Razvoj web aplikacije za internetsku trgovinu

Percan, Patrik

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za internetsku trgovinu

Rijeka, srpanj 2023.

Percan Patrik

0069088230

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za internetsku trgovinu

Mentor: Doc. dr. sc. Marko Gulić

Rijeka, srpanj 2023.

Percan Patrik

0069088230

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 8. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Patrik Percan (0069088230)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Razvoj web aplikacije za internetsku trgovinu / Development of a web application for web store**

Opis zadatka:

Razviti web aplikaciju za internetsku trgovinu. Aplikacija treba imati odvojeni administracijski dio i korisnički dio. Administrator mora imati mogućnost dodavanja proizvoda i informacija o njima. Također, administrator mora imati mogućnost upravljanja narudžbama. Registrirani korisnik mora imati mogućnost upisivanja svojih podataka kao i mogućnost naručivanja i kupnju odabranih proizvoda. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Next.js zajedno sa Sanity studio platformom za upravljanje sadržajem (podacima). Također, plaćanje treba implementirati pomoću Stripe platforme za procesiranje plaćanja. Za razvoj klijentskog dijela aplikacije treba koristiti React JavaScript knjižicu za razvoj korisničkog sučelja uz učinkovito renderiranje aplikacije na uređajima s različitim veličinama zaslona.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 26. ožujka 2023.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Ovim putem izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, srpanj 2023. ____________________

SADRŽAJ

1 UVOD

1.1 Motivacija

U početku samog projekta, motiv je bio učenje o razvoju internetskih aplikacija. Budući da se osnovni projekti, koji se rade u samome početku učenja, izvode u vidu nekakvih internetskih trgovina, cilj je bio to prilagoditi na malo veću razinu i ući dublje u suštinu same funkcionalnosti aplikacije. Prilikom razvoja, naglasak nije bio na oblikovanju i estetskom prikazu koji bi omogućio najbolje vizualno iskustvo, već na uspješnoj implementaciji funkcionalnosti za naplatu proizvoda. Budući da bi bilo izuzetno zahtjevno izgraditi kompleksan sustav naplate od nule, pronađen je elegantan pristup koji uključuje već postojeću platformu koja je od velike pomoći mnogim pojedincima u području e-trgovine. U ovom radu, korištena je platforma *Stripe* te će ona biti opisana i primijenjena unutar aplikacije koja je izrađena i koja se koristi kao sustav trgovanja na internetu. Sama motivacija za izradu ovoga projekta proizlazi iz rapidnoga razvoja tržišta koji se odvija u području razvoja internetskih aplikacija. Za vrijeme COVID-19 pandemije popularnost internetskoga trgovanja nikada nije bila veća, što je prisililo brojne trgovce na alternativna postavljanja svoga proizvoda na tržište. Svijet se sve više okreće virtualnome korištenju svakodnevnih ljudskih obveza, a to posebno možemo uočiti na području trgovanja. Tim putem, lokalni proizvodi postaju lako dostupni na globalnoj razini gdje nije potrebno voditi previše računa o samoj nabavi i dostavi istoga. Prednosti je trgovcima kod ovakvoga vođenja svoje e-trgovine mobilnost i neovisnost o drugim faktorima osim proizvoda i same aplikacije. Oni postaju mobilni, a ujedno i usredotočeni i motivirani na vlastiti razvoj čime se dobiva na boljoj organizaciji vremena provedenoj za prodaju proizvoda.

1.2 O samim tehnologijama

Za razvoj je klijentske strane upotrijebljena React, *open-source* knjižnica JavaScript jezika. React koristi arhitekturu zasnovanu na komponentama, što omogućava razvojnim programerima da izgrade kompleksna korisnička sučelja tako što ih razbijaju na manje, ponovno upotrebljive komponente. Ovaj modularni pristup potiče ponovnu upotrebljivost koda, što čini razvoj efikasnijim i održivim. React također slijedi obrazac jednosmjernoga protoka podataka, poznat i kao *Flux* arhitektura. Taj obrazac olakšava razumijevanje i upravljanje protokom podataka unutar aplikacije. Enkapsulirajući podatke u jednosmjerni protok, React pomaže programerima održavati predvidljivo stanje i olakšava ispravljanje pogrešaka i testiranje. Ujedno, React implementira virtualni DOM (eng. *Document Object Model*), koji je lagana kopija stvarnoga DOM-a. To omogućava Reactu da izvodi efikasna ažuriranja i renderiranje komponenata tako što minimizira izravno manipuliranje stvarnim DOM-om. Kao rezultat, React aplikacije imaju bolju izvedbu i brže renderiranje. Budući da je tema ovoga rada internetska trgovina u kojoj se očekuje varijacija proizvoda, trebalo je pronaći efikasan i brz način na koji se može manipulirati proizvodima koji će se koristiti putem aplikacije. Iz toga je razloga korišten Sanity studio, usluga koja omogućuje kreiranje predložaka samih proizvoda na poslužiteljskoj strani. Sanity Studio pruža visoko prilagodljiv sustav za upravljanje sadržajem (eng. *Content management system*) koji omogućava razvojnim programerima da definiraju i strukturiraju sadržaj prema svojim specifičnim potrebama. Ta fleksibilnost omogućuje stvaranje prilagođenih radnih tokova i osigurava besprijekorno iskustvo upravljanja sadržajem. Sanity Studio nudi značajke suradnje u stvarnome vremenu koje omogućava više korisnika da istovremeno rade na stvaranju i uređivanju sadržaja. To potiče timsku suradnju, poboljšava produktivnost i optimizira proces stvaranja sadržaja, posebno za udaljene timove. Posljednja je komponenta web aplikacije implementirana uvođenjem Stripea, alata koji će omogućiti prethodno navedena internetska plaćanja, kao i vođenje evidencije transakcija i narudžbi. S obzirom na to da je potrebno koristiti poslužiteljsku stranu unutar aplikacije kako bi se primali određeni podaci koji se šalju sa Sanity studija i šalju određeni podaci na Stripe sučelje, iskorišten je NextJS. Isti je uzet u obzir zbog njegovih optimiziranih performansi za razliku od ostalih platformi koje se koriste u istu svrhu.

2 TEHNOLOGIJE

2.1 Sanity studio

Sanity studio [1] je moćna, korisnički prilagođena platforma za upravljanje sadržajem koja pruža širok spektar prednosti kod kreiranja i upravljanja sadržajem. To je korisničko sučelje koje je smješteno „u oblaku“, što znači da za njegovu održivost i korištenje nije potrebno imati lokalnu memoriju kako bi ista normalno mogla obavljati svoju funkciju unutar aplikacije. Najčešće je korišten za upravljanje većom količinom blogova, popisom proizvoda ili portfelja. Krase ga karakteristike robusnosti kod suradnje više članova koji istovremeno rade na određenim projektima. Isti se vrlo lako integrira u razna klijentska okruženja u kojima se razvijaju same aplikacije. Kako mu je primarna zadaća upravljanje sadržajem, implementirane su *povuci i spusti* (eng. *drag and drop*) funkcionalnosti koje su popraćene jednostavnim korisničkim sučeljem primjene same aplikacije.

Cijeli projekt započinje instalacijom samoga Sanity studija. Kao što je prije navedeno, samo sučelje uvelike olakšava dinamičku promjenu dijelova proizvoda koji će se koristiti. Analogno tomu prikaz i promjena različitih dijelova proizvoda kao što su slike, imena i cijene bit će uvelike olakšano naspram ostalih tehnologija. Instalacija se izvodi preko *dependency* dijela koda unutar same aplikacije [2, 3]. Jednostavnom pretragom na stranicama Sanity studija, lako se pronalazi i detaljan postupak koji se mora odraditi preko komandne linije kako bi se završio instalacijski proces. Potrebno je posebno obratiti pažnju na verzije unutar *dependency* dijela u *package.json* datoteci u tome slučaju. Zbog čestih većih nadogradnji samoga programskog sučelja Sanity studija zna se dogoditi da isti neće raditi kako je predviđeno u dokumentaciji ili uopće ne želi otvoriti određene funkcionalnosti. Te se verzije programa također jednostavno preuzimaju jednom naredbom preko komandne linije.

Nakon instalacije unutar projekta stvara se zasebni direktorij (grana) datoteka koje su potrebne za upravljanje cijelom poslužiteljskom stranom Sanity studija. Modeliranje poslužiteljske strane započinje kreiranjem shema koje će biti potrebne u daljnjoj obradi željenoga sadržaja. Taj sadržaj može varirati prema projektu, a lako se modelira preko određenih linija koda unutar datoteke na putanji

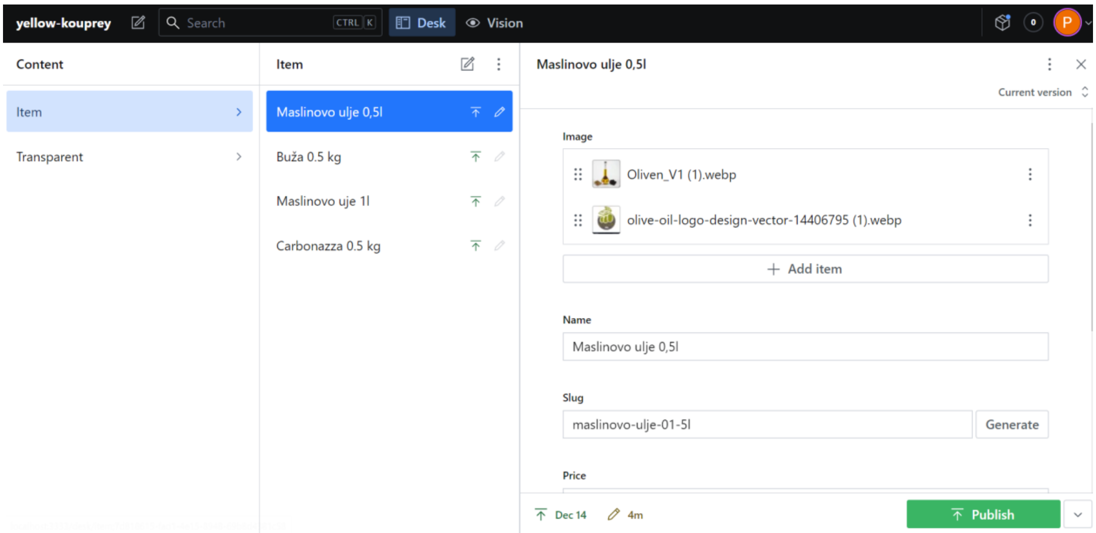
„projekt/sanity_lokalni_projekt_folder/schemas/datoteka_sadržaja“

Ta datoteka vraća običan JavaScript objekt, što će može vidjeti u *kodu* 2.1. On se sastoji od polja karakteristika koje opisuju svaki taj sadržaj koji se koristi. Karakterističan kod koji se može modificirati po potrebi prikazan je i u samoj Sanity dokumentaciji [4, 5]. Unutar koda korištene karakteristike jesu: ime, slika, cijena, detalji te *slug*. Na slici 2.1. prikazan je *slug* koji predstavlja jedinstvenu šifru za svaki proizvod koji se koristi. Šifra će se generirati po imenu proizvoda u kasnijoj implementaciji.

```
export default{
    name: 'item',
    title: 'Item',
    type: 'document',
    fields: [
        {
            name: 'image',
            title: 'Image',
            type: 'array',
            of: [{type: 'image'}],
            options: {
                hotspot: true,
            }
        }, ...
```

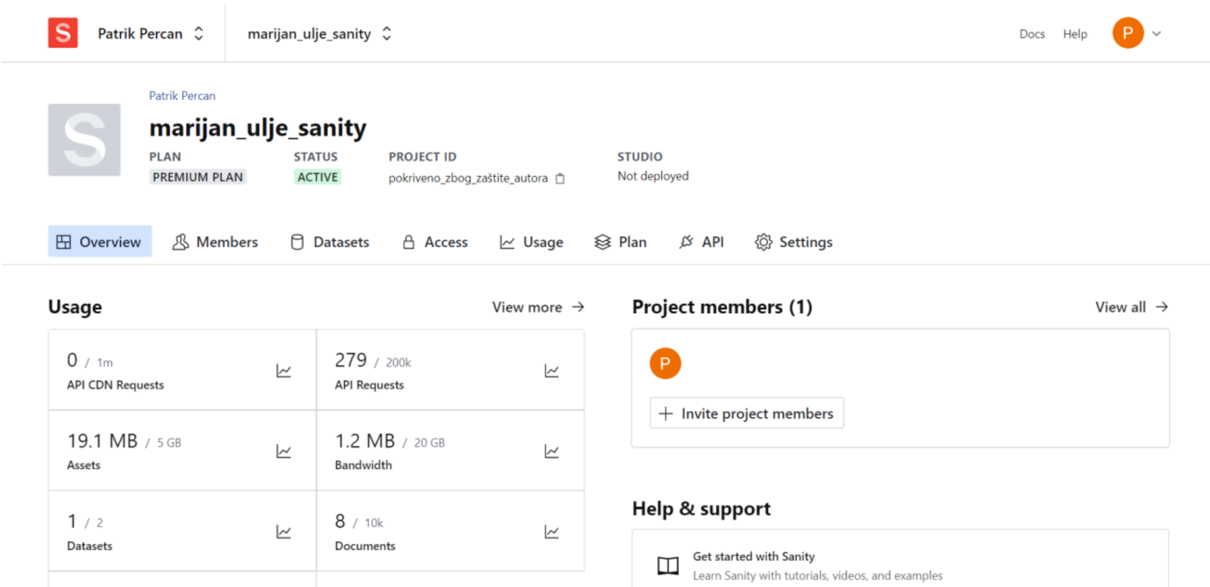
Kod 2.1. – Izrada sheme u Sanity studiju

Nakon izrade gore navedenih parametara za postavljanje shema, iste je potrebno dodati u *shema.js* datoteku koja se nalazi također na gore navedenoj putanji. To je preduvjet za prikaz shema na poslužiteljskoj strani Sanityja. Iako se ovdje govori o poslužiteljskoj strani, dodavanje sadržaja shema nalazi se na adresi *localhost:3333*. U tome trenutku prethodno izrađene sheme još nisu povezane u projekt koji se izrađuje. Sanity se koristi s dvije komande. Za pokretanje se koristi *sanity start* koji otvara *localhost:3333* te *sanity manage* preko koje postoji globalni pregled nad svim projektima i informacijama koje se koriste za povezivanje. Naredbom za pokretanje samoga Sanity studija preko komandne linije, postoji mogućnost izmjene i podešavanja sadržaja izrađenih shema na adresi *localhost:3333.* Isto se može vidjeti na *slici 2.2.*, gdje su prikazani svi parametri koji su postavljeni putem koda. U tome trenutku do izražaja dolaze *povuci-i-spusti* funkcionalnosti i jednostavnost korištenja ove programske podrške.



Slika 2.1 – Primjer jednostavnoga manipuliranja Sanity shemama

Na slici se jasno može provjeriti gdje je koji dio koda koji je napisan i kreiran. Sam „*Item*" kao globalna shema je kreirana s lijeve strane i predstavlja svaki mogući proizvod koji je moguće modelirati. Njegovim otvaranjem s lijeve strane mogu se uočiti i svi parametri koji su navedeni za karakteriziranje proizvoda. Tu je i *slug* koji će predstavljati identifikator svakoga proizvoda. On se može upisivati ručno, po volji programera ili biti ručno generiran ovisno o zauzeću i imenu koje proizvod posjeduje. Sami se proizvodi dodaju u sredini samoga prikaza, pritiskom na kvadratastu ikonu precrtanu olovčicom.



Slika 2.2. – Povezivanje projekta s poslužiteljskom stranom Sanity studija

Sljedeće što je potrebno napraviti je povezati prije izrađene sheme s projektom koji je u fazi izrade. Sanity daje detaljna objašnjenja na svojim stranicama te se isto vrlo lako odrađuje. Na putanji

„projekt/lib/client.js"

potrebno je izraditi datoteku *client.js* koja će služiti za prije navedeno povezivanje projekta i izrađenih shema. Unutar te datoteke treba napraviti funkciju koja prima jedan objekt kao parametar. Taj će se objekt sastojati od pet parametara potrebnih za komunikaciju sa sučeljem na poslužiteljskoj strani. Parametri su: *projectID, dataset, apiVersion, useCdn i token*. Većina istih pronalazi se pokretanjem *sanity manage* naredbe nakon koje se otvara globalni pregled projekta i sve informacije koje su potrebne za povezivanje projekta, što je prikazano na *slici 2.3*. Ranije spomenuta mobilnost iste platforme tu dolazi do izražaja iz razloga što se u sam Sanity studio prijavljuje s korisničkim profilom unutar kojega je sve spremno programeru za korištenje. Ispisom ovoga koda, povlače se sve informacije s naših shema i ne moramo se brinuti o daljnjim postavljanjima na poslužiteljskoj strani. *Kod 2.2.* modeliran je po dokumentacijskome dijelu koji je kreiran od kreatora Sanity studija [6].

```
import sanityClient from '@sanity/client';
import imageUrlBuilder from '@sanity/image-url';

export const client = sanityClient({


    projectId: 'ovdje_ide_projectID',
    dataset: 'production',
    apiVersion: '2022-10-11',
    useCnd: true,
    token: 'ovdje_ide_generirani_token'
}
);

const builder = imageUrlBuilder(client);
export const urlFor = (source) => builder.image(source);
```

Kod 2.2. – Povezivanje projekta s poslužiteljskom stranom Sanity studija

2.2 NextJS

Jedan je od popularnijih JavaScript okvira za povezivanje s poslužiteljskom stranom *NextJS* [7]. Za njegovo se korištenje najprije treba konfigurirati *NodeJS* na radnome računalu te ponovno preko određenih paketa preuzeti sam *NextJS*. Taj je proces vrlo jednostavno napraviti s nekoliko linija koda unutar terminala za pisanje naredbi.

NextJS pruža značajke kao što su *brzo osvježavanje* (eng. *hot-reloading*), brzo i točno izvješće o pogreškama te mogućnost generiranja prikaza na poslužitelju (eng. *server-side-rendering*) što je i razlog za odabir te tehnologije. Budući da je bilo pitanje kako i kojom brzinom će se učitavati prije objašnjeni podaci sa Sanity studija, značajka *server-side rendering*-a daje u ovome slučaju odgovor na pitanje brzine i efikasnosti rada aplikacije u različitim okruženjima.

Server-side-rendering je tehnika gdje poslužitelj generira HTML dokument iz određenoga koda i šalje ga direktno pregledniku. Ne oslanja se na preglednik da generira isti HTML dokument korištenjem JavaScript jezika. U NextJS-u, *server-side rendering* je ugrađen od početka korištenja te se korisnik ne mora brinuti o tome dijelu implementacije. Kada poslužitelj radi generiranje stranice, šalje pregledniku cjelokupni kod u formatu HTML-a, CSS-a te JavaScripta koji je spreman za korištenje. Iako danas postoje brze mreže na kojima se razvija protok informacija, treba uzeti u obzir da postoji nekolicina korisnika koji ne mogu pristupiti „bržim" mrežama. Time se nastoji postići optimalnost u komunikaciji aplikacije i korisnika. Ta tehnologija uvelike pomaže kod prikaza stranice s mnogo komponenti i logike koja se odvija u pozadini, a rasprostranjena je preko više stranica koje zajedno čine internetsku aplikaciju.

Nakon odrađene instalacije, dovoljno je pokrenuti naredbu *npx create-next-app@latest* gdje će se napraviti cijela grana datoteka unutar koje se može početi razvijati sama aplikacija. NextJS nadalje neće biti potreban, samo za još nekoliko funkcionalnosti.

S obzirom na to da projekt nije baziran samo u React programskoj podršci i težnja je ka brzini prikaza same aplikacije, ne mogu se koristiti funkcije kao što su *useEffect( )* preko kojih se primaju podaci s poslužiteljske strane. U tu se svrhu koristi NextJS funkcija *getServerSideProps( )*. Ona omogućava prije spomenuti *server-side rendering* i smanjuje vrijeme učitavanja aplikacije. Treba napomenuti da je ova funkcija asinkrona funkcija pa se tako treba i definirati unutar koda. Iz istoga razloga, prilikom prihvaćanja parametara proizvoda, potrebno je upotrijebiti ključnu riječ *await* [8, 9], kako bi ista uvijek bila spremna na prihvat novih informacija kada ih poslužitelj pošalje. Gore navedene funkcionalnosti koda mogu se pogledati na *kodu 2.3*. Unutar funkcije se radi upit prema poslužiteljskoj strani koja vraća sve informacije o svim proizvodima koji su prethodno definirani. Iste se te informacije moraju pohraniti u određenu varijablu. Sanity studio vraća polje koje se sastoji od definiranih varijabli potrebnih za opis proizvoda. Budući da kod Reacta nema potrebe za znanjem kakav je tip varijable, sve se sprema u varijablu *items* gdje se nalaze svi vraćeni podaci.

```
export const getServerSideProps = async () => {
  const query = '*[_type == "item"]';
  const items = await client.fetch(query);

  return {
    props: { items }
  }
}
export default homePage
```

Kod 2.3 – Zahtjev za pribavljanje svih proizvoda

Taj će proces biti odrađen na svim onim mjestima unutar aplikacije gdje je potrebno zatražiti podataka s poslužiteljske strane. U tome slučaju, sličan će kod biti napisan kod primanja podataka o pojedinačnome proizvodu kada se istome pristupa.

Kod izrade stranice gdje će se prikazivati sam proizvod nakon odabira istoga, potrebno je malo izmijeniti funkciju (*kod 2.3*).

NextJS sadrži usluge koje se zovu *file based routing* [10]. Analogno tomu u NextJS-u se može iskoristiti ime datoteke kao i njezin identifikator na sljedeći način. S obzirom na to da identifikatori moraju biti unikatni kako bi se uvijek izabrao ispravan proizvod, nije poželjno raditi za svaki proizvod posebnu datoteku. Tu pomaže funkcionalnost NextJS-a gdje se ime datoteke proizvoda (koje će biti vizualno isto za sve proizvode na krajnjemu prikazu) zamjenjuje tim identifikatorom proizvoda. To se postiže na način da se ime datoteke postavi u *slug* koji se prije spominjao te se isti „zamota“ unutar uglatih zagrada. To znači da će samo ime ove datoteke biti dinamično i da će se koristiti u svakom slučaju kada se radi o populaciji te datoteke određenim proizvodima. U tome je projektu ta datoteka smještena na putanji:

projekt/pages/items/[slug].js

Time se štedi na vremenu i na mogućnosti dolaska do pogreške kod generiranja većega broja proizvoda pisanjem samoga koda.

Dohvaćanje će podataka trebati ponovno napraviti prilikom generiranja stranice u kojoj se prikazuje određeni proizvod. U tome se slučaju može ponovno koristiti *getServerSideProps( )* funkcija, no ako se traži određena brzina u izvršavanju i generiranju stranica, istu se neće iskoristiti. Pošto su parametri koji su već vraćeni s poslužitelja zapisani u jednu varijablu, iskoristit će se sadržaj iz te varijable. Unutar React komponente koja će generirati stranicu proizvoda, obavezno se uvoze dvije stavke koje su prethodno definirane. Te su dvije stavke *client* i *urlFor*. To se radi iz razloga što će se tako napraviti API poziv koji će povući sve potrebne parametre proizvoda i ispisati ih na stranicu.

```

import {client, urlFor} from '../../lib/client';
//ostatak koda koji je prikriven

 export const getStaticProps = async ({ params: { slug }}) => {
   const query = `*[_type == "item" && slug.current == '${slug}'][0]`;
   const itemsQuery = '*[_type == "item"]'

   const item = await client.fetch(query);
   const items = await client.fetch(itemsQuery);

   return {
     props: { items, item }
   }
 }
```

Kod 2.4. – Ponovno dohvaćanje parametara na stranici pojedinačnoga proizvoda

Kao što je prije rečeno, *getServerSideProps( )* se zamjenjuje s *getStaticProps( )* [11]. Razlika je u funkcijama ta da će *getStaticProps( )* iskoristiti da se stranice proizvoda tako generiraju prije korisnikovoga zahtjeva čime se dobiva na brzini i dinamičnosti kod pregledavanja. Posebno je bitno naglasiti da se u samoj funkciji može razdijeliti parametre cijeloga proizvoda na pojedinačne. U tome će slučaju biti potreban samo identifikator proizvoda kako bi se mogao napraviti zahtjev prema poslužitelju da vrati samo određeni proizvod. Iz toga se razloga koriste dvije varijable; *item* u koji se sprema samo traženi proizvod i *items* gdje će se spremati svi proizvodi za kasniju upotrebu. *Items* varijabla na samome kraju nije iskorištena u tome kodu, a trebala je služiti za pohranjivanje informacija o ostalim proizvodima kako bi se sa stranice pojedinačnoga proizvoda moglo prijeći direktno na nove proizvode. Iznad opisane funkcionalnosti koda mogu se uočiti na *kodu 2.4.* Bilo je zamišljeno da to izgleda kao padajući izbornik s jedne strane stranice pojedinačnoga proizvoda ili kao reklama koja prikazuje proizvode u prostoru ispod traženoga proizvoda.

2.3 React

React [12], razvojna knjižnica koja se koristi u okviru JavaScript jezika, jedna je od najčešće korištenih knjižnica za razvoj web aplikacije na klijentskoj strani. Kod instalacije, nude se brojne druge ekstenzije same knjižnice koje uvelike olakšavaju razvoj i pisanje koda.

Prednosti su korištenja Reacta brojne. U početku je potrebno spomenuti lakši razvoj dinamičnih dijelova same aplikacije, gdje se s manje linija koda postiže bolji vizualni efekt na samoj aplikaciji. Kod se piše u okviru logički povezanih komponenti, koje omogućuju bolju i lakšu iskoristivost u određenim situacijama (primjer: lakše pronalaženje greške na višemu nivou razvoja). Posebno treba napomenuti lakoću učenja istoga, jer se učenje sastoji od znanja HTML-a i osnovnih znanja JavaScripta. Zbog otvorenoga izvora koda (eng. *open-source*) licence, također je prisutan i velik broj modula koji omogućuju lakše pronalaženje greški u kodu. React se može primijeniti na više razvojnih platformi, a uz to daje jako brze odzive u grafičkome prikazu komponenti koje se samostalno izrađuju.

Vizualni dio internetske aplikacije počinje izradom samih React komponenti koje će se koristiti za prikaz globalnih cjelina u aplikaciji. Svaki projekt započinje modificiranjem sadržaja koji će se prikazivati na početnoj, *index.js* stranici. Istoj se pristupa po putanji:

projekt/pages/index.js

Sve stranice koje se žele prikazati kao zasebne pa tako i prethodno opisana stranica proizvoda, moraju se postaviti u ovaj direktorij. Težnja je unutar ovoga projekta bila izraditi minimalističku stranicu i posvetiti se samome plaćanju na kraju. Tako je na početnoj stranici postavljena slika koja predstavlja korisniku samu temu aplikacije i čemu ona zapravo služi. Za to je područje također kreiran React fragment, koji je u početku bio predviđen da umjesto slike prikazuje video uradak. Za prikaz proizvoda također će se koristiti React fragment koji omogućuje prikaz svih proizvoda koji se pošalju s poslužitelja na vrlo efikasan način. Nakon stiliziranja i dobivanja željene pozicije prikaza ovih fragmenata upotrebljavamo funkciju „*reactFragment.map( )*“ gdje je taj fragment u tome slučaju sam proizvod. U *Reactu, map( )* funkcija je ugrađena metoda JavaScriptovog niza koja se često koristi za iteraciju kroz elemente niza i generiranje nove verzije niza s promijenjenim ili transformiranim vrijednostima. Ovo je posebno korisno kada se želi prikazati dinamička lista elemenata ili kada se generira dinamički sadržaj na temelju dostupnih podataka. Sama funkcija vraća parametre svih proizvoda koji se dobiju sa Sanity studija. U tome je slučaju potrebno povući identifikator proizvoda i cijeli njegov objekt kako bi isti mogli točno prikazati.

U tome trenutku izrade, sam fragment koji predstavlja proizvod još nije izrađen. Postoje još brojni fragmenti koji čine strukturu cijele aplikacije koji su povezani u jednu funkcionalnu cjelinu. Svi se fragmenti smještaju na putanju:

projekt/components

Tako se zadržava uredna i strukturirana organizacija samoga direktorija projekta. Ti fragmenti izrađuju se i postavljaju unutar stranice po želji korisnika. U tome je slučaju izrađeno više fragmenata, no par ključnih za izvođenje cijele logike aplikacije su: *Item* koji predstavlja samu strukturu i prikaz zasebnoga proizvoda, *Layout* koji predstavlja strukturu početne stranice, *introTransparent* za prikaz uvodne slike/video sadržaja te *Cart* za prikaz i logiku koja se prikazuje u samoj košarici. U *kodu 2.5.* prikazana je potpuna izrada fragmenta koja se izrađuje po želji korisnika aplikacije. Sve izrađene fragmenmte potrebno je samo postaviti u zajednički direktorij kako bi se održala kontinuiranost i urednost pisanja programskog koda, te kako bi sam prevodioc znao od kuda čitati iste fragmente.

```
import React from 'react';
import Link from 'next/link';
import {urlFor} from '../lib/client';

function Item({item: {image, name, slug, price}}) {
  return (
    <><div>
      <Link href={`/products/${slug.current}`}>
        <div className="product-card">
          <img src={urlFor(image && image[0])} width={250} height={250} className="product-
          <p className="product-name">{name}</p>
          <p className="product-price">{price} kn </p>
          <p className="product-price">{parseFloat((price) / 7.5354).toFixed(2)} EUR </p>

        </div>
      </Link>
    </div>
    </>
  )
}

export default Item
```

Kod 2.5. – Prikaz cijelog „Item" fragmenta u programskom kodu

Za prebacivanje logike sa stranice na stranicu koristit će se kuke, (eng. *React Hook API references*). Kuke će nam uvelike pomoći da se izbjegne izrada pojedinačnih klasa za spremanje takvih vrijednosti. Prije nego je uvedena podrška za same kuke, praćenje stanja se odrađivalo preko klasa i pozivanja funkcija kao što su *this.setState()* koje su spajale stara i nova stanja u novo stanje. React Hook API je definiran s poljem u koje se spremaju dvije vrijednosti i tako prati staro i novo stanje. Prva je varijabla sadašnja vrijednost koja se prebacuje, dok je druga vrijednost funkcija preko koje se modificira prva vrijednost iz polja. Potrebno je posebno obratiti pažnju da se *React Hook API* ne koristi unutar klase. Ujedno, isti se ne smiju pozivati unutar petlji ili unutar grananja. Ukoliko se ova pravila i žele prekršiti, prevoditelj će odmah prekinuti izvođenje i vratiti grešku. U tome će se projektu koristiti nekoliko takvih vrijednosti za praćenje i mijenjanje stanja unutar same aplikacije. Na dolje se navedenoj putanji može pronaći zapis korištenja kuka u izradi ovoga projekta:

projekt/context/appContext.js

Izrada praćenja stanja započinje uvozom kuka iz standardne knjižnice Reacta [13]. U tome se primjeru koriste četiri vrste kuka [14, 15]. To su *createContext*, u*seContext*, *useState* i *useEfffect*. Potrebno je ponovno izraditi React fragment koji će predstavljati sva stanja. Za izvođenje je logike i slanje određenih parametara preko stranica potrebno pratiti par parametara. To će se praćenje odnositi na to je li košarica prikazana, koji je sadržaj same košarice, ukupna cijena unutar košarice, količina proizvoda u košarici i količina pojedinoga proizvoda u košarici. Kako se unutar ovoga React fragmenta radi o logici koja se šalje preko svih stranica, povratna vrijednost vraćat će fragment *Context.Provider* unutar kojega se šalju gore navedeni parametri. Cijela se internetska aplikacija mora „zamotati" unutar fragmenta *StateContext* kako bi parametri bili pomicani sa stranice na stranicu. Zato unutar datoteke *_app.js*, sve što funkcija vraća, treba definirati unutar fragmenta *StateContext*. Isti je potrebno dodati u zaglavlju datoteke s putanje unutar koje se nalazi u direktoriju aplikacije. Vrlo je važan parametar kod korištenja *StateContext* funkcije parametar *children* koji se šalje samoj funkciji. Budući da je on poslan preko samoga *StateContext* fragmenta, on se ne prikazuje na stranici već služi kao pozadinski proces za slanje parametara. To su parametri koji će nadalje biti potrebni za „pamćenje“ vrijednosti kod prelaska sa stranice na stranicu. Parametri koji će se koristiti u tome primjeru prikazani su na *kodu* 2.6.

```
import React, {createContext, useContext, useState, useEffect} from "react";

const Context = createContext();
export const StateContext = ({ children }) => {
    const [showCart, setShowCart] = useState(false);
    const [cartItems, setCartItems] = useState([]);
    const [totalPrice, setTotalPrice] = useState(0);
    const [totalQuantities, setTotalQuantities] = useState(0);
    const [qty, setQty] = useState(1);
    //nastavak koda koji je prikriven
  return (
    <Context.Provider
      value={{
        showCart,
        setShowCart,
        cartItems,
        totalPrice,
        totalQuantities,
        qty,
        incQty,
        decQty,
        onAdd,
      }}
    >
      {children}
    </Context.Provider>
  )
```

Kod 2.6. – Definiranje i korištenje „StateContext“ funkcija

S obzirom na to da *StateContext* fragment vraća sva stanja koja se žele pratiti kroz stranice, isti je potrebno implementirati unutar svakoga fragmenta koji će se koristiti za praćenje stanja. Funkcionalnost koja je tako implementirana je da se prilikom napuštanja stranice ili pregledom stranice novoga proizvoda sve informacije koje se spremaju u košaricu ostaju vidljive i nakon izlaska iz košarice.

Kako bi se lakše dohvaćali parametri koji se šalju gore prikazanom funkcijom, radi se izvoz iste koja radi preko funkcije *useContext( )* kojoj se prosljeđuje parametar samoga konteksta aplikacije koji se prati. Daljnje se korištenje svodi na destrukturiranje parametara koji se šalju preko iste funkcije i korištenja istih u samim dijelovima koda. Prikaz će primjera destruktutriranja biti prikazan na funkciji koja je definirana unutar same „*StateContext*“ datoteke (*kod 2.7*), a odnosi se na jednostavan primjer inkrementiranja i dekrementiranja količine proizvoda koji se prati preko svih stranica.

```
const [index, setIndex] = useState(0);
const {decQty, incQty, qty, onAdd} = useStateContext();
//ostatak prikrivenog koda

                        <span className = 'minus' onClick = {decQty}><AiOutlineMinus /></span>
                        <span className = 'num' onClick = ''>{qty}</span>
                        <span className = 'plus' onClick = {incQty}><AiOutlinePlus /></span>
```

Kod 2.7. – Destrukturiranje i korištenje „state“ funkcija

2.4 Stripe

Stripe [16] je platforma za procesiranje plaćanja koja omogućuje tvrtkama da prihvate i upravljaju online plaćanjima. Osnovana 2010. godine, platforma je brzo postala jedno od najpopularnijih rješenja za plaćanje, procesuirajući milijarde dolara transakcija svake godine. Popularnost se Stripea može pripisati njegovoj jednostavnosti korištenja, naprednim značajkama i sveobuhvatnim sigurnosnim mjerama. Pomoću Stripea tvrtke mogu prihvaćati plaćanja svim glavnim debitnim i kreditnim karticama, kao i digitalnim načinima plaćanja poput Apple Payja i Google Payja. Procesiranje plaćanja putem Stripea je brzo i pouzdano, a sredstva se obično polažu na račun tvrtke u roku od dva radna dana. Dodatno, Stripe nudi niz alata za upravljanje plaćanjima, uključujući prilagodljive obrasce plaćanja, fakturiranje i pretplate. Te značajke omogućuju tvrtkama da pojednostave svoje procese plaćanja, smanje administrativni rad i ponude bolje iskustvo plaćanja svojim kupcima.

Sigurnost je vrhovni prioritet za Stripe, a platforma nudi naprednu zaštitu od prijevare i sigurno pohranjivanje. Stripe također nudi opsežnu dokumentaciju, podršku i resurse kako bi pomogao tvrtkama da započnu i optimiziraju svoje procese plaćanja. Prije samoga kretanja u korištenje istoga, moguće je provjeriti rad sustava probnom karticom kako bi se olakšalo korisniku inicijalna upotreba. Sve u svemu, Stripe je pouzdana, sigurna i jednostavna platforma za procesiranje plaćanja koja može pomoći tvrtkama svih veličina da prihvate plaćanja putem interneta. Platforma Stripe bit će detaljno objašnjena u poglavlju 4. kroz primjer plaćanja unutar web aplikacije koja je tema ovoga završnog rada.

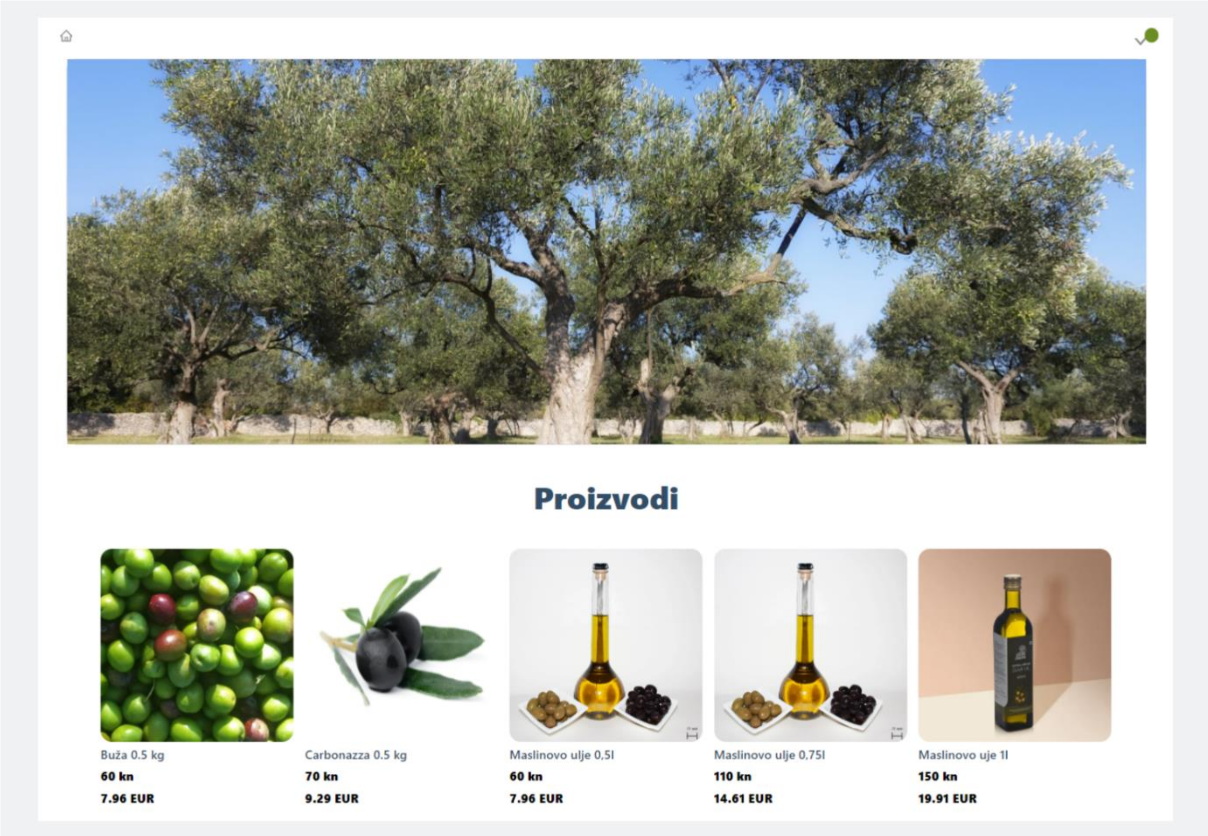
3 OPIS RADA SAME APLIKACIJE

Minimalistički je pristup i jednostavnost u radu sa samom aplikacijom bio prioritet u izradi ovoga projekta. Ujedno, treba naglasiti da je cilj pokazati kako svatko s malo znanja i mogućnosti čitanja pripadajuće dokumentacije može izraditi sličnu aplikaciju po svome principu u svoju korist. Aplikacija će u tome trenutku biti prikazana na lokalnome razvojnom poslužitelju. Iako je cilj izvesti i pokazati postupak i samoga izvoza aplikacije s vlastitom domenom, isti će biti opisan u zadnjemu poglavlju, poglavlju 5.

3.1 Početna stranica

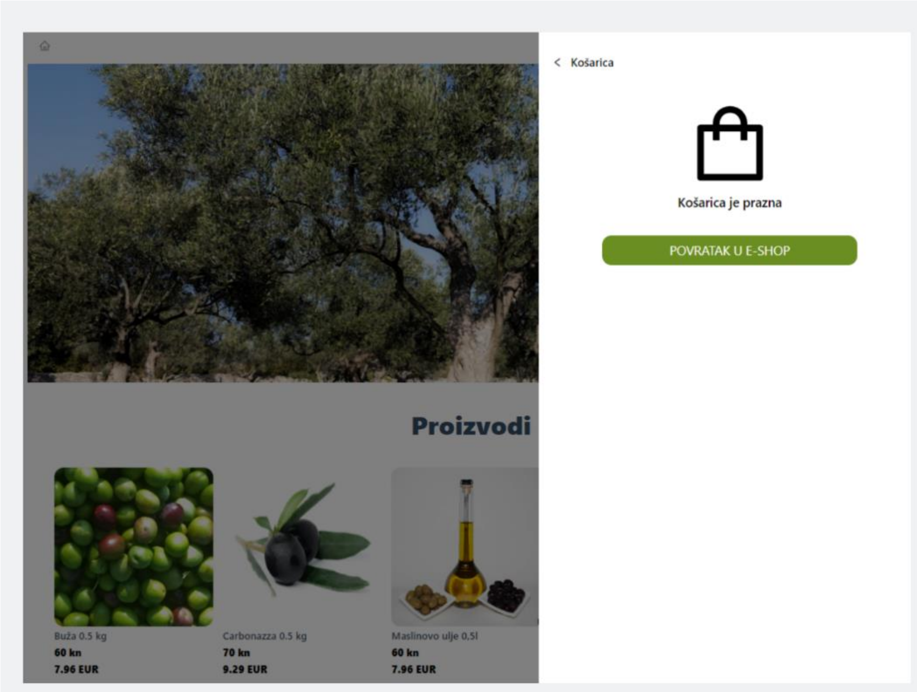
Sam se ulazak u aplikaciju ostvaruje pristupom željenoj adresi na kojoj će se aplikacija nalaziti. U tome slučaju, nakon davanja komande *npm run dev* unutar lokalnoga direktorija gdje se nalazi projekt, pokreće se lokalni razvojni poslužitelj koji služi za sam vizualni prikaz završnoga proizvoda kod razvoja. Isti se otvara na adresi *localhost:3000/.* Prvo se učitavanje odvija malo duže nego obično, no to se događa iz razloga inicijalizacije cjelokupnoga projekta i pripreme za samo učitavanje dijelova koda koji će se prikazivati na određenoj adresi.

Ulaskom u aplikaciju vidljiv je prikaz koji se prikazuje korisnicima/kupcima proizvoda (*slika 3.1.*). U inicijalnome otvaranju stranice odmah se uočava uvodna slika ili video o kojoj je bilo riječi u ranijim poglavljima. Ona predstavlja uvod u samu tematiku aplikacije. Prikazom videa, korisniku se dočarava pravi ambijent unutar kojega su proizvodi pribavljeni i prerađeni za daljnju korisnikovu upotrebu. Proizvodi upotrebljavaju neke od efekata kod pokreta i odabira kako bi se više dopali kupcima. Isti su iskazani u dvojnome iskazivanju cijena, jer je za vrijeme izrade aplikacije bilo pod obavezno prikazivati dvije cijene zbog promjene valute. Ispod samih proizvoda mogu se uočiti informacije o prodavaču; ime, prezime, adresa, kontakt podaci i poveznica za društvene mreže. Iz razloga zaštite samoga prodavača, taj scenarij neće biti prikazan u ovome radu.



Slika 3.1. – Prikaz početne stranice

Navigacija se po samoj početnoj stranici odrađuje na jednostavan, minimalistički način. U gornjemu se lijevom kutu može uočiti ikona koja vraća korisnika na početnu stranicu gdje ima na raspolaganju ponovno izabrati i pretraživati po proizvodima koji se nude u trgovini. Gornji desni kut, prikazan strelicom koja gleda prema dolje, rezerviran je za otvaranje košarice koja izgleda kao da iskače i prekriva trećinu početne stranice (*slika 3.2*). Ukoliko se u košarici nalazi barem jedan proizvod, strelica će biti djelomično prekrivena zelenim znakom koji daje naznaku da ista nije prazna. Ukoliko je košarica prazna, strelica nema vizualnu oznaku uz sebe.



Slika 3.2. – Prikaz izdvajanja košarice

3.2 Pojedinačni proizvodi

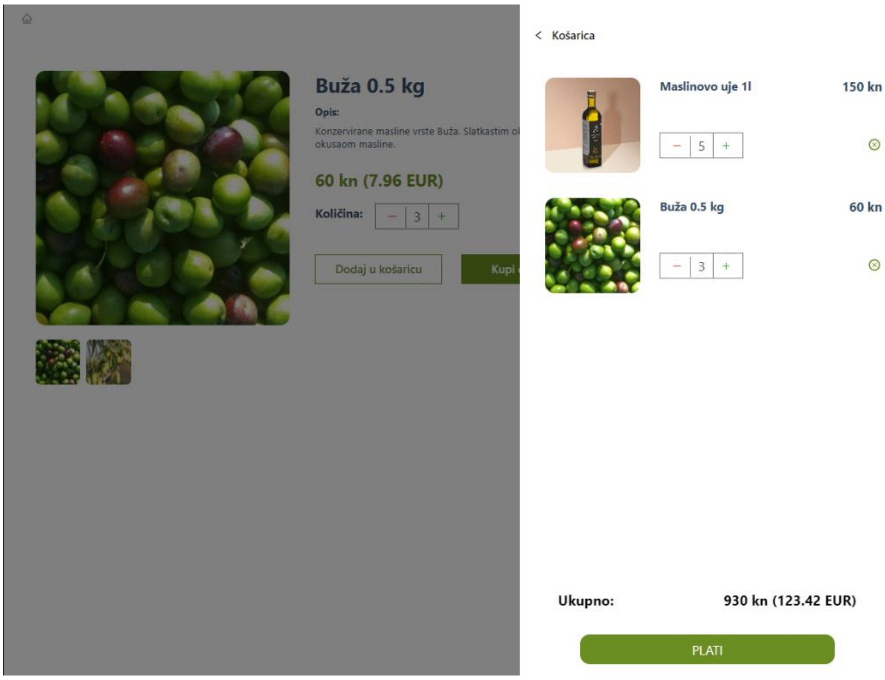
Nakon što kupac izabere određeni proizvod, otvara isti i prikazuje mu se dolje navedeni prikaz (*slika 3.3.*). U fokusu je s lijeve strane sam proizvod i sve informacije o njemu. S obzirom na to da se u većini slučajeva nastoji prikazati više slika o samome proizvodu, ispod glavne slike proizvoda prikazane su i ostale. Prelaskom miša po istima one se pomiču u gornji okvir i čine prikaz ostalih slika ljepše vidljivim. Klikom na određenu sliku, ista se povećava i moguće je koristiti značajke zumiranja po istoj. Korisnik može povećavati i smanjiti količinu koju želi prebaciti u košaricu. U tome dijelu implementacije košarice i manipulacije proizvoda, gdje se piše sam programski kod, potrebno je posebno pripaziti na logiku prema kojoj će se sve odvijati. Količina se ograničava na neku razumnu brojku, kao što je 50 u ovome slučaju te ne prelazi manju od jedan. Pritiskom na „Kupi odmah!" stranica je napravljena da odmah radi prebacivanje na domenu Stripea gdje će se izvršiti sama kupovina pojedinačnoga proizvoda. Isti će postupak kasnije detaljno biti pojašnjen. Dodavanjem u košaricu izbacuje se poruka koja pokazuje kako je proizvod uspješno dodan u košaricu na samome vrhu stranice. Količina koja se postavi kod odabira samoga proizvoda prebacuje se i u košaricu, a to se radi jednim od postupaka koji je opisan u prijašnjemu poglavlju. Pritiskom na gornji lijevi kut, na ikonu male kućice, korisnik se vraća na početnu stranicu dok su mu sve promjene po košarici spremljene za daljnje korištenje.

Slika 3.3. – Prikaz pojedinačnoga proizvoda i funkcionalnosti

3.3 Pregled i manipulacija košaricom

Odabirom jednoga ili više proizvoda te dodavanjem istih u košaricu, korisnik se ima mogućnost vratiti na početnu stranicu i nastaviti s pretraživanjem drugih proizvoda. Ukoliko je to odučio, isto se odrađuje pritiskom na ikonu kućice u gornjemu lijevom kutu. Potom se vraća na početnu stranicu gdje ponavlja identičan proces i za drugi proizvod. Odabirom drugoga proizvoda i dodavanjem u košaricu, korisnik ima mogućnost pregleda stavki koje se trenutno nalaze u istoj. To odrađuje pritiskom na strelicu u gornjemu desnom kutu koja gleda prema dolje, a označena je zelenim uzorkom. Korisniku se otvara košarica koja prekriva desnu trećinu stranice i blago zatamnjuje glavni prikaz aplikacije i scene u kojoj se korisnik nalazio kada je otvorio košaricu (*slika 3.4.*).

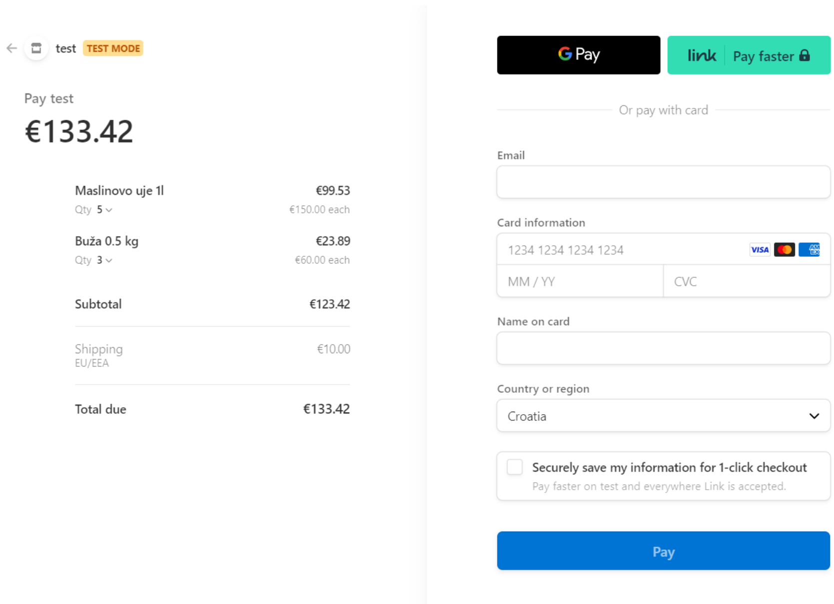
Dodavanje se proizvoda u košaricu vrši po principu reda. Prvi će dodani proizvodi biti uvijek prikazani na samome vrhu košarice, a naknadno se dodani proizvodi postavljaju ispod. Svakome se proizvodu prikazuje ime, prva slika u polju slika koje su dostupne na stranici samoga proizvoda, cijena i ponovna mogućnost upravljanja količinom prije same kupnje. Naknadno, proizvod se može i izbaciti iz same košarice pritiskom na zelenu ikonu označenu s oznakom x. Cijena se sama formira i prikazuje u obje valute prilikom svake izmjene unutar košarice. Pritiskom na samu ikonu koja je označena s „Plati“, podaci se o proizvodima šalju na Stripe poslužitelj koji će kasnije biti prikazan.



Slika 3.4. – Manipulacija proizvodima u košarici

3.4 Stripe preusmjeravanje

Praćenjem prijašnjih koraka i pritiskom na poveznicu za plaćanje svih proizvoda iz košarice dolazi do samoga preusmjeravanja na Stripe poslužitelj. Tako se nakon par sekundi učitavanja odmah mogu uočiti sve naše stavke iz trgovine na samoj Stripe usluzi (*slika 3.5.*).



Slika 3.5. – Prikaz generičnoga toka plaćanja usluge

S lijeve se strane stranice uočavaju svi detalji koji su potrebni za provjeru kupcu prije same kupnje. U usluge je također dodana i usluga slanja proizvoda koja se naplaćuje po predodređenoj tarifi. Modeliranje će te tarife biti naknadno objašnjeno.

S desne se strane upisuju podaci koji su potrebni za svaku kupnju. Adresa e-pošte bit će od iznimne važnosti jer će se preko nje vršiti sva komunikacija između kupca i prodavača nakon obavljene kupnje. Nakon unosa svih ispravnih podataka kupnja je gotova i proizvodi se mogu platiti.

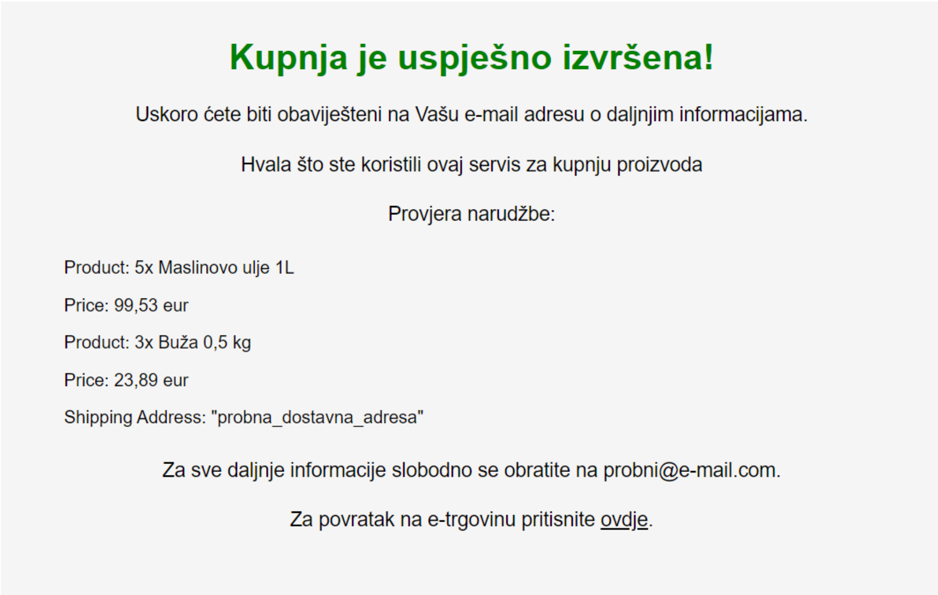
Radi sigurnosti osobnih podataka, usluga kasnije neće biti testirana s pravim podacima koji se koriste prilikom transakcija. Kao što je vidljivo sa slike, koristi se testna verzija plaćanja koja je implementirana kako bi se lakše ispitalo je li sve postavljeno kako treba. Unutar same slike nedostaje nekoliko informacija koje korisnik upisuje, kao što je adresa dostave pošiljke. To je još jedan od iznimno važnih elemenata u koraku kupnje, no ovdje izostavljen radi jednostavnosti korištenja testnoga načina. Budući da je ovo generički tok plaćanja koji je kreiran od Stripe sučelja, ostali su važni podaci zanemareni.

3.5 Pojedinačna kupnja proizvoda

Određeni korisnici mogu i ciljano ući u aplikaciju i opcijom „brze kupnje“ izabrati jedan proizvod za kupnju. Ovdje se radi o pristupu koji je orijentiran točno određenom broju kupaca koji imaju točnu viziju što žele naručiti. Ukoliko se takav korisnik nalazi na prikazu koji se nalazi na slici 3.3., pritiskom na zelenu poveznicu koja je označena s „Kupi odmah!“ korisnikov se odabir šalje na samu Stripe poveznicu unutar koje će se ta kupnja moći realizirati. Prilikom samoga pritiska na tu poveznicu potrebno je sačekati par sekundi kako bi došlo do inicijalizacije samoga Stripe sučelja. Proces koji se potom odvija je isti kao i onaj opisan u poglavlju 3.4.

3.6 Povratak u aplikaciju nakon kupnje

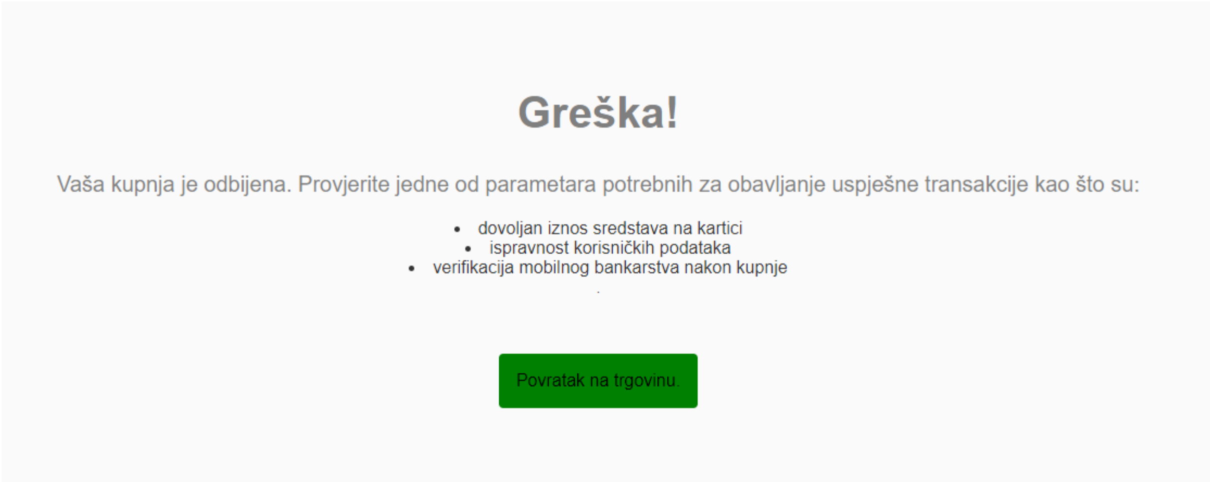
Nakon što je odrađen proces kupnje proizvoda i sam Stripe potvrdi podatke kartice te napravi nalog za transakciju, napravi se povratak na stranicu koja pokazuje sažetak cijele kupovine (*slika 3.6.*). Proizvodi i količine se ponovno ispisuju te njihove cijene kako bi kupac mogao još jednom provjeriti je li sve prošlo onako kako je predviđeno. Ukoliko dođe do neke pogreške u procesu narudžbe i kupnje, dodatno je istaknut i kontakt radi uspostavljanja komunikacije s prodavačem s ciljem što bržega rješenja problema. Na samome je dnu stranice posebno istaknuta poveznica kojom se kupac može ponovno vratiti u e-trgovinu i ponovno odraditi kupnju.



Slika 3.6. – Sažetak kupnje i povratak na e-trgovinu

3.7 Preusmjeravanje nakon neuspješne transakcije

Ukoliko dođe do slučaja gdje korisnik nema dovoljno sredstava na računu ili je preko modernih sustava verifikacije naplate koji se odvijaju mobilnim bankarstvom (*PayWay* ili sl.) odbijena mogućnost transakcije na drugi korisnikov račun, Stripe vraća poseban kod preko kojega ponovno radi preusmjeravanje na određenu stranicu kako bi obavijestio korisnika da je nešto pošlo po krivu (*slika 3.7.*). U tome se slučaju korisniku ne prikazuju elementi koje je namjeravao kupiti, već samo prikladna poruka o neuspješnoj kupovini. Kako bi se poboljšalo korisnikovo iskustvo, prikazana je poveznica kojom se korisnik može vratiti u e-trgovinu. Ukoliko je došlo do slučajne greške prilikom transakcije, korisnik se pritiskom na tu poveznicu vraća u trgovinu te su mu u košarici ponovno postavljeni proizvodi koje je prethodno namjeravao kupiti. Nakon provjere razloga koji je izbacio grešku u transakciji i ispravljanja iste s korisničke strane, otvaranjem košarice i pritiskom na poveznicu „Plati“ proces započinje ponovno.



Slika 3.7. – Obavještavanje nakon neuspjele transakcije

4 STRIPE

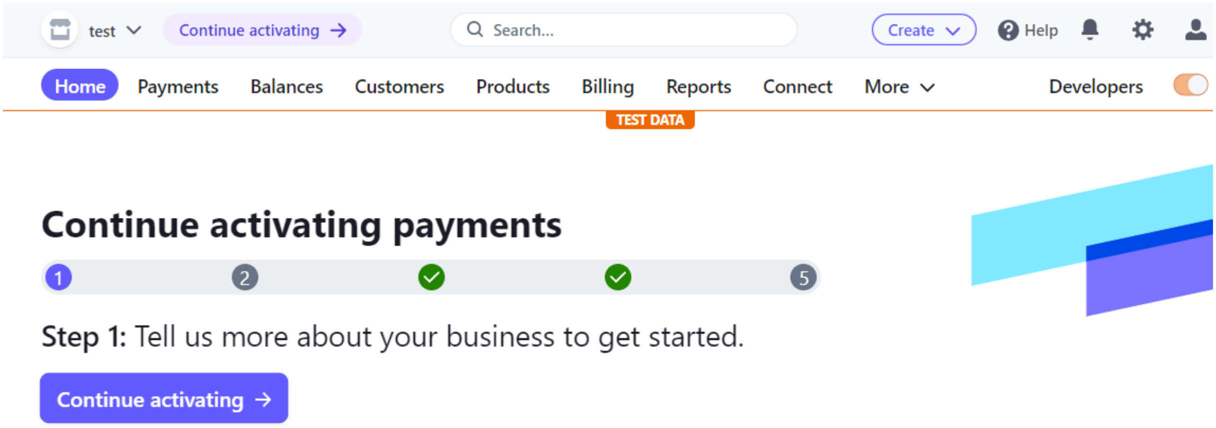
U projektu se Stripe iskorištava kako bi se ostvarila plaćanja proizvoda koji su ponuđeni u virtualnoj trgovini. Isti se može iskoristiti za digitalne proizvode, usluge pa i donacije. Neke od prednosti platforme Stripe su i dodatne funkcionalnosti mogućnosti povratka novca kao i prigovor na rad klijenata, a obje funkcionalnosti uključene su u samome procesu plaćanja. Tako se ostvaruje provjera nad samim pružateljima usluga. Osim toga, Stripe nudi razne integracije s drugim popularnim web aplikacijama kao što su WordPress, Shopify, Squarespace s ciljem povećanja i proširenja funkcionalnost same web aplikacije. Iako je o samome Stripeu već bilo rečeno nekoliko riječi na početku ovoga rada, u ovome će poglavlju biti detaljno opisan. Poglavlje će biti podijeljeno na nekoliko potpoglavlja koja će se sastojati od inicijalizacije same podrške u vlastitoj aplikaciji, uslugama koje se nude njegovim korištenjem na korisničkoj strani, implementacije u kodu te pregleda kupovina i obavještavanje korisnika o izvršenoj kupovini.

4.1 Inicijalno postavljanje sučelja

U projektu će biti potrebno dodati još nekoliko datoteka koje će biti ispunjene vlastoručno ispisanim programskim kodom zaslužne za komunikaciju između aplikacije i samoga Stripe sučelja. To se mora realizirati u par sljedećih koraka. Za početak je potrebno napraviti vlastiti korisnički račun. Isti se kreira jednostavno, kao i za većinu sučelja koja zahtijevaju vlastite korisničke račune za korištenje. Nakon izrade, prikazuje se sučelje kontrolne ploče s koje se mogu iščitati brojne informacije o samoj prodaji i ostalim parametrima. Isto će biti detaljno prikazano i opisano u jednome od sljedećih poglavlja. Potom je potrebno, prateći dokumentaciju koja je napisana za više programskih podrški, implementirati isto u aplikaciju. Stripe koristi obrasce koji se koriste za primanje informacija o podacima plaćanja. Tako je moguće napraviti vlastiti obrazac ili koristiti generički kako bi se prikupili podaci o korisniku. Svi se podaci kasnije jednostavno iščitavaju s kontrolne ploče svakoga korisnika. Pri samome postavljanju korisničkoga računa, Stripe odmah daje mogućnost verifikacije biznisa koji će se odvijati preko izrađene stranice. Tako se u desetak skočnih prozora moraju navesti ključni podaci o istome. To su podaci kao što su matični broj poslovnoga objekta, PDV identifikacijski broj, IBAN te adresa biznisa. Preko tih podataka Stripe radi detaljnu provjeru kako bi se na što sigurniji način sučeljem koristili prodavači i kupci. Vrlo je važno naglasiti kako je u trenutku kada je korišten Stripe za izradu samoga projekta, isti bio dostupan za pregled samo na određenim internetskim preglednicima. Tako je u tome slučaju došlo do poteškoća s prikazom određenih mogućnosti koje se mogu odraditi preko samoga sučelja jer je korišten krivi preglednik. Nakon zamjene preglednika, sve su mogućnosti uredno bile vidljive. Prilikom ulaska u samo sučelje Stripea ispod adrese javit će se napomena, ukoliko određeni preglednik preko kojega se koristi sučelje nije podržan. Ujedno se prikazuje i lista preglednika preko kojih je to moguće obaviti.

4.2 Mogućnosti upravljačke ploče

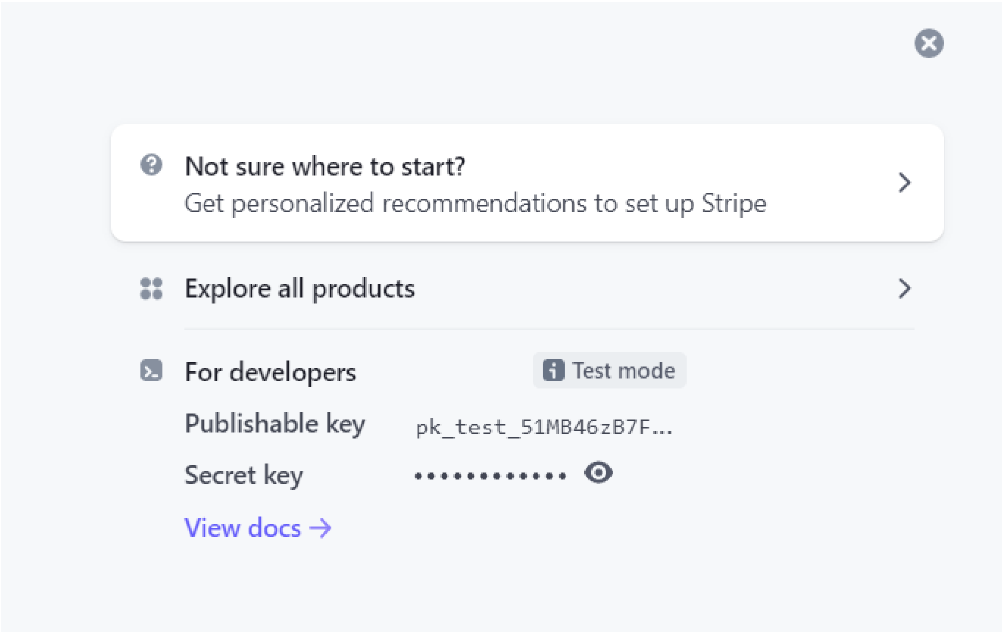
Potrebno je ponoviti da će se u izradi ovoga projekta koristiti samo testna verzija plaćanja zbog nepotrebnoga prebacivanja novaca među računima i plaćanja provizije na isto. Stripe pri ulasku odmah postavlja sučelje za rad u takozvanome testnom načinu rada (*slika 4.1.*). U tome se načinu rada mogu isprobavati kupnje s karticom koja nije stvarna i služi samo za prikaz podataka i prebacivanja s vlastite aplikacije na Stripe sučelje.



Slika 4.1. – Prikaz mogućnosti početne stranice

Tu se odmah može uočiti kompleksnost samoga sučelja. U gornjemu desnom kutu, žuto označena ikona omogućuje uključenje i isključenje prethodno spomenutoga testnog načina rada. Ako korisnik i želi izaći iz testnoga načina rada, mora odraditi verifikaciju svojega biznisa koja je također prije spomenuta. Bez iste sustav dozvoljava rad samo u testnome načinu. Gornji je lijevi kut rezerviran za informacije o samome računu. Postoji mogućnost da se doda više računa kojima je lakše upravljati na jednome mjestu. Odmah su ispod vidljive poveznice kojima se možemo kretati po najvažnijim dijelovima stranice. To su plaćanja, valute, kupci, proizvodi itd. Odmah se ispod tih poveznica mogu uočiti ključevi koji će nam pomoći za daljnju komunikaciju.

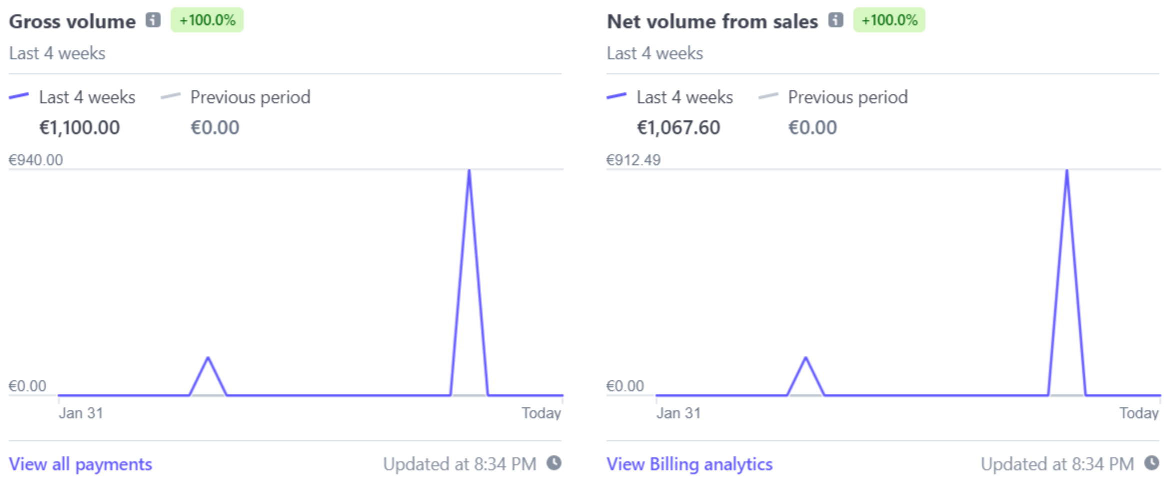
Oba načina rada (testni i prave transakacije) koriste dva ključna parametra za komunikaciju. (*slika 4.2.*) To su *Publishable key* i *Secret key*. Oni služe za spajanje izrađene aplikacije s korisničkim računom i jednim od ova dva načina rada. Uz ta dva ključa, potrebno je napomenuti kako se mora izraditi i ključ pod imenom *Shipping rate ID* bez kojega aplikacija neće ispravno slati podatke ako nije ispravno konfiguriran. Isti će biti prikazan na kontrolnoj ploči pa tako i konfiguracijom u programskome kodu. Na samome ulasku u sučelje Stripea na početnoj kontrolnoj ploči možemo vidjeti nekoliko informacija koje će biti važne za samu konfiguraciju povezivanja. To su prije spomenuti ključevi.



Slika 4.2. – Prikaz dva glavna ključa za povezivanje

Njih se iskorištava u kodu. Oni se ne smiju dijeliti i uvijek trebaju biti tajni. S obzirom na to da se ovdje koristi testni slučaj, isti mogu ostati poluvidljivi. Ključevi se ne mogu zasebno generirati, već ih sustav sam generira i stavlja korisniku na raspolaganje.

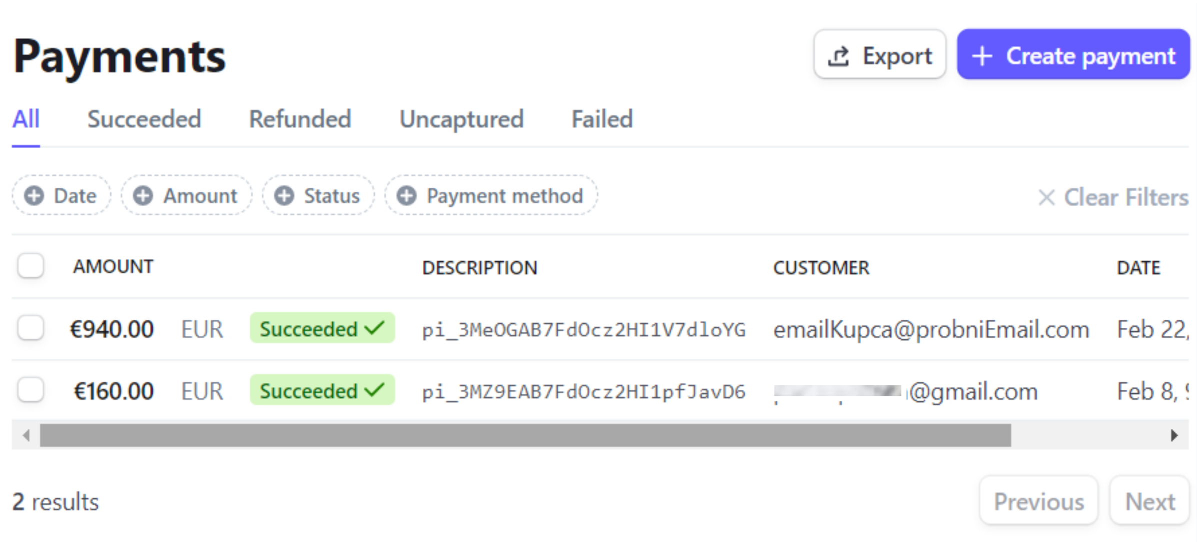
Ispod ključeva dolazi do ispisa grafova koji prikazuju statistiku prodaje za željeni period. (*slika 4.3.*) Kako bi se prikazao rad grafova, odrađena je nekolicina testnih kupnji na određene datume da bi se vidio porast i pad u korištenju usluga.



Slika 4.3. – Grafovi bruto i neto zarade (sa i bez cijene slanja pošiljke)

Sučelje daje mogućnost prikaza raznih grafova, sve u svrhu olakšavanja prikaza podataka važnih za korisnikovo poslovanje.

Navigira li se dalje po stranici, u kratici *payments* mogu se provjeriti sve kupnje koje su obavljene kroz sučelje. (*slika 4.4.*) Tako se na samoj stranici prikažu sve kupnje u određenome datumskom periodu koji je postavljen. Svaku je kupnju moguće detaljno pogledati klikom na istu. Tamo je moguće iščitati podatke o kupnji kao što su podaci za uplatu, koje je proizvode korisnik kupio, evaluacija rizika kupnje koji se odrađuje na samome poslužitelju Stripea, IP adresa kupca te IP adresa koja se određuje preko IP adrese itd. Kako bi se olakšalo praćenje uplata, postoji mogućnost detaljnoga ispisa uplata klikom na poveznicu gdje piše *export*. Ujedno je moguće odrađivati i povratna plaćanja korisnicima. Isto je implementirano kako bi se olakšao protok sredstava ukoliko dođe do neželjenih greški i poteškoća u željenome radu. Isto se ostvaruje pritiskom na poveznicu *Create payment*.

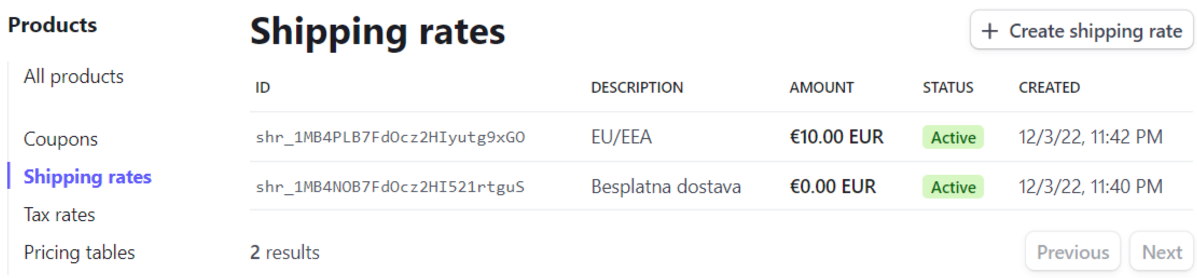


Slika 4.4. – Pregled plaćenih usluga

Uz praćenje plaćenih usluga/proizvoda, sučelje također nudi i praćenje kupaca i njihovoga interesa prema proizvodima i količinama trošenja. Za svakoga je kupca moguće iščitati njegove podatke koji su poslani preko aplikacije u Stripe sučelje koji se kasnije spajaju na jedno mjesto. Analogno tomu kupci se kategoriziraju po potrošnji te se prikazuju statistike o njima i njihovim kupovinama. Tako je prodavaču olakšano prilagoditi se kupcu i orijentirati se prema svakome pojedinačno. Uz to, na raspolaganju su odmah i sve informacije kojima se može ostvariti direktna komunikacija sa samim kupcem.

Na samome sučelju, kao što je moguće organizirati i grupirati kupce te se prilagoditi njihovim potrebama, također je moguće grupirati i pogledati statistike samih proizvoda. Tako se proizvođaču omogućuje lakše manipuliranje i prikaz dostupnih, nedostupnih, popularnih i manje popularnih proizvoda. Treba pripaziti da se sami proizvodi moraju ručno dodati prije nego što se sama aplikacija pusti u rad. Ukoliko se isti ne dodaju, prikaz statistike neće biti točan.

Kako bi kupnja bila još intuitivnija, moguće je implementirati i kupone. Kuponi služe kako bi pridodali popust na određeni proizvod. Kako većina stranica može pridodati kupon za sve stavke ili stavku pojedinačno, ovdje je moguće to ostvariti samo po jednoj stavci iz trgovine. Kuponi se generiraju tako da im se pridodaje ime te identifikator, koji su ključni za njihov rad. Mogućnosti se šire i dalje, gdje je moguće pridodati spuštanje fiksne tarife cijene ili samoga postotka na cijenu. Ujedno je moguće i odrediti trajanje samoga kupona koji je moguće iskoristiti. Oni mogu biti ponavljajući ili regenerirajući, tako da korisnici mogu više puta kroz različite kupnje iskoristiti kupon ili ga iskoristiti samo jednom te potom moraju pronaći drugi generirani kod, ako isti postoji. U tome dijelu dolazi do prije navedenih naknada za slanje proizvoda po određenim dijelovima svijeta. Izrađuje se poseban plan za slanje u određene dijelove svijeta. Pri izradi same grupacije opcije slanja, potrebno je odrediti parametre koji će se koristiti za određena slanja. Tako Stripe daje mogućnost odabira valute koja će se koristiti samo za cijenu slanja proizvoda, opis same vrste slanja te prikaz prosječnoga vremena unutar kojega će proizvod biti dostavljen na kućnu adresu. To se unosi ručno gdje sam pružatelj usluga mora imati pouzdane informacije o istome. Nakon izrade željene specifikacije, sustav izrađuje identifikator koji će se nadalje koristiti u samome kodu kako bi se specificiralo područje slanja proizvoda. U tome će primjeru biti prikazane dvije regije slanja, područje Hrvatske i EU/EEA zona. Budući da je projekt rađen dok se još Hrvatska nije nalazila u EU zoni, cijene su razdvojene za ta dva područja.



Slika 4.5. – Prikaz izrade informacija nakon izrade plana slanja

Svi se planovi mogu ručno mijenjati u bilo kojemu trenutku izravnim pritiskom na određeni plan, kao što je prikazano *slikom 4.5*. Korisniku je omogućeno lakše upravljanje istima i prikaz informacija koje su potrebne na jednome mjestu vezane za slanje proizvoda. Kako bi se riješio problem s plaćanjem poreza, moguće je za regije odrediti i količinu poreza koja se plaća. Ukoliko korisnik teži većoj skalabilnosti svoga biznisa, može se poslužiti s implementiranom dijelom dodjele poreza po regijama koje se izrađuju po području iz kojega kupac kupuje. Sustav sam dodjeljuje porez te ga obračunava u cijenu. O tim postocima korisnik ne treba tada voditi brigu jer su oni ažurni i sam Stripe vodi računa o istome. Svakako bi bilo poželjno na samome inicijalnom testiranju provjeriti obračunavaju li se oni kako treba da ne bi došlo do nepotrebnih odskakanja u radu aplikacije. Iako se sve ove postavke odnose na fizičke proizvode, potrebno je navesti i mogućnost kreiranja načina plaćanja koji je primjeren za usluge koje se mogu nuditi na određenim aplikacijama. Za to su zaslužne tarife plaćanja (eng. *price plans*). [19]. To su tarife za određene usluge koje se mogu validirati na vremenskoj bazi ili po bazi korištenja usluga. U prijevodu, ukoliko netko želi nuditi usluge koje neće biti dostupne korisniku na neograničeno vrijeme ili korištenje, tim putem može regulirati iznose za određeno vrijeme korištenja svojih usluga. Kao i za proizvode, ukoliko želimo potpunu mogućnost praćenja informacija o istim, oni moraju biti dodani ručno unutar Stripe sučelja prije puštanja aplikacije u pogon gdje se koriste. Sve se te postavke mogu pronaći u sekciji *Products* na vrhu sučelja.

Ukoliko se Stripe planira koristiti na velikoj skalabilnoj razini, postoje sučelja preko kojih je moguće povezati izrađenu aplikaciju s drugim platformama namijenjenim prodaji proizvoda. Tako je moguće isti povezati s platformama kao što je danas popularni *Wix* ili *Squarespace*. Preko njih je također moguće ostvarivati kupnje, gdje će se svaka transakcija prikazati unutar samoga Stripe sučelja, dok izrađena aplikacija neće biti potrebna. Ujedno se ne radi niti preusmjeravanje ako se kupnja ostvaruje na nekim drugim stranicama na izrađenu aplikaciju, već samo dolazi do prikaza transakcije s napomenom na kojemu se sučelju ona dogodila.

Također je moguće implementirati aplikaciju na način da radi kao posrednik usluga. Danas je vrlo popularna ova metoda kupnje proizvoda, gdje sam vlasnik stranice ima mogućnost prikaza proizvoda koji nisu u njegovome posjedu. Ideja je blago povećanje cijene proizvoda drugoga proizvođača te osobna dobit. Sve se informacije trebaju uzeti sa stranice gdje se primarno nalazi proizvod te prikazati na vlastitoj stranici. Ukoliko kupac kupi s izrađene stranice, sredstva koja su inicijalno namijenjena za proizvod automatski se prosljeđuju osobi koja ima proizvode ili usluge na raspolaganju, dok se razlika ostavlja na računu korisnika koji je u vlasništvu izrađene aplikacije. Ukoliko se aplikacija izrađuje za tu namjenu, ona ima svrhu raditi kao običan posrednik usluga. Ovakav se način davanja usluga u stranoj literaturi naziva *dropshipping.*

Nadalje, Stripe uvodi opciju plaćanja članova ili suradnika po principu ostvarenja. Cilj je postaviti tim ljudi na vlastitome korisničkom računu koji će biti autonoman naspram ostaloga trgovanja. Sve ovisi o samome vlasniku usluga i dogovoru među suradnicima. Stripe daje mogućnost zarade po postotku i automatsko plaćanje suradnika na određena ostvarenja. Tako je svakome korisniku definiran račun po kojemu se određuje koliko postotne zarade ide kome. Ukoliko je riječ o manjim sustavima i aplikacijama, moguće je odraditi obično plaćanje korisnika koji surađuju s vlasnikom usluge na jednostavan način, tako da sam Stripe pamti tko je suradnik u poslovanju.

4.3 Developers sučelje

Kako bi se korisnicima olakšao posao traženja pogreški kod implementacije platforme, osim testnoga načina rada programske podrške, Stripe uvodi i sučelje pod imenom *Developers*. To se sučelje može pronaći na lijevoj strani u gornjemu desnom kutu početne stranice, odmah pokraj mogućnosti paljenja i gašenja testnoga načina rada. Otvaranjem toga sučelja, odmah na prikaz dolazi grafičko sučelje koje prikazuje API pozive, njihove uspješne i neuspješne realizacije. To se sve prikazuje preko grafa koji omogućuje korisniku lakše snalaženje s istima po vremenu unutar kojega su se oni odvijali. Na toj su stranici prikazane samo osnovne informacije o samim pozivima. Kako bi se dodatno pomoglo budućim korisnicima s lijeve je strane moguće prikazati i ostale kategorije koje će biti od pomoći kod implementacije. Tu se mogu ponovno vidjeti *Publishable key* i *Secret key* koji se nekada i ne prikažu na samoj početnoj stranici. Uz te ključeve, moguće je izraditi i *Restricted key*. To je mogućnost koja se najčešće koristi unutar aplikacije na poziv samoga vlasnika usluge – administratora, tj. isti se može iskoristiti kada se koriste usluge čitanja podataka sa Stripe sučelja za razne primjene praćenja rada sustava (eng. *monitornig*).

Uz gore navedene mogućnosti, ujedno postoji mogućnost uvida u „događaje i zapisnike“ (eng. *events and logs*). *Event* grupa prikazuje sve događaje koji su vezani za sama plaćanja. Tako su najčešće stavke koje se ovdje prikazuju uspješnost i neuspješnost transakcija, poruke zbog kojih je transakcija uspješno/neuspješno provedena te ukoliko je isteklo vrijeme sjednice za plaćanje koja je bila generirana. Svaki je događaj prikazan s pripadajućim identifikatorom te vremenom kada je izvršen. Pojedini se događaji mogu i otvoriti te se iz istih može iščitati koji parametri su poslani putem koda kako bi se lakše otklonila podrška, ako je do nje došlo. Unutar skupine *logs* također je moguće provjeriti zapisnike koji su slični kao i zapisnici događaja. Svi su zapisi poredani vremenski te je svakome pridodan identifikator. Ukoliko se otvori pojedini zapisnik, ponovno je moguće provjeriti jesu li parametri, koji su slani putem koda, ispravno poslani. Ujedno, moguće je provjeriti i IP adrese korisnika koji su imali poteškoću te statusni kod pogreške kako bi se lakše otkrila greška.

4.4 Implementacija u kodu

Cijeli proces implementacije same programske podrške sustava plaćanja započinje, kao što je prije navedeno, izradom samoga računa na Stripe platformi. Budući da je isti proces vrlo intuitivan, opis će izrade korisničkoga računa biti preskočen u tome dijelu. Nadalje, potrebno je izraditi novi dokument koji će biti baza za pisanje koda vezanoga za implementaciju samoga Stripe sučelja. Tako je potrebno pozicionirati se unutar projekta u direktorij

projekt/pages/api/Stripe_poslužitelj

Unutar posljednjega direktorija na navedenoj putanji odvija se cijeli proces poslužiteljske strane koja će biti korištena [17, 18]. Sve pisano unutar ovoga direktorija neće biti prikazano na samoj *frontend* instanci sučelja. Zbog toga nema potrebe raditi zaseban poslužitelj kao što je to kod slične Node.js platforme.

Unutar novoizrađene datoteke potrebno je povući (*import*) samo Stripe sučelje, koje je ranije konfigurirano kao *dependency*. Na taj se način nije potrebno brinuti o ručnoj instalaciji ili dodavanju dodatnih sučelja koja se koriste u aplikaciji. Prije početka pisanja samoga koda, vrlo je bitno detaljno proučiti samu dokumentaciju koja je dostupna na stranicama Stripea. Nakon „povlačenja" sučelja unutar same datoteke, potrebno je napraviti novu instancu Stripea. Ukoliko se koristi jedan od modernijih uređivača koda, on će u početku instanciranja sam ponuditi atribute koji se koriste. Proučavanjem dokumentacije ili uz pomoć uređivača koda, utvrđuje se da je glavni parametar *apiKey* koji se prosljeđuje novoj instanci Stripea. Taj je *apiKey* ujedno *Publishable key* koji je bio opisan u prethodnome potpoglavlju. Nadalje se sve svodi na čitanje dokumentacije i korištenje koda koji je djelomično napisan u samoj dokumentaciji. Kod koji može poslužiti svodi se na dvije kategorije. Prva će se kategorija koristi u tome projektu, a to je kod za procesuiranje samoga plaćanja. Druga je kategorija izrada toka podataka koji korisnici unose kako bi se isti pohranili na poslužitelj i iskoristili u svrhu plaćanja [20]. Tu je korištena instanca toka koju je Stripe predefinirao. Kako bi implementacija bila još jednostavnija i lakša korisnicima koji nisu dobro upoznati s jezikom u kojemu razvijaju aplikaciju, moguće je odabrati djelomično predefiniran kod za više jezika koji će se koristiti na samome *frontendu* i na *backendu*.

Zatim je potrebno napraviti funkciju koja će se spajati na Stripe poslužitelja. Izradom nove datoteke na putanji

projekt/lib/stripe_poziv

izrađuje se nova funkcija pod nazivom *getStripe( )* [21]. Unutar iste potrebno je provjeriti jedan slučaj, a to je slučaj u kojemu provjeravamo je li aplikacija već povezana s poslužiteljem. Izradom nove varijable u koju se spremaju parametri potrebni za komunikaciju, drugom se funkcijom *loadStripe( )* šalju ponovno tajni *Publishable key*. Nakon provjere, funkcija vraća potvrdu o povezivanju. Iako je u *kodu 4.1.* djelomično prikazan ključ, savjetuje se da se takve informacije ne prikazuju direktno unutar koda već kao varijabla u *.env* datoteci. Ona služi kako bi se takve povjerljive informacije teže saznale prilikom postavljanja na poslužitelj.

```
import {loadStripe} from '@stripe/stripe-js';
let stripePromise;

const getStripe = () => {
    if(!stripePromise){
        stripePromise = loadStripe("pk_test_51Hq...");
    }
    return stripePromise;
}

export default getStripe;
```

Kod 4.1. – Povezivanje sa Stripe poslužiteljem

Nakon uzimanja koda koji je globalno dostupan na korištenje i modificiranje, potrebno je izraditi par izmjena unutar istoga. Te se izmjene odnose na prvotno prikazanu putanju gdje se nalazi „*Stripe_poslužitelj*“. Kako bi kod bio pregledniji za čitanje i pronalaženje moguće pogreške, svi su parametri koji će biti potrebni za komunikaciju s poslužiteljem izdvojeni na vrh samoga *try-catch* bloka koji je predefiniran u dokumentaciji. Unutar tih parametara mogu se uočiti oni koje je moguće modificirati kako bi se prilagodili korisničkome iskustvu u različitim situacijama implementacije. Tu se koristi predefiniran tok plaćanja pa je iz toga razloga *submit_type*, *mode*, *payment_method_types* te *billing_address_collection* postavljen kao što je navedeno u dokumentaciji. Oni služe za postavljanje načina plaćanja te prikupljanja korisnikove adrese na koju će pošiljka biti poslana. Ukoliko bude potrebno izmijeniti jedan od gore navedenih parametara, to se može odraditi jednostavnim čitanjem dokumentacije. Nadalje, potrebno je konfigurirati naknadu koja će biti naplaćena za slanje samoga proizvoda. Stripe nudi mogućnost postavljanja parametara za slanje te nakon izrade stvara jedinstveni identifikator preko kojega će se koristiti prije postavljeni parametri za slanje. Za izradu same karakteristike, primjerice slanja proizvoda, potrebno je vratiti se na korisničku ploču te otići po putanji na:

Kontrolna_ploča/products/shipping_rates/Create_shipping_rate

Tako se dodaju parametri koji će činiti jedinstveni način slanja pošiljki. Nakon što se izradi sam identifikator slanja, isto je potrebno zalijepiti u kod na mjesto *shipping_options*. Vrlo je bitno obratiti pažnju na valutu koja se koristi. Ona mora biti ista unutar koda i unutar sučelja gdje se izrađuje plan slanja pošiljki. Nadalje je potrebno vratiti sve parametre koji će se koristiti za obradu plaćanja i prikaza pred plaćanje. Tako se vraća valutu (koja mora biti ista onoj postavljenoj u Stripe sučelju), ime proizvoda i sliku proizvoda. Zatim je potrebno postaviti i parametre za omogućavanje kupnje više istih proizvoda. Čitanjem dokumentacije potrebno je zaključiti kako se količina novčanih sredstava mora prikazivati u centima pa se isti u kodu množi sa sto. Ovisno o tome prođe li kupnja uspješno ili neuspješno, internetski preglednik napraviti će preusmjeravanje na stranicu koja je prikazana na *slici 3.6.* ili će odraditi povratak na samu trgovinu uz notifikaciju da se dogodila neka pogreška. Rješenje unutar programa opisano je *kodom 4.2*.

```
import Stripe from 'stripe';
const stripe = new Stripe("sk_test_51Hq...");
export default async function handler(req, res) {
  if (req.method === 'POST') {
    try {
      const params = {
        submit_type: 'pay',
        mode: 'payment',
        payment_method_types: ['card'],
        billing_address_collection: 'auto',
        shipping_options: [
          { shipping_rate: 'shr_1MYmcZEnylLNWUqjttIDeIKw' },
        ],
        line_items: req.body.map((item) => {
          return {
            price_data: {
              currency: 'eur',
              product_data: {
                name: item.name,
                images: [newImage],
              },
              unit_amount: item.price * 100,
            },
            adjustable_quantity: {
              enabled:true,
              minimum: 1,
            },
            quantity: item.quantity
          }
        }),
        success_url: `${req.headers.origin}/success`,
        cancel_url: `${req.headers.origin}/abort`,
      }
```

Kod 4.2 – Provjera komunikacije te izmjena informacija klijenta i poslužitelja

Gore će navedeni postupak raditi tek nakon što se funkcionalnosti povežu s poveznicama koje odrađuju samo preusmjeravanje prema Stripeu. Tako je potrebno izmijeniti što će raditi poveznica kada se pritisne na nju. Unutar dvije datoteke, *Cart.js* te *Item.js* potrebno je na *onClick* funkcionalnost poveznica dodati funkciju koja će raditi prosljeđivanje parametara. Za početak se mora uvesti datoteka Stripea koja vraća provjeru o povezivanju na poslužitelj. U novokreiranoj funkciji treba pozivati instancu Stripea koja je ranije uvedena. Potom je potrebno čekati na odgovor poslužiteljske strane koja je konfigurirana u datoteci *Stripe_poslužitelj*. Funkcija koja odrađuje čekanje prima dva ključna parametra. To su putanja do „*Stripe_poslužitelj*" datoteke te objekt koji će provjeravati sve opcije koje moraju biti zadovoljene prije komunikacije s poslužiteljem. Nadalje je potrebno odraditi provjeru poteškoće sa spajanjem. Ukoliko je spajanje uspješno odrađeno, čeka se odgovor poslužiteljske strane, vraća se korisniku povratna informacija da je sve uspješno odrađeno te se radi prebacivanje na Stripe sučelje. Iznad navedene funkcionalnosti prikazane su s *kodom 4.3.*

```
const checkoutProcess = async() =>{
  const stripe = await getStripe();
  const response = await fetch('/api/stripe', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(cartItems),
  });
  if(response.statusCode === 500) return;
  const data = await response.json();
  console.log(data)
  toast.loading('Procesuiranje kupovine...');
  stripe.redirectToCheckout({ sessionId: data.id });
}
```

Kod 4.3. – Funkcije za prebacivanje parametara na poslužiteljsku stranu Stripea

4.5 Mogućnosti obavještavanja korisnika

Uspješnim se preusmjeravanjem i slanjem parametara proizvoda na poslužiteljsku stranu Stripe sučelja dolazi do mogućnosti unosa informacija potrebnih za daljnje procesuiranje kupnje. Kako je u tome slučaju korišten generički tok podataka koji korisnik unosi, prikaz će biti isti kao i iz poglavlja 3.4. Ukoliko se korisnik odluči na kupnju, nakon unosa validnih podataka, provjere istih na poslužiteljskoj strani te validacije i transakcije sredstava, korisniku se vraća povratna informacija o uspješnoj kupnji. Na samoj stranici uspješne kupnje, korisnik je ponovno obaviješten o naručenim proizvodima, cijeni te količini istih. U drugome slučaju, korisnika se prebacuje na stranicu gdje je informiran da kupnja nije uspješno izvršena te da provjeri određene parametre. Ti su parametri također generički jer se kod greški neuspješne kupnje najčešće radi o sličnim problemima. To su najčešće problemi nedovoljne količine sredstava na računu, neispravnost korisničkih podataka ili verifikacija koja se odvija nakon zahtjeva za kupnjom putem e-bankarstva (*PayWay* i sl.). Kako bi se dodatno informiralo korisnika o uspješnoj kupovini, unutar Stripe sučelja može se omogućiti povratno informiranje putem elektroničke pošte. Kako korisnik prilikom unosa podataka vezanih za kupnju unosi i vlastitu e-adresu, Stripe nakon omogućavanja te opcije šalje generičku poruku sa svim podacima koji se mogu također iščitati sa stranice uspješne kupnje. Potrebno je napomenuti kako se povratna e-pošta neće slati ukoliko se koristi testni način rada. To je napravljeno iz sigurnosnih razloga kako bi se izbjegao veliki broj generiranja i slanja e-pošte prema određenim korisnicima i zagušenja poslužiteljske strane.

5 MOGUĆNOSTI INSTALACIJE APLIKACIJE NA POSLUŽITELJSKOJ STRANI

Izradom same aplikacije i testiranjem svih njezinih mogućnosti moguće je odraditi i instalaciju iste na jednome od poslužitelja koji pružaju takve usluge. Većina takvih poslužitelja naplaćuje usluge davanja svoga virtualnog prostora za tuđe internetske aplikacije. Kako bi demonstracija funkcionalnosti bila odrađena u potpunosti, u ovome će poglavlju biti opisane dvije tehnologije kojima se može odraditi prebacivanje i instalacija aplikacije na određenoga poslužitelja. Obje tehnologije koriste sličan način rada pa će tako biti navedene kao alternativne varijante jedna drugoj. Potrebno je reći da ne postoji osobna preferencija u izboru jedne od platformi te da obje funkcioniraju na vrlo sličan način. Korisnik je taj koji će na samome kraju odlučiti koju će platformu koristiti. Odluka se platforme najčešće svodi na cijenu usluga, količinu prometa koju korisnik može iskoristiti po određenoj cijeni te jednostavnost korištenja i manipulacija samom aplikacijom koja je postavljena na poslužitelja.

5.1 Netlify platforma

Netlify [22] je platforma u oblaku koja pruža web developerima i timovima sveobuhvatni set alata za izgradnju, implementaciju i upravljanje modernim web projektima. Omogućuje developerima jednostavno implementiranje statičkih stranica i *serverless* funkcija s nekoliko klikova te pruža napredne značajke poput kontinuirane implementacije, obrade obrazaca i testiranja podjele. Platforma je poznata po svojoj jednostavnosti korištenja i korisnički prijateljskom sučelju za razvoj weba, kao i besprijekornoj integraciji s popularnim alatima za razvoj weba poput Gita, GitHuba i Bitbucketa. Također nudi ugrađene mogućnosti CDN-a (mreže za dostavu sadržaja), što osigurava brzo i učinkovito učitavanje web stranica za korisnike diljem svijeta. Osim toga, Netlify pruža širok raspon integracija s uslugama treće strane, poput Slacka, Trelloa i Zapiera, kako bi timovi mogli pojednostaviti svoje radne procese i učinkovitije surađivati. Njegovi snažni alati za analizu i A/B testiranje omogućuju developerima da steknu uvid u ponašanje korisnika i optimiziraju svoje stranice za bolju izvedbu i stope pretvorbe.

Sama integracija aplikacije unutar Netlifyja započinje vrlo jednostavno. Korisnik je primoran izraditi korisnički račun preko kojega će pristupati samome računu na kojemu može voditi evidenciju o svojim internetskim aplikacijama. Izrada računa omogućena je preko korisničkoga računa e-pošte, no ne preporuča se. Poželjno je da se korisnik odmah u početku prijavi putem jedne od platformi za razvoj programskoga koda. Te su platforme navedene u odlomku iznad. Na samoj početnoj stranici moguće je uočiti sam protok podataka koji je ostvaren kroz sve aplikacije koje se nalaze na poslužitelju. Kako ovaj pružatelj usluga daje besplatne usluge od 100 GB prometa, postoje i opcije gdje pretplatničkim načinom rada korisnik može ostvariti veći promet.

Samom prijavom u Netlify korisnički račun, na početnoj je stranici vrlo uočljiva poveznica *Add new site* koja korisnika vodi na odabir aplikacije ili stranice koju želi uvesti na poslužitelja. Ukoliko se korisnik prijavljuje s jednom od iznad navedenih platformi za razvoj koda, Netlify automatski prikazuje sve repozitorije koji se nalaze na korisnikovoj razvojnoj platformi. Nakon odabira, korisniku se također daje mogućnost odabira budućega vlasnika aplikacije, inicijalne naredbe s kojom će se aplikacija pokretati na poslužitelju te grane repozitorija iz koje se čita kod. Završetkom podešavanja ovih parametara kreće prebacivanje na poslužitelja. Proces traje nekoliko minuta, no ovisi o raznim parametrima. Završetkom se samoga prebacivanja generira besplatna domena, adresa kojom se pristupa samoj stranici. Ta je adresa bazirana na imenu samoga repozitorija u kojemu se nalazi te određenim nasumičnim odabirom slova i brojeva kako bi ju učinili jedinstvenom.

5.2 Vercel platforma

Vercel [23] je platforma u oblaku za izgradnju i implementaciju internet stranica i aplikacija. Vercel je prvotno stvoren kao platforma za implementaciju popularnoga React okvira, ali se od tada proširio da podržava širok raspon programskih jezika i okvira. Koristi *serverless* arhitekturu, što znači da programeri mogu izgraditi i implementirati aplikacije bez brige o upravljanju poslužiteljima ili infrastrukturi. Jedna je od ključnih značajki Vercela automatska mogućnost skaliranja. To znači da kako se povećava promet na web mjestu ili aplikaciji, Vercel automatski može skalirati potrebne resurse za rukovanje opterećenjem. To pomaže osigurati da web stranice i aplikacije ostaju učinkovite i dostupne, čak i tijekom razdoblja velikoga prometa. Još je jedna prednost korištenja Vercela, a to je njegova jednostavna integracija s popularnim alatima za razvoj poput GitHuba i GitLaba. To olakšava programerima implementaciju njihovoga koda i suradnju s članovima tima.

S obzirom na to da je za implementaciju i proces instaliranja aplikacije na poslužitelja u gornjemu primjeru objašnjen jednostavniji način, kroz grafičko sučelje, Vercel će biti pojašnjen kroz konfiguraciju u komandnoj liniji. Naravno, za početak potrebno je otići na stranice same platforme gdje se odvija čitanje dokumentacije korištenja [24]. Prikaz će biti odvojen na GUI korištenje te na CLI korištenje. Pregledavanjem CLI dijela dokumentacije prva je naredba za globalnu instalaciju platforme napisana na samome početku. Ista glasi (instalacija za npm slučaj korištenja pročitana iz dokumentacije):

npm i –g vercel

Komponentu je bitno instalirati sa parametrom „*-g*“, koji omogućuje globalnu instalaciju u radnoj okolini. Budući da je to globalna instalacija, u početku nije potrebno voditi računa o samoj putanji gdje će isti biti postavljen. Nakon samokonfiguracije i instalacije platforme u radnu okolinu, potrebno je ponovno preko komandne linije otvoriti samu putanju direktorija u kojoj se nalazi projekt koji je potrebno postaviti na poslužitelja. Potom, jednostavnim unosom komande „*vercel*“ na komandnoj ploči započinje proces postavljanja aplikacije na poslužiteljsku stranu. Za početak, Vercel provjerava ako je korisnik prijavljen preko jednoga od gore navedenih računa na kojima se odvija skladištenje same aplikacije. Ukoliko korisnik nije prijavljen, isto je moguće učiniti preko intuitivnoga korištenja komandne linije. Nadalje, moguće je postaviti radni direktorij ukoliko isti nije predefiniran za upotrebu unutar NextJS platforme ili nasuprot odrediti koji je to direktorij. Na samome kraju, korisniku se postavljaju dodatne opcije preko kojih se radi upravljanje između aplikacije i poslužiteljske strane. Korisnik ima tri mogućnosti: *Build command, Development command* te *Install command* koje se izvršavaju prije samoga postavljanja aplikacije na poslužitelj. Nadalje, par minuta nakon obavljanja pozadinskih procesa, korisniku se generira poveznica preko koje pristupa dijelu gdje se nalaze dodane postavke za vlastiti projekt. Detaljnim se čitanjem dokumentacije uočava kako je potrebno voditi računa ukoliko se projekt oslanja na „varijable okoline“ (eng. *enviromental variables*) koje se prikazuju na samoj generaciji stranice. Ukoliko se one koje se koriste za prikaz na stranici ne pošalju u samo sučelje Vercela, iste se neće prikazati na zaslonu aplikacije. Kada korisnik želi prikazati takve varijable, dodavanje se može ponovno odraditi preko grafičkoga sučelja (eng. *GUI*) te komandne linije (eng. *CLI*). Za dodavanje varijabli preko komandne linije potrebno je upisati željenu varijablu u dolje navedeni dio koda

vercel env add NEXT_PUBLIC_IME_VARIJABLE

Potom je potrebno spremiti promjene koje su učinjene na samome projektu. Isto se odrađuje ponovnim upisom naredbe *vercel*. Na ovoj se platformi također odrađuje automatsko generiranje besplatne domene sa ključnim riječima imena projekta te dodanim nasumičnim slovima i brojevima kako bi domena bila jedinstvena.

6 ZAKLJUČAK

Cilj je projekta bio prikazati kako na jednostavniji način odraditi mogućnost izrade vlastite e-trgovine. Za uspješno odrađivanje zadatka koji je opisan kroz cijeli projekt, potrebno je imati programerskoga predznanja te je vrlo poželjno imati prijašnje prakse u izrađivanju drugih internetskih aplikacija. Nadalje, potrebno je reći da ovo nije projekt za početnike, koliko god se isti činio jednostavno. Ukoliko postoje budući korisnici koji će se pozivati na ovaj projekt, istima se savjetuje da se detaljno pozabave čitanjem dokumentacije. Kako unutar iste često postoje brojne nepoznanice i iskusnim programerima, korisnicima se savjetuje da se o svakoj nepoznanici detaljno posavjetuju preko raznih provjerenih platformi koje iste mogu pojasniti. Preporuka je kod izrade takvoga projekta raditi u timu od više ljudi. Tako se posao dijeli u skupine gdje svaka skupina/pojedinac ima određeni zadatak pa je pojedinac primoran dodatno istražiti i naučiti kako se radi sama implementacija i pronalaženje grešaka u vlastitome dijelu izrade projekta.

6.1 Mišljenje o izrađenoj e-trgovini

E-trgovina ima potencijal da prikaže novi način na koji ljudi kupuju putem interneta. S povećanom potražnjom za internetskom trgovinom, ova je aplikacija spremna za uspjeh na konkurentnom tržištu e-trgovine. Kako se sve više potrošača okreće internetskoj kupovini, potražnja za pouzdanim i korisnički prijateljskim platformama e-trgovine samo će nastaviti rasti. No cilj je zadržati korisničko iskustvo kojim se pokazuje da je prodavaču stalo do svakoga korisnika pojedinačno te da nije cijela kupovina odrađena robotski, virtualnim putem. Intuitivno sučelje, siguran sustav plaćanja i sveobuhvatni katalog proizvoda o kojemu vlasnik usluge vrlo lako upravlja, kupcima se nudi minimalističko iskustvo kupovine. Nadalje, snažan poslužiteljski sustav Stripea i Sanityja olakšava prodavaču upravljanje njihovom zalihom, obradu narudžbi, praćenje pošiljki te ostale mogućnosti. Neki ključni elementi ove aplikacije uključuju popise proizvoda s detaljnim opisima i slikama, košarice za kupnju koje omogućuju korisnicima da dodaju proizvode i pregledaju svoje stavke prije plaćanja te sigurnu obradu plaćanja kako bi se osigurale sigurne transakcije. Kako se stalno prilikom izrade spominjao minimalistički pristup problemu, radi se alternativan način kupnje naspram onoga koji je danas prisutan u većini slučajeva. E-trgovina nudi personalizirano iskustvo kupovine za kupce pa tako ne nudi mogućnost stvaranja korisničkih računa gdje kupci mogu pratiti povijest kupovina te podatke o plaćanju. Možda se to ne čini kao da je praktično za kupce, no u ovome primjeru traži se i određena sigurnost prema osjetljivim korisničkim podacima koji se koriste za novčane transakcije. Tako je sve osjetljive informacije i informacije o kupnji moguće slati korisniku na njegovu e-adresu odmah nakon kupnje. O tome također vlasnik usluga treba voditi minimalno brigu jer sve to odrađuje Stripe nakon konfiguracije.

Kako bi se zadržalo zadovoljstvo kupaca te njihova lojalnost, prioritet je direktna komunikacija sa samim kupcem putem e-adresa koje su iskazane nakon same kupnje. Tako se djelomično prikazuje korisniku da je prodavač tu za njega u slučaju dodatnih pitanja te korisnik ima osjećaj da se ne odvija sve virtualnim putem. Budući da je prioritet izrade za maloprodajne vrijednosti i proizvode, veliki broj stalnih korisnika nema fizički pristup do prodavača pa se to postavlja kao idealno rješenje prema takvim kupcima.

Iako je ta e-trgovina usmjerena prema običnome čovjeku kojemu je u interesu vlastita zarada, ista ima potencijala privući velike i male tvrtke koje se mogu baviti određenim vrstama preprodaje ili drugačijih oblika profitabilnoga korištenja preko izrađene aplikacije. Male tvrtke mogu imati koristi od niskih početnih troškova izrade aplikacije i pojednostavljenih procesa inicijalizacije poslovanja, dok veće tvrtke mogu iskoristiti napredne značajke i skalabilnost koje su opisane u ranijim poglavljima i potpoglavljima. Osim toga, globalna dostupnost Stripe i Sanity platformi čini aplikaciju idealnom platformom za tvrtke koje žele proširiti svoju bazu kupaca i ući na nova tržišta.

6.2 Naknadna nadogradnja sučelja

U završnim fazama, gdje je aplikacija gotova i dostupna za instalaciju na samoga poslužitelja, treba napraviti osvrt na njezin razvoj te samostalno okarakterizirati njezine nedostatke. Prema aplikaciji se gleda sa svrhom mogućnosti dorade postojećih funkcionalnosti ili dodavanja potpuno novih mogućnosti. U tome je slučaju osvrt rađen samostalno te je uočeno par stavki koje je moguće opcionalno dodati, a one ovise o krajnjemu načinu primjene svake izrađene aplikacije po tome principu. Ukoliko se drugi, budući korisnici ne budu poistovjetili s autorom i njegovim minimalističkim pristupom ka rješavanju zadatka ovoga rada, moguće je u samu aplikaciju dodati sučelje izrade vlastitoga računa, gdje će pojedini kupci i prodavač imati svoje račune. Svaka će vrsta računa od te dvije (kupac i prodavač) imati različite ovlasti i mogućnosti prikaza sučelja. Ukoliko pratimo iznad opisanu težnju gdje prodavač teži zadržavanju prave komunikacije koja se odvija u realnome vremenu sa svojim klijentima, moguće je također uvesti i opciju komunikacije putem poruka . Tako bi se korisnicima prikazala ikona prilikom koje bi se otvorio zasebni skočni prozor gdje bi se ta komunikacija odvijala. Nadalje, ukoliko se za potrebe većega prihoda bude radilo reklamiranje proizvoda i same aplikacije, moguće je uvesti kupone koji bi se generirali po posebnim prilikama te tako smanjili monotonost kod kupovine. Na taj se način intrigira kupce da budu u tijeku s akcijama i promocijama koji se odvijaju, a to vodi do veće količine aktualnih, ali i novih kupaca. S obzirom na to da je ovu aplikaciju moguće implementirati za više različitih usluga i u različitim situacijama, bilo bi poželjno promijeniti tok plaćanja koji je opisan u poglavlju 4. Tako je moguće pružiti korisnicima veću autonomnost i prilagodbu ka regiji u kojoj se korisnici nalaze. Moguće je uvesti i veći broj načina plaćanja. Kako se ne bi sve svelo na kartično plaćanje, moguće je uvesti i gotovinsko plaćanje pri preuzimanju. Kod takvoga je oblika plaćanja prije potrebno dodatno se savjetovati sa samim partnerom koji se bavi dostavom proizvoda.

BIBLIOGRAFIJA

[1*] Bhupendra Solanki: „Sanity Studio“, s interneta, https://golden.com/wiki/Sanity_(company)-AMM4KXZ, 15. lipnja 2023.

[2] Carlsen F.: „“Create a Sanity project“, s interneta, https://www.sanity.io/docs/create-a-sanity-project, 23. studeni 2022.

[3] Taofeek A.: „How to get started with Sanity CMS“, s interneta, https://www.section.io/engineering-education/getting-started-with-sanity-cms/, 23. studeni 2022.

[4] Carlsen F.: „Create a schema and configure Sanity Studio“,s interneta, https://www.sanity.io/docs/create-a-schema-and-configure-sanity-studio, 23. studeni 2022.

[5] Carlsen F.: „Shema types“, s interneta, https://www.sanity.io/docs/schema-types, 23. studeni 2022.

[6] Carlsen F.: „Sanity studio project configuration“, s interneta, https://www.sanity.io/docs/configuration, 23.studeni 2022.

[7] Vercel Inc.: „NextJS - The React Framework for the Web“, s interneta, https://nextjs.org/, 15.lipnja 2023.

[8] Lance S.: „Learn how to use getServerSideProps function“, s interneta, https://www.learnbestcoding.com/post/25/nextjs-how-to-use-getserversideprops, 12. prosinac 2022.

[9] Merced A.: „getInitialProps vs. getServerSideProps in Next.js“, s interneta, https://blog.logrocket.com/getinitialprops-vs-getserversideprops-nextjs/, 12. prosinca 2022.

[10] Thompson - Edwards O.: „How to Use File-Based Routing in Next.js“, s interneta, https://upmostly.com/next-js/how-to-use-file-based-routing-in-next-js, 14. prosinca 2022.

[11] Scott G.: „getStaticProps vs getServerSideProps for Next JS“, s interneta, https://www.ohmycrawl.com/getstaticprops-vs-getserversideprops/, 14. prosinca 2022.

[12] Walke J.: „React - The library for web and native user interfaces“, s interneta, https://react.dev/, 15. lipnja 2023.

[13] *React official documentation*, „Context“, s interneta, https://reactjs.org/docs/context.html, 21. prosinca 2022.

[14] Gilshaan J.: „React Native Hooks & How To Use useState and useEffect“, s interneta, https://gilshaan.medium.com/react-native-hooks-how-to-use-usestate-and-useeffect-3a10fd3e760c, 22. prosinac 2022.

[15] Pavlutin D.: „A Guide to React Context and useContext() Hook“, s interneta, https://dmitripavlutin.com/react-context-and-usecontext/, 22. prosinac 2022.

[16] Stripe Inc.: „Stripe - Payments infrastructure for the internet“, s interneta, https://stripe.com/en-hr, 15. lipnja 2023.

[17] Motola C.: „Embedding Stripe on website“, s interneta, https://spreadsimple.com/blog/embedding-stripe-on-website/, 13. siječnja 2023

[18] Asthana N.: „How to accept payment on your site by integrating Stripe Payments“, s interneta, https://medium.com/@nishantasthana/how-to-accept-payment-on-your-site-by-integrating-stripe-payments-617352fd7ffa, 13. siječnja 2023.

[19] *Stripe official documentation*: „Recurring pricing models“, s interneta, https://stripe.com/docs/products-prices/pricing-models, 15.siječanj 2023

[20] *Stripe official documentation*: „Creating custom payment flows“, s interneta, https://stripe.com/docs/payments/quickstart, 29.siječnja 2023.

[21] *Stripe official documentation*: „Prebuilt Checkout page using NextJS“, s interneta, https://stripe.com/docs/checkout/quickstart?client=next, 28.siječanj 2023.

[22] Netlify Corp.: „Netlify - Develop and deploy websites and apps in record time“, s interneta, https://www.netlify.com/, 15. lipnja 2023.

[23] Vercel Inc.: „About Us“, s interneta, https://vercel.com/about, 15. lipnja 2023.

[24] *Vercel official documentation*: „Introduction to Vercel“, s interneta, https://vercel.com/docs, 2.veljače 2023.

SAŽETAK

U ovom radu opisan je razvoj web aplikacije za internetsku trgovinu. Za poslužiteljski dio aplikacije korišteni su NextJS i Sanity studio, dok je React korišten za klijentski dio. NextJS pruža prednosti serverskog generiranja i prikaza na strani klijenta, ubrzavajući učitavanje stranica i poboljšavajući korisničko iskustvo. Intuitivno sučelje omogućuje jednostavno definiranje ruta i navigaciju između stranica. React optimizira performanse učitavanja sadržaja koristeći virtualni DOM i podržava koncept komponenti za organizaciju i održavanje koda. Stripe je integriran za plaćanje u aplikaciji, pružajući jednostavne načine plaćanja i automatsku obradu transakcija. Podržava razne načine plaćanja, uključujući Apple Pay/Google Pay i globalne kartične plaćanja, kao i mogućnost plaćanja gotovinom pri preuzimanju.

Ključne riječi – e trgovina, sustav naplate, Stripe, React, Next.js, Sanity

Abstract

This paper describes the development of a web application for online shopping. Next.js and Sanity Studio were used for the server-side of the application, while React was employed for the client-side. Next.js offers the advantages of server-side rendering and client-side rendering, resulting in faster page loading and improved user experience. Its intuitive interface enables easy route definition and seamless navigation between pages. React optimizes content loading performance through the use of a virtual DOM and utilizes a component-based approach for code organization and maintenance. Stripe integration was implemented for seamless payment processing in the web application, supporting various payment methods such as Apple Pay/Google Pay, global card payments, and cash-on-delivery. Automated transaction processing ensures a hassle-free payment experience for users.

Keywords – e commerce, payment system, Stripe, React, Next.js, Sanity