a ne čistog objekta stanja (npr. izračun cijene košarice na temelju liste proizvoda u njoj). U slučaju praktičnog primjera postoje četiri glavna *store*-a, a to su:

- *store* korisnika – sadrži objekt korisnika te metode za prijavu i odjavu korisnika koje dodaju ili brišu korisnika te *getter*-e poput je li korisnik ulogiran ili ne,
- *store* košarice – sastoji se od liste proizvoda u košarici, objekta za adresu dostave i načina plaćanja kao i popratnih mutacija za upravljanje istima. Uz mutacije sadrži i *getter*-e za dohvati ukupne cijene i ukupnog broja proizvoda,
- *store* notifikacija – trgovina za vlastitu Toast komponentu, radi se o listi objekata tipa Toast,
- *store* učitavanja – najjednostavnija trgovina, sadrži samo status aktivnog učitavanja stranice, a koristi se za prikaz komponente za učitavanje.

Važno je napomenuti da se trgovine u svim aplikacijama na jednoj stranici inicijaliziraju u vrijeme dolaska na aplikaciju pa je potrebno inicijalno popuniti vrijednosti istih. Tako bi se za svako ponovno učitavanje stanice trebala ponovno dohvatiti i provjeriti korisnička sesija te ispuniti trgovinu korisnika i košarice. Navigiranje na stranici (ukoliko je napravljeno na preporučen način) neće ponovno učitavati stranicu što znači da trgovine ostaju definirane tijekom rada aplikacije. Kada ne bi bilo SSR-a i njegovog inicijalnog učitavanja, aplikacija bi prilikom inicijalnog učitavanja stranice trebala napraviti poziv na poslužitelj koji bi provjerio sesiju korisnika (koristeći *HttpOnly* kolačić) te ispuniti objekt korisnika ukoliko je sesija valjana. Taj dio može se odraditi na pripremanju stranice na strani poslužitelja zahvaljujući Nuxt-ovoj gotovoj metodi *nuxtServerInit* koja se okida prilikom inicijalnog učitavanja.

```
async nuxtServerInit(context: any, nuxt_context: Context) {
  if (nuxt_context.req.headers.cookie?.includes("Bearer")) {
    const sessionResponse = await nuxt_context.$axios
      .$post(
        `
          ${nuxt_context.$axios.defaults.baseURL}
          /api/users/validate-session
        `
      )
      .catch((_: any) => {});

    if (sessionResponse) {
      context.commit("user/signInUser", sessionResponse.user);
      context.commit("cart/updateCart", sessionResponse.cart);
    }
  }
}
```

Primjer 11. *nuxtServerInit* metoda

Gornja metoda ovjerava korisničku sesiju te spremi korisnika u korisnički *store* i podatke košarice u *store* košarice. S obzirom da se taj dio izvršava na strani poslužitelja, korisnik ne vidi promjenu odnosno trzanje zaglavlja i navigacije aplikacije prilikom početnog učitavanja te automatski ima učitanu košaricu što, uzevši u obzir brzinu izvršavanja zahtjeva, čini korisničko iskustvo jako kvalitetnim.

Nuxt osim SSR-a omogućuje i definiranje svih atributa koji se nalaze u glavi HTML dokumenta poput opisa, naslova i sličnih što dodatno pospješuje SEO aplikacije. Svaka stranica aplikacije ima specificiran naslov stranice sa aktivnom lokalizacijom, što znači da će početna stranica na engleskom jeziku biti „Home“, dok će na hrvatskom pisati „Početna“.

5.4.3.Komponente

Komponente su sastavi dio svake moderne aplikacije te imaju dva glavna cilja: razdvojiti kod na više dijelova te omogućiti ponovno korištenje radi smanjenja pisanja koda koji bi u tom trenutku predstavljao predložak. Baš iz istog razloga je napravljeno nekoliko komponenata koje se ponavljaju uz već gore spomenute koje su bile isključivo prezentacijske.

Prve dvije komponente koje omogućuju jednostavno ponovno korištenje su komponenta modala i dijaloga. Modal služi kao omot oko sadržaja istog, prekriva cijeli ekran, a sadržaj centrira na sredinu ekrana. Modali se koriste kod korisničkih unosa, bilo recenzije ili u administratorskom modulu za dodavanje i uređivanje zapisa. Modal na sebi ima i slušač klikova koji će, ukoliko korisnik klikne van modala (na prekriveni dio aplikacije), poslati događaj komponenti koja ga konzumira kako bi se mogao natrag sakriti.

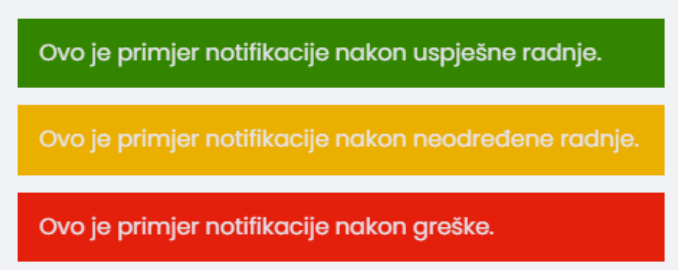
Komponenta dijaloga slična je komponenti modala uz nekoliko promjena. Za razliku od modala, komponenta dijaloga prima dva glavna *prop*-a, zaglavlje dijaloga i tijelo. Na taj način dobije se komponenta koja nije specifična za neki slučaj već se radi o agnostičnoj komponenti koja se može koristiti u bilo kojem slučaju. Prilikom korištenja komponente, komponenti se pridodaje i direktiva *v-show* koja označava je li komponenta vidljiva ili nije. Radi se o jednoj od brojnih Vue direktiva koje pomažu u smanjivanju pisanja koda predloška s obzirom da se odnose na jednostavne stvari poput prikazivanja, iscrtavanja u DOM-u i slično. Komponenta dijaloga zapravo je modal stoga u sebi sadrži Modal omot oko svega, a sadržaj dijaloga jest naslov koje je ujedno i zaglavlje, tijelo te na dnu dva gumba koja predstavljaju podnožje. Klikovi na gumbove šalju događaj komponenti koja konzumira dijalog na temelju kojih se obrađuje željena akcija.

```
// Dialog.vue
<template>
  <Modal @hideModal="this.$emit('onCancel')">
    <div class="dialog-wrapper">
      <h3>{{ title }}</h3>
      <p>{{ description }}</p>
      <div class="buttons">
        <button @click="this.$emit('onCancel')" class="button-link">
          {{ $t("button_cancel") }}
        </button>
        <button @click="this.$emit('onOk')" class="button-destructive">
          {{ $t("button_confirm") }}
        </button>
      </div>
    </div>
  </Modal>
</template>
```

```
<script lang="ts">
import Vue from "vue";
export default Vue.extend({
  name: "Dialog",

  props: {
    title: String,
    description: String,
  },
});
</script>

// primjer korištenja komponente dijaloga
<Dialog
  v-show="showDialog"
  :title="Naslov dijaloga"
  :description="Tijelo dijaloga za prikaz sadržaja"
  @onOk="handleDialogOk"
  @onCancel="handleDialogCancel"
/>
```

Primjer 12. Komponenta dijaloga i primjer korištenja

Još jedna zanimljiva komponenta je komponenta notifikacija na aplikaciji. Notifikacije u aplikaciji označavaju povratnu informaciju za korisnika prilikom izvršavanja neke radnje, bila ona uspješna ili neuspješna. U ovom slučaju radi se o nešto složenijoj komponenti koja za prikaz koristi podatke iz globalnog *store*-a notifikacija. Komponenta prilikom kreiranja postavlja slušač na promjene stanja notifikacija, te se na svaku promjenu (bilo dodavanje ili brisanje notifikacije) povuče novo stanje na temelju kojeg se prikažu poruke. Komponenta koristi listu poruka kako bi u isto vrijeme mogla prikazati više poruka, koja svaka na sebi ima rukovatelja klikova koji ju briše iz liste. Svako novo pojavljivanje poruke pokreće tajmer od pet sekundi nakon kojeg se poruka automatski obriše iz liste.



Slika 11. Izgled notifikacija na aplikaciji

Sama komponenta je relativno jednostavna, koristi Vue *v-for* direktivu koja je dosta često korištena kroz aplikaciju. Radi se o direktivi koja iterira kroz danu listu ili objekt te vraća svaki element i iscrtava ono što se nalazi unutar elementa na kojoj se direktiva nalazi. Kada se koristi *v-for* direktiva potrebno je istom elementu zadati i *key* atribut koji korisniku i programeru nije potreban, dok Vue na temelju njega zna upravljati DOM-om kada se dogodi

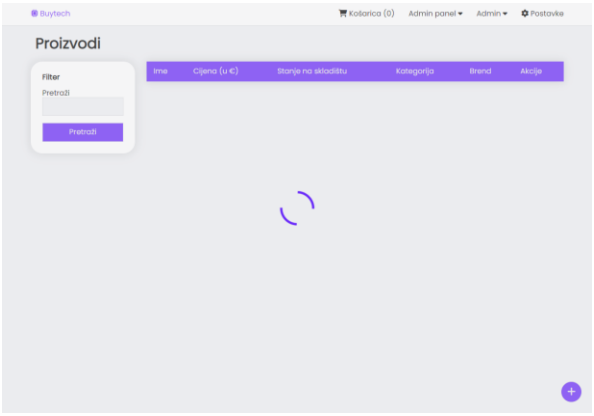
promjena. Izgled, odnosno boja poruke napravljena je dinamički te svakom kontejneru notifikacije dodaje klasu jednakog imena kao i tip poruke što može biti *„success“*, *„warning“* ili *„error“*.

```
<template>
  <div class="toast-container">
    <div
      v-for="toast in toastList"
      :key="toast.id"
      :class="`toast ${toast.type}`"
      @click="handleRemoveMessage(toast)"
    >
      {{ toast.message }}
    </div>
  </div>
</template>
```

Primjer 13. Iterator poruka na komponenti notifikacija

Paginacija je, kao i ostatak komponenata, dinamična te se može pojaviti na svim stranicama kojima bi straničenje moglo biti potrebno (početna stranica i stranice sa tablicama). Komponenta zahtijeva dva *prop*-a, a to su ukupan broj stranica te stranica na kojoj se trenutno nalazimo. Straničenje je napravljeno na strani poslužitelja te se pomoću parametara pretraživanja zahtijeva određena stranica tako što se na stranici koja ima paginaciju doda *„?page={stranica}“* na kraj URL-a.

Posljednja komponenta vrijedna spomena je komponenta globalnog učitavanja. Radi se o tzv. *Spinner*-u koji se, zahvaljujući *plugin*-ovima koje Nuxt nudi, automatski prikazuje prilikom svakog zahtijeva i prikazana je za vrijeme trajanja zahtijeva. Iako ne predstavlja baš najbolje korisničko iskustvo, na ovaj način se tjera korisnika da pričeka što je zatražio na stranici te vizualno raspoznaje da se nešto u pozadini događa. U slučaju da je na stranici moguć neki duži zahtjev prema poslužitelju, vjerojatno bi se za takav zahtjev komponenta ignorirala i prikazala samo na razini komponente.



Slika 12. Prikaz komponente globalnog učitavanja

5.4.4.Layout, middleware i plugin-ovi

Layouts, *middleware* i *plugin* direktoriji su koji pripomažu u izradi aplikacije u Nuxt-u čime se gradi bolje programersko iskustvo.

Layouts direktorij služi za izradu rasporeda elemenata na aplikaciji koji se pojavljuju diljem cijele aplikacije kako bi se smanjilo ponavljanje programskog koda. Zaglavlje aplikacije je odličan primjer jedne takve komponente. Radi se o komponenti koja je prisutna na svakom ekranu aplikacije te ne postoji niti jedan ekran koji bi trebao sakriti zaglavlje. Ručni unos zaglavlja na svaku stranicu stvorilo bi nepotrebno ponavljanje koda. Nuxt će prilikom pokretanja aplikacije tražiti *default.vue* datoteku u *layouts* direktoriju te na temelju nje će složiti raspored svake stranice. S druge strane, *default.vue* datoteku možemo smatrati jednom od incijalnih jer će se pokrenuti na svakoj stranici. Takva stranica u sebi sadrži komponentu zaglavlja, komponentu notifikacija i komponentu globalnog učitavanja s obzirom da se one mogu pojaviti na svim stranicama. Osim njih, sadrži i komponentu tipa *<Nuxt />* koja je zapravo sadržaj stranice koja se iscrtava na ekranu.

```
<template>
  <div>
    <Header />
    <Toast />
    <Spinner v-if="$store.state.loader.active" />
    <Nuxt />
  </div>
</template>


<script lang="ts">
import Vue from "vue";

export default Vue.extend({
  mounted() {
    this.$store.dispatch("cart/stateShippingAddressInit");
  },
});
</script>
```

Primjer 14. HTML sadržaj default.vue datoteke

Komponenta globalnog učitavanja za prikaz koristi *v-if* direktivu koja ju postavlja u DOM samo ako je stanje učitavanja postavljeno na *true*. S obzirom da se navedena datoteka izvršava na svakoj stranici, ona prilikom učitavanja pokreće akciju sa *store*-a košarice te iz lokalne pohrane izvlači podatke o adresi koje je korisnik upisao prilikom unosa narudžbe što stvara bolje korisničko iskustvo jer korisnik ne treba svaki put upisivati svoju adresu.

Osim *default.vue* datoteke, u *layouts* direktoriju nalazi se i *error.vue* datoteka koja predstavlja definiciju stranice koja se prikazuje ukoliko korisnik pokuša otići na neku stranicu aplikacije koja ne postoji.

Middleware direktorij sastoji se od datoteka koje imaju istu namjenu kao i one na strani poslužitelja – provesti dio koda prije nekog zahtijeva. U ovom slučaju, *middleware* se koristi kao blokada na stranice na koje korisnik nema pristup. Recimo, ne prijavljeni korisnik nema pristup profilu, dok obični korisnik nema pravo pristupa administratorskom modulu. S druge strane, prijavljeni korisnik ne može doći do ekrana prijave ili registracije prije nego se odjavi. *Middleware* će preusmjeriti korisnika na stranicu koja je zadana u samoj definiciji *middleware*-a, gdje će npr. korisnika koji nema administratorska prava *middleware* prebaciti na početnu stranicu ukoliko korisnik pokuša doći do administratorskog modula.

```
// admin middleware
export default ({ store, redirect, localePath }: any) => {
  if (!store.getters["user/isAdmin"]) {
    return redirect(localePath(`/`));
  }
};

// korištenje admin middleware-a
export default Vue.extend({
  name: "dashboard",

  middleware: "admin",
});
```

Primjer 15. Admin *middleware* i njegova primjena

Middleware na klijentskoj strani za validaciju i autorizaciju koristi varijable koje čita sa klijentske strane, a te varijable se mogu uz malo znanja promijeniti. Tako recimo korisnik koji nije administrator si može postaviti zastavicu *isAdmin* na *true* te će mu se na zaglavlju pojaviti padajući izbornik za admin panel. Naravno, dodatna validacija je napravljena na strani poslužitelja pa poslužitelj neće vratiti nikakve podatke za stranice koje se nalaze na administratorskom modulu.

Plugin direktorij sadrži datoteke koje služe da bi proširile postojeći kontekst Nuxt-a ubrizgivanjem novih svojstava i metoda ili dodavanjem neke vrste *middleware*-a već postojećem *plugin*-u. Razlikujemo tri datoteke, gdje datoteka *axios.ts* sadržava slušače na zahtjeve prema poslužitelju te prilikom slanja zahtjeva postavlja vrijednost globalnog učitavanja na *true*, dok se na odgovor ili grešku postavlja nazad na *false*. Upravo zahvaljujući toj vrijednosti se prikazuje *Spinner*.

Druga datoteka služi za ubrizgivanje prilagođenog *plugin*-a imena *toast* za upravljanje notifikacijama na aplikaciji. On se sastoji od dvije metode, a to su *showMessage* koja dodaje novu poruku u listu notifikacija i *removeMessage* koja briše odabranu poruku iz liste. Nakon korištenja bilo koje od ove dvije funkcije, komponenta *Toast* dobiva podatke i prikazuje poruke. Treća datoteka je zapravo zamjena za česti folder imena *utils* koji se sastoji od pomoćnih metoda koje se koriste kroz više komponenata. Radi se o *plugin*-u imena *utils* koji u sebi ima metode *getImageUrl* i *getPrettyTime*. Metoda *getImageUrl* vraća putanju do slike na poslužitelju s obzirom da se u bazu sprema samo relativna putanja, dok metoda *getPrettyTime* pretvara format vremena iz baze u ljepši i lakše čitljiviji format.

```
// korištenje plugin-a toast
this.$toast.showMessage({
  message: "Poruka",
  type: "success",
});
this.$toast.removeMessage(poruka: Toast);

// korištenje plugin-a utils
this.$utils.getImageUrl("putanja");
this.$utils.getPrettyTime("datum");
```

Primjer 16. Korištenje plugin-ova

5.4.5.Internacionalizacija i lokalizacija

Internacionalizacija i lokalizacija je odrađena koristeći službeni dodatak Nuxt-a pod imenom „i18n“ što zapravo označuje skraćenicu u računalstvu za internacionalizaciju i lokalizaciju. Radi se o prilagodbi aplikacije različitim jezicima, a pokriveni su engleski i hrvatski jezik. Internacionalizacija je odrađena samo na strani klijenta, stoga opisi i kategorije proizvoda uvijek su na engleskom jeziku. Isto vrijedi i za korisničke recenzije.

Zadani jezik lokalizacije je engleski kao i jezik zamjene ukoliko se traženi pojam ne nalazi za odabrani jezik. Zanimljiva stvar ove funkcionalnosti je modularnost tj. za dodavanje novog jezika potrebno je definirati prijevode koji su zapravo kopija engleskog jezika te definiranje nove lokalizacije u samoj *nuxt.config.js* datoteci. Nakon tog, novoj lokalizaciji se može pristupiti mijenjanjem URL-a na razini preglednika ili dodavanjem nove opcije u listu opcija aplikacije u zaglavlju.

Sva navigacija koristi *localePath(„putanja“)* metodu koja vraća danu putanju sa prefiksom trenutno korištene lokalizacije. Prijevodima u aplikaciji se pristupa korištenje *plugin*-a *„t“* koji na temelju danog pojma provjerava postoji li prijevod za taj pojam u traženom jeziku.

```
<label required for="mail">{{ $t("email-label") }}</label>
<input
    v-model="formData.email"
    :placeholder="$t('email-placeholder')"
    type="text"
    id="mail"
    required
/>
```

Primjer 17. Korištenje lokalizacije

5.4.6. Tamna tema

Dizajn same aplikacije rađen je usputno, što isto vrijedi i za paletu boja korištenoj na aplikaciji. Primarna boja odabrana je po uzoru na primarnu boju Facebook aplikacije, nakon čega su odabrane nijanse pozadine, rubova, teksta, polja za unos te pozadine modala i *Spinner*-a. Tako se na paleti boja sa lijeve strane nalaze nijanse za svijetlu temu, a sa desne nijanse za tamnu temu.

Slika 13. Paleta boja aplikacije

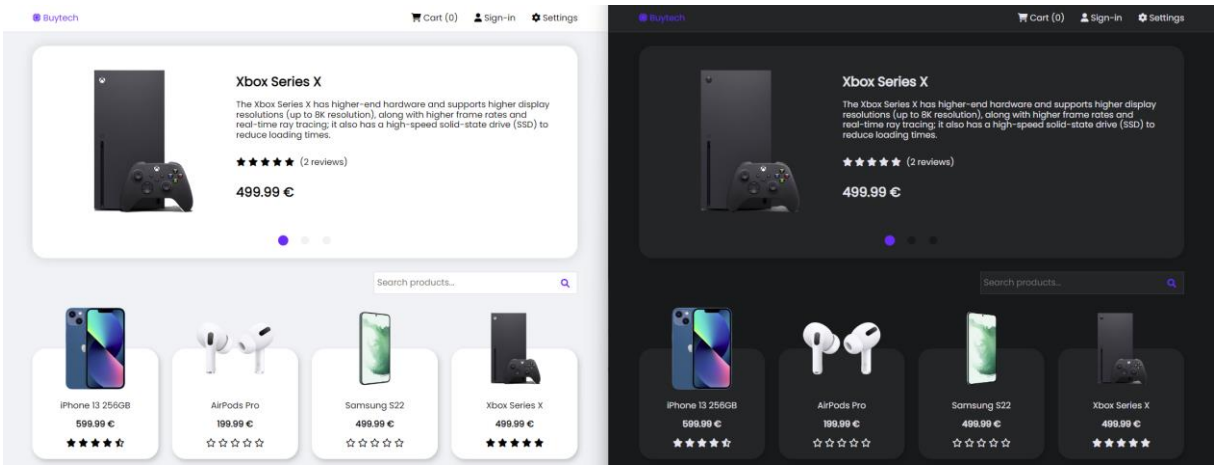
Za implementaciju tamne teme korišten je službeni Nuxt-ov dodatak po imenu *„color-mode“*. Ono što jedan takav dodatak radi u pozadini je dodjeljivanje *css* klase tijelu aplikacije u formatu *„{color}-mode“* te pamćenje posljednje odabrane teme (koristeći lokalnu pohranu). Osim mijenjanja teme, dodatak može i prepoznati temu operacijskog sustava te na temelju nje prikazati aplikaciju u odgovarajućoj boji, što je zapravo i zadana postavka. Izbornik za mijenjanje teme nalazi se u zaglavlju u padajućem izborniku postavka aplikacije.

Kako bi dodatak radio pravilno, potrebno je definirati boje u *main.css* datoteci koristeći *css* varijable. Tako se na samom vrhu spomenute datoteke može naći definicija varijabli od kojih svaka poprima neku boju. Korijenski element (:root) poprima definiciju svijetle palete boja

skupa sa definicijom primarne boje i boje teksta gumbova s obzirom da te dvije boje ostaju konzistentne tijekom mijenjanja teme. Kako je navedeno, dodatak dodaje *css* klasu na tijelo aplikacije što možemo iskoristiti za prepisivanje varijabli boja svijetle teme u tamnu temu. Bez puno zahtjevne logike i razmišljanja, implementirana je mogućnost tamne teme koja nekima čini aplikaciju pogodnijom za korištenje.

```
:root {
  --color-primary: #555cee;
  --button-text-color: #e5e5ec;

  /* light-mode */
  --bg: #f1f2f5;
  --bg-secondary: #ffffff;
  --border-color: #dedfe2;
  --font-color: #050505;
  --input-color: #f1f2f5;
  --input-text-color: #66676c;
  --loader-bg: #dedfe280;
  --calendar-color: 0;
}

.dark-mode {
  --bg: #18191a;
  --bg-secondary: #242526;
  --border-color: #393a3b;
  --font-color: #e5e5ec;
  --input-color: #3a3b3c;
  --input-text-color: #a5b2b9;
  --loader-bg: #393a3b80;
  --calendar-color: invert(1);
}
```

Primjer 18. Definicija boja



Slika 14. Prikaz aplikacije u svijetloj i tamnoj temi

5.5. Pokretanje projekta

Nakon kloniranja projekta sa *git* repozitorija, projekt nije moguće pokrenuti bez kratke instalacije. Za početak, potrebno je instalirati sve korištene dodatke pomoću *npm*-a. Ono što je potrebno odraditi je pozicionirati se u korijenski folder projekta i pokrenuti komandu *npm install* ili skraćeno *npm i*. Nakon što se instaliraju svi dodatci, potrebno se pozicionirati u *client* folder i ponoviti postupak – izvršiti *npm install*. Prvi *npm install* preuzeti će potrebne dodatke za pokretanje poslužitelja, dok će drugi napraviti istu stvar samo za klijenta. Uz instalaciju dodataka, potrebno je i dodati *.env* datoteku u korijenski direktorij sa sljedećih pet postavki u formatu „POSTAVKA = VRIJEDNOST:

- NODE_ENV – obično poprima vrijednost *„development“* ili *„production“*, a služi poslužitelju kako bi koristio logove i detaljne opise grešaka samo tijekom programiranja,
- PORT – željeni port, ako se ne specificira, poslužitelj postavlja port na 5000,
- MONGO_URI – *string* konekcije za MongoDB bazu podataka,
- JWT_SECRET – tajna za potpis JWT tokena, služi za zaštitu autorizacije,
- STRIPE_SK – tajni ključ za Stripe-ov API, može se pronaći i generirati na Stripe-ovoj stranici.

Nakon dodavanja *.env* datoteke, potrebno se je pozicionirati u korijenski direktorij projekta i pokrenuti dvije predefinirane skripte. Prva skripta je *npm run server* te pokreće poslužitelja sa *nodemon*-om dodatkom koji rekompilira i ponovno pokreće server prilikom svake promjene, a druga skripta koja pokreće klijenta glasi *npm run client*. U ovom trenutku klijentu se može pristupiti koristeći bilo koji internetski preglednik na adresi http://localhost:3000, dok je poslužitelj aktivan na adresi http://localhost:5000.

Pripremanje za produkciju izgledalo bi nešto drugačije. Za produkciju potrebno bi bilo za početak izgraditi aplikaciju. I poslužitelj i klijent imaju svoje *build* skripte koje glase *npm run build* nakon čega se mogu optimizirane verzije aplikacija pokrenuti preko skripte *npm run start*. Sljedeći korak bio bi pokretanje optimiziranih aplikacija preko servisa poput PM2 koji je jedan od najpopularnijih alata za pokretanje JavaScript aplikacija na razini udaljenog servera te postavljanja domene i popratnih stvari koje se vežu uz *dev-ops* dio.

6. Zaključak

Nakon dubljeg proučavanja teorijskog dijela i izrade praktičnog dijela aplikacije, shvatio sam koliko su neke stvari prilikom izrade web aplikacija stvarno bitne. Kada aplikacija ne bi zahtijevala nikakav SEO (recimo aplikacija za internu upotrebu), SSR i slične stvari koje jedan Nuxt nudi ne bi bile potrebne. No u slučaju web trgovine i sličnih stranica, SSR u smislu aplikacija na jednoj stranici stvarno dolazi do izražaja.

S druge strane, REST ima jako skalabilan dizajn i može se koristiti na više načina, dok Express koji stvara jedan takav servis ima poprilično intuitivan način izrade istog. Ono što je najbitnije, uz pravilnu upotrebu dodataka, može se postići sigurna aplikacija koja obrađuje zadovoljavajući broj zahtijeva u sekundi. Iako postoje brže i bolje alternative Node.js-u i Express-u, smatram da su njih dvoje za ovakve aplikacije sasvim dovoljni. MongoDB za razliku od SQL jezika pojednostavljuje komunikaciju sa bazom eliminirajući pisanje upita koji mogu biti kompleksni, no smatram da je njegova implementacija ne toliko adekvatna kao SQL na razini aplikacije koja ima više stotina tablica.

Iako danas postoji nekoliko popularnih servisa za izradu web trgovine bez znanja programiranja, smatram da je prilagođeno rješenje koje je napravljeno od nule bolji, ali i skuplji izbor. Prilagođeno rješenje nudi stopostotnu opciju prilagodbe svakog dijela aplikacije što u nekim slučajevima može biti potrebno, a s obzirom da postoji poslužitelj koji obrađuje zahtjeve, izrada mobilne aplikacije (uz postojeću web aplikaciju) ne bi predstavljala preveliki problem – isto vrijedi i za bilo koju drugu vrstu aplikacije jer se putanje na poslužitelju mogu pozvati sa bilo koje platforme što bi bio prvi korak u nadogradnji napravljene aplikacije.

Popis literature

[1] C. Győrödi, R. Győrödi, G. Pecherle, A. Olah, „A comparative study: MongoDB vs. MySQL“ (2015.) [Na internetu] Dostupno na: https://ieeexplore.ieee.org/abstract/document/7158433

[2] R. Archer, „Express.js: Web App Development with Node.js Framework“ (14.4.2016.) [Na internetu] Dostupno na: https://dl.acm.org/doi/abs/10.5555/3035728

[3] J. McGarvie, „Vue.js: The Documentary“ (24.2.2020.) [Na internetu] Dostupno na:

[4] „What is Vue?“ (bez dat.) [Na internetu] Dostupno na: https://vuejs.org/

[5] D. Svjetličić, „Nuxt.js over Vue.js: when should you use it and why“ (18.10.2019.) [Na internetu] Dostupno na: https://www.bornfight.com/blog/nuxt-js-over-vue-js-when-should-you-use-it-and-why/

[6] „Node.js Introduction“ (bez dat.) [Na internetu] Dostupno na: https://www.w3schools.com/nodejs/nodejs_intro.asp

[7] „An overview of HTTP“ (bez dat.) [Na internetu] Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[8] „HTTP/1.x vs HTTP/2 – The Difference Between the Two Procols Explained“ (bez dat.) [Na internetu] Dostupno na: https://cheapsslsecurity.com/p/http2-vs-http1/

[9] L. Richardson, S. Ruby, „RESTful web services.“ (2008.) [Na internetu] Dostupno na: https://res.infoq.com/articles/richardson-ruby-restful-ws/en/resources/04.pdf

[10] Codecademy Team, „What is REST?“ (bez dat.) [Na internetu] Dostupno na: https://www.codecademy.com/article/what-is-rest

[11] E. Saks „JavaScript frameworks: Angular vs React vs Vue“ (2019.) [Na internetu] Dostupno na: https://www.theseus.fi/bitstream/handle/10024/261970/Thesis-Elar-Saks.pdf?sequence=2

[12] D. Abramov, „React v16.8: The One With Hooks“ (6.2.2019.) [Na internetu] Dostupno na: https://reactjs.org/blog/2019/02/06/react-v16.8.0.html

[13] M. Astley, D. C. Sturman, G. A. Agha „Middleware“ (2001.) [Na internetu] Dostupno na: http://osl.cs.illinois.edu/media/papers/astley-2001-cacm-customizable_middleware_for_modular_distributed_software.pdf

[14] A. Comsian, „Express File Upload with Multer in Node.js“ (24.9.2019.) [Na internetu] Dostupno na: https://attacomsian.com/blog/express-file-upload-multer

[15] „Stripe Checkout“ (bez dat.) [Na internetu] Dostupno na: https://stripe.com/docs/payments/checkout

Popis slika

Popis primjera programskih kodova