Razvoj web aplikacije za prodaju proizvoda iz odabranog OPG-a

Klanjac, Lucija

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

RAZVOJ WEB APLIKACIJE ZA PRODAJU PROIZVODA IZ ODABRANOG OPG-A

Rijeka, srpanj 2024.

Lucija Klanjac

0069089110

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

RAZVOJ WEB APLIKACIJE ZA PRODAJU PROIZVODA IZ ODABRANOG OPG-A

Mentor: Izv. prof. dr. sc. Marko Gulić

Rijeka, srpanj 2024.

Lucija Klanjac

0069089110

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: **Lucija Klanjac (0069089110)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Razvoj web aplikacije za prodaju proizvoda iz odabranog OPG-a / Development of a web application for the sale of products from the selected FFC**

Opis zadatka:

Razviti web aplikaciju za prodaju proizvoda iz odabranog OPG-a. 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 odabranih proizvoda. Aplikaciju treba izraditi upotrebom MERN Stack razvojnog okvira koji se sastoji od 4 tehnologije (MongoDB, Express, React, Node.js). Također, treba implementirati cjelovitu autentifikaciju unutar spomenute web aplikacije. Za vizualni dizajn web aplikacije koristiti neki od React UI knjižica.

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

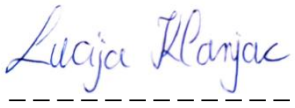
Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad.

Rijeka, srpanj 2024.



Lucija Klanjac

SADRŽAJ

1 UVOD

U današnjem digitalnom dobu, prisutnost na internetu postala je neizbježna i ključna za uspjeh u gotovo svim sektorima poslovanja. Potrošačke navike sve se više okreću *online* platformama, stoga je posjedovanje funkcionalnog *web shopa,* odnosno *web* trgovine, velika prednost, a često i nužnost za sve koji se bave prodajom. Ovaj završni rad predstavlja razvoj i implementaciju *web shop* aplikacije namijenjene vinariji In Sylvis, omogućavajući joj da na jednostavan i efikasan način predstavi svoje proizvode široj publici, olakša proces naručivanja i unaprijedi komunikaciju s kupcima. Važnost posjedovanja *web shop* aplikacija za OPG-ove poput vinarije In Sylvis je višestruka. *Web shop* im omogućava proširenje tržišta na regionalni, nacionalni pa čak i međunarodni nivo. Zatim, mogućnost direktne prodaje proizvoda povećava profitabilnost, a digitalna prisutnost pomaže u izgradnji brenda i povjerenja kod potrošača.

Korisnici ovog *web shopa* mogu navigacijom kroz aplikaciju saznati više o vinariji In Sylvis i vinima koje ona nudi. Također, imaju mogućnost dodavanja ili uklanjanja vina iz košarice. Ako se registriraju ili prijave u već postojeći račun, omogućena im je i kupnja odabranih proizvoda. Trenutno, aplikacija ne podržava elektroničko plaćanje, ali je planirano da se ta funkcionalnost implementira u budućnosti. Nakon kupnje proizvoda, korisnik dobiva potvrdu narudžbe. Dodatno, administrator *web shopa,* nakon prijave, ima pristup *admin* stranici aplikacije gdje može vidjeti sažetak poslovanja i pratiti napredak poslovanja. Na *admin* stranici, administrator može upravljati i uređivati proizvode, korisnike i narudžbe, što omogućuje efikasno i organizirano vođenje i optimizaciju svih aspekata poslovanja.

Ovaj *web shop* je *full stack* aplikacija napravljena korištenjem MERN Stacka i drugih tehnologija i biblioteka, kao što su Cloudinary, Tailwind CSS i drugi, koji će biti detaljnije opisani u sljedećem poglavlju.

Ovaj rad detaljno opisuje razvoj *web* aplikacije, uključujući korištene tehnologije, izgled same aplikacije i funkcionalnosti koje nudi. U 2. poglavlju „Tehnologije“ opisane su sve korištene tehnologije i biblioteke. Zatim je u 3. poglavlju „Opis rada same aplikacije“ detaljno opisana aplikacija sa svim njezinim funkcionalnostima. U 4. poglavlju „Opis funkcionalnosti“ opisane su dvije važne funkcionalnosti kao primjer kako aplikacija

zapravo radi, uz slike i objašnjenje koda te cijelog *frontend* i *backend* procesa. Zaključak je iznesen u posljednjem poglavlju.

2 TEHNOLOGIJE

Ovaj *web shop* je *full stack* aplikacija, što znači da uključuje razvoj kako klijentske strane (*frontend*) tako i poslužiteljske strane (*backend*) aplikacije. *Full stack* razvoj podrazumijeva korištenje različitih tehnologija i alata za izradu cjelokupnog sustava koji obuhvaća sve aspekte aplikacije, od korisničkog sučelja do baze podataka. Klijentska strana aplikacije zadužena je za interakciju s korisnicima i prikaz podataka, dok poslužiteljska strana obrađuje zahtjeve korisnika, upravlja podacima i komunicira s bazom podataka. Korištenje *full stack* pristupa omogućuje sveobuhvatan razvoj aplikacije, što rezultira boljom integracijom i efikasnošću cijelog sustava.

Za izradu ove *web* aplikacije korišten je tehnološki skup MERN Stack, koji čine četiri tehnologije: MongoDB [1], Express.js [2], React.js [3] i Node.js [4]. Na poslužiteljskoj strani korišteni su Node.js zajedno s njegovim okvirom Express.js za izradu servera i upravljanje rutama, te MongoDB za upravljanje bazom podataka. Na klijentskoj strani korišten je React.js za izgradnju dinamičkog korisničkog sučelja.

Osim MERN-a, za izradu su korištene i brojne druge tehnologije, biblioteke i alati. Na poslužiteljskoj strani korišteni su:

- bcryptjs [5] - biblioteka za enkripciju zaporki
- joi [6] - biblioteka za validaciju podataka
- jsonwebtoken [7] - biblioteka za rad s *JSON Web Tokenima (JWT)*
- cloudinary [8] - biblioteka za rad s Cloudinary servisom za upravljanje slikama i videozapisima
- dotenv [9] - biblioteka za učitavanje varijabli okoline iz *.env* datoteka

Na klijentskoj strani korišteni su:

- axios [10] - popularna biblioteka za *HTTP* zahtjeve koja omogućava jednostavnu komunikaciju s *backendom*
- jwt-decode [11] - biblioteka za dekodiranje *JSON Web Tokena (JWT)*
- moment [12] - biblioteka za rad s datumima i vremenom
- tailwindcss [13] - moderni *CSS* okvir za brzo kreiranje prilagodljivih korisničkih sučelja

- reduxjs/toolkit [14 -, službeni set alata za efikasniji rad s Reduxom koji pojednostavljuje konfiguraciju i upravljanje stanjem aplikacije

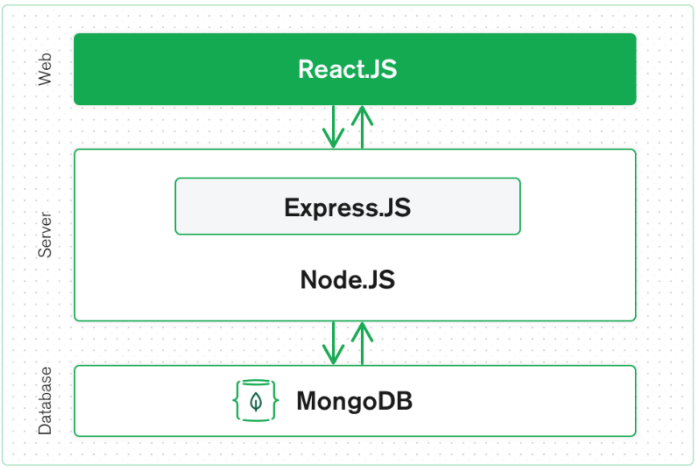
U nastavku će biti detaljnije objašnjeni najvažniji i najviše korišteni spomenuti alati, tehnologije i biblioteke.

2.1 MERN

MERN Stack je predefinirani skup tehnologija temeljen na JavaScriptu. Sastoji se od četiri tehnologije po kojima je i dobio naziv:

- MongoDB - dokumentna baza podataka
- Express(.js) - web okvir za Node.js
- React(.js) - klijentski JavaScript okvir
- Node(.js) - vodeći JavaScript web poslužitelj (*runtime*)

Jedna od ključnih prednosti MERN-a leži u sposobnosti pojednostavljenja i ubrzanja razvoja web aplikacija koristeći jedan zajednički jezik – JavaScript. S obzirom na to da se sve četiri komponente ovog *stacka* baziraju na JavaScriptu, programeri mogu raditi na svim aspektima aplikacije, od *frontenda* do *backenda*, bez potrebe za učenjem različitih programskih jezika. Na slici 2.1. prikazana je organizacija tih četiri tehnologije unutar MERN Stacka.



Slika 2.1. – Organizacija Mern Stacka [15]

MongoDB omogućuje fleksibilno i efikasno pohranjivanje podataka u *JSON* formatu, dok Express.js pojednostavljuje izradu robusnih *web* okvira. React.js pruža

moćne alate za izgradnju interaktivnih i odzivnih korisničkih sučelja, a Node.js osigurava visoke performanse i skalabilnost na serverskoj strani. Još jedna važna prednost MERN-a je to što je potpuno besplatan. Sve četiri komponente su *open-source* tehnologije, što znači da su slobodno dostupne za korištenje bez troškova licenciranja. Otvorenost zajednice koja razvija i održava ove tehnologije osigurava brz razvoj, redovita ažuriranja te obilje resursa i dokumentacije, što dodatno olakšava rad s MERN-om. Zahvaljujući ovim prednostima, MERN Stack postaje sve popularniji izbor među programerima za razvoj modernih web aplikacija.

2.1.1 React(.js)

Najviša razina MERN stoga je React, deklarativna, učinkovita i fleksibilna JavaScript biblioteka za izgradnju korisničkih sučelja [16]. React omogućuje kreiranje složenih korisničkih sučelja kroz jednostavne komponente i njihovo povezivanje u konačni *HTML*, *CSS* i JavaScript kod. Neke od prednosti React.js-a su sljedeće:

- Virtualni DOM (Document Object Model) - Virtualni DOM je koncept koji se koristi u Reactu, ali i u drugim modernim JavaScript okvirima i bibliotekama kako bi se poboljšala efikasnost ažuriranja korisničkog sučelja. Umjesto da direktno mijenja stvarni DOM, React kreira apstraktnu kopiju stvarnog DOM-a u memoriji, pod imenom virtualni DOM. Kada dođe do promjene u stanju ili podacima aplikacije, React uspoređuje novi virtualni DOM s prethodnim, identificira razlike i primjenjuje samo stvarne promjene na stvarni DOM. Ovaj pristup rezultira učinkovitijim ažuriranjem korisničkog sučelja, poboljšava performanse aplikacija i pojednostavljuje razvoj kompleksnih sučelja bez direktnog upravljanja stvarnim DOM-om.
- Jednostavno integriranje s drugim bibliotekama - React se lako može integrirati s drugim bibliotekama i okvirima kao što su Axios za slanje *HTTP* zahtjeva ili Redux za upravljanje stanjem aplikacije.
- Komponentna arhitektura - React se temelji na komponentnoj arhitekturi, što znači da se korisničko sučelje sastoji od neovisnih, ponovno iskoristivih i lako održivih komponenata. Svaka komponenta ima svoje stanje (*state*) i metode za upravljanje tim stanjem.
- Deklarativno programiranje - React koristi deklarativni pristup u definiranju korisničkog sučelja, što znači da se fokusira na opisivanje kako sučelje treba

izgledati u određenom stanju, a ne na detaljnom upravljanju svakom njegovom promjenom.

- Velika zajednica i podrška - React ima široku podršku i aktivnu zajednicu programera diljem svijeta. Dostupna je obilna dokumentacija, tečajevi te mnoge *open-source* biblioteke koje značajno olakšavaju razvoj aplikacija.

U ovom su radu korištene i sljedeće React biblioteke i paketi:

- Material UI [17] (icons-material, material, x-data-grid) - popularna React komponentna biblioteka za brzi razvoj korisničkog sučelja s materijalnim dizajnom i dodatnim ikonama i komponentama za rad s tablicama podataka
- react-dom [18] - paket koji pruža DOM-specifične metode koje olakšavaju rad s Reactom na webu
- react-icons [19] - biblioteka koja pruža popularne ikone kao React komponente
- react-redux [20] - vezni sloj za povezivanje Reacta i Reduxa
- react-router-dom [21] - biblioteka za upravljanje navigacijom i rutama u React aplikacijama
- react-toastify [22] - biblioteka za prikaz obavijesti (*toast*) u React aplikacijama
- Recharts [23] - komponenta za izradu grafikona u Reactu
- Semantic UI (css, react) [24] - *CSS* stilovi i React komponente za popularni okvir za dizajn korisničkog sučelja

2.1.2 Node(.js)

Node je besplatno, *open-source, cross-platform* JavaScript okruženje za izvršavanje koje omogućuje developerima kreiranje servera, *web* aplikacija, alata za naredbeni redak (eng. *command line tools*) i skripti [4]. Node.js je poznat po svojoj asinkronoj, događajem vođenoj arhitekturi koja je optimizirana za razvoj skalabilnih mrežnih aplikacija. Ova platforma koristi brzi *V8 JavaScript engine* koji omogućuje visoku izvedbu i efikasno upravljanje više istovremenih zahtjeva bez blokiranja, što značajno poboljšava responzivnost aplikacija i ubrzava njihov razvoj.

Node.js koristi JavaScript na obje strane aplikacije, što olakšava programerima upravljanje kompleksnošću projekata i dijeljenje koda. Veliki ekosustav paketa putem NPM-a (Node Package Manager) [25], najveće softverske knjižice na svijetu, pruža mnoga gotova rješenja i alate za brzu implementaciju funkcionalnosti u aplikacije.

Zbog svoje sposobnosti horizontalnog i vertikalnog skaliranja, Node.js je idealan izbor za razvoj modernih aplikacija koje zahtijevaju brzu reakciju na rastući broj zahtjeva ili korisnika.

Aktivna zajednica koja podržava Node.js neprestano dijeli znanje i pruža podršku, čineći ga pouzdanim i inovativnim okruženjem za izgradnju *web* aplikacija u današnjem digitalnom svijetu.

2.1.3 Express(.js)

Express je minimalni i fleksibilni Node.js web aplikacijski okvir koji pruža robusni skup značajki za web i mobilne aplikacije [2].

Poznat po svojoj jednostavnosti i fleksibilnosti, Express omogućuje brzo kreiranje serverskih aplikacija uz minimalan *overhead*. Zahvaljujući svojoj arhitekturi *middlewarea*, Express pojednostavljuje obradu zahtjeva, rutiranje, manipulaciju *HTTP* zahtjevima i odgovorima te integraciju s raznim template *engineima*. Express je popularan izbor među developerima zbog svoje jednostavnosti korištenja, dobre dokumentacije te velike i aktivne zajednice koja pruža mnogo dodataka (*middlewarea*) i paketa putem NPM-a.

2.1.4 MongoDB

MongoDB je *open-source* dokumentno orijentirana baza podataka dizajnirana za pohranu velikih količina podataka koja omogućuje vrlo efikasan rad s tim podacima. Klasificirana je kao NoSQL (Not only SQL) baza podataka jer pohrana i dohvaćanje podataka u MongoDB-u nisu organizirani u obliku tablica. Umjesto tablica, MongoDB koristi kolekcije. Omogućuje stvaranje više baza i kolekcija, pri čemu su kolekcije skupovi dokumenata. Dokumenti se sastoje od polja, odnosno parova ključeva i vrijednosti, pri čemu svaki dokument sadrži jedinstveni ID. Na slici 2.2. prikazan je primjer jednog dokumenta iz baze podataka, koji predstavlja jedan proizvod iz web trgovine.

Slika 2.2. – Primjer zapisa u MongoDB

Osim spomenute dokumentne orijentiranosti, ostale prednosti korištenja MongoDB-a su:

- Baza podataka bez sheme - MongoDB je baza podataka bez fiksne sheme, što znači da jedna kolekcija može sadržavati različite vrste dokumenata. Drugim riječima, u MongoDB bazi podataka jedna kolekcija može sadržavati više dokumenata koji se mogu razlikovati po broju polja, veličini i sadržaju. Nije obavezno da jedan dokument bude sličan drugom, kao što to zahtijevaju relacijske baze podataka. Zbog ove fleksibilnosti, MongoDB pruža veliku slobodu u organizaciji podataka.
- Indeksiranje - U MongoDB bazi podataka, svako polje u dokumentima je indeksirano primarnim i sekundarnim indeksima što olakšava i ubrzava dohvaćanje podataka iz velike količine podataka.
- Replikacija - MongoDB pruža visoku dostupnost i redundantnost pomoću replikacije, gdje se stvara više kopija podataka pa se onda te kopije šalju na različite poslužitelje. Zahvaljujući tome, ukoliko jedan poslužitelj zakaže, podaci se mogu dohvatiti s drugog poslužitelja.
- Skalabilnost - MongoDB pruža horizontalnu skalabilnost pomoću *shardinga*. *Sharding* podrazumijeva distribuciju podataka na više poslužitelja, pri čemu se velike količine podataka dijele na manje, brže i lakše upravljive podatkovne komade koristeći ključeve *shardinga*. Ti podatkovni komadi su ravnomjerno raspoređeni preko *shardova* koji se nalaze na mnogim fizičkim poslužiteljima.
- Visoka performansa - Performansa MongoDB-a je vrlo visoka i omogućuje brzu i učinkovitu trajnost podataka u usporedbi s drugim bazama podataka zahvaljujući funkcijama poput skalabilnosti, indeksiranja, replikacije i drugih.

2.2 Cloudinary

Cloudinary je *end-to-end* rješenje za upravljanje slikama i videozapisima za *web* stranice i mobilne aplikacije, pokrivajući sve od prijenosa slika i videozapisa, pohrane, manipulacija, optimizacija do isporuke [26]. Ova platforma podržava automatsko skaliranje, prilagođavanje i optimizaciju medijskih sadržaja kako bi se osigurala visoka kvaliteta i brza isporuka, bez obzira na uređaj ili mrežnu vezu. Cloudinary također nudi bogat skup *API-ja* i integracija s popularnim razvojnim okvirima i *CMS* sustavima što ga fleksibilnim alatom za razne potrebe. Zahvaljujući naprednim mogućnostima prepoznavanja slika, dodavanja filtara i transformacija u stvarnom vremenu, Cloudinary pomaže poboljšati korisničko iskustvo i optimizirati performanse *web* stranica i aplikacija.

2.3 Tailwind CSS

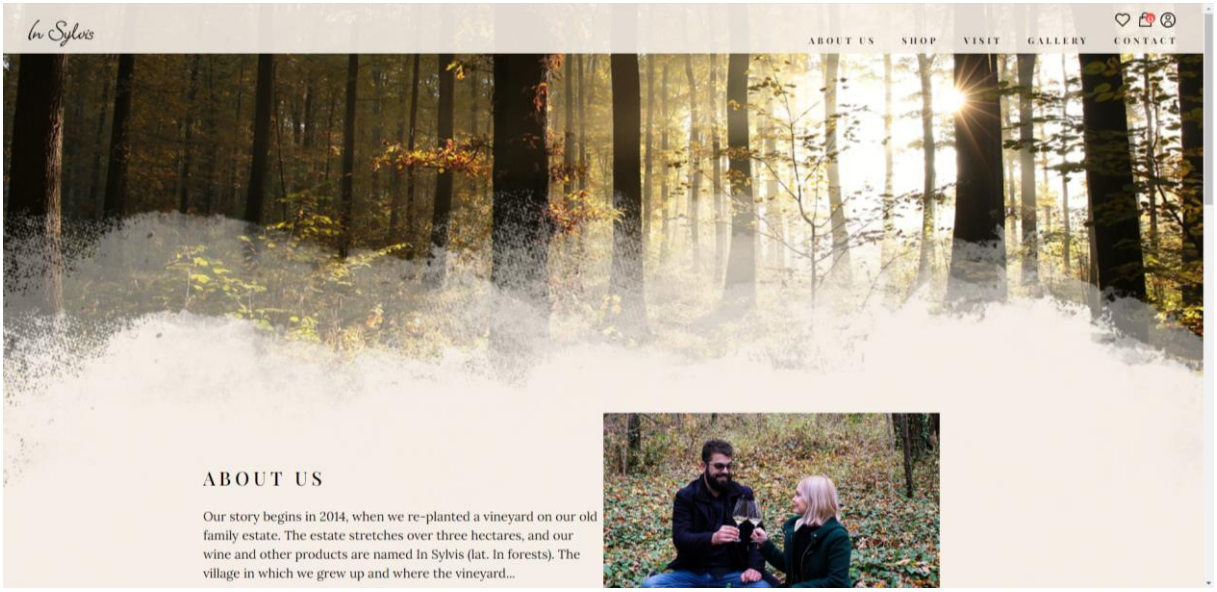
Tailwind CSS je *CSS* okvir koji se temelji na upotrebi alatnih klasa i unapređuje razvoj *web* stranica pružanjem širokog spektra unaprijed definiranih alatnih klasa. Ove klase pojednostavljuju proces stiliziranja bez potrebe za pisanjem prilagođenog *CSS*-a, osiguravajući dosljednost i skalabilnost. Tailwind CSS se razlikuje od tradicionalnih *CSS* okvira jer stavlja fokus na korištenje funkcionalnih klasa umjesto konvencionalnih komponenata, što omogućuje programerima učinkovito kreiranje responzivnih i vizualno atraktivnih sučelja uz minimalan napor.

3 OPIS RADA SAME APLIKACIJE

U ovom poglavlju detaljno je opisan rad same aplikacije kroz perspektivu korisnika i administratora. Ističu se ključne funkcionalnosti koje aplikacija nudi te načini na koje korisnici i administrator koriste te funkcionalnosti.

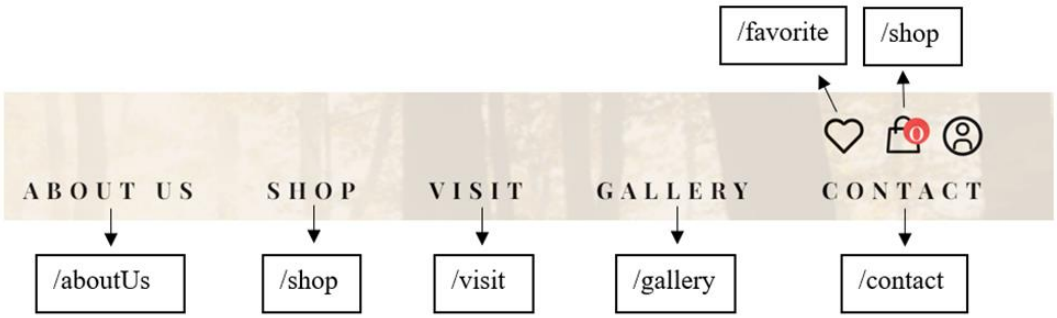
3.1 Početna stranica

Kada korisnik pristupi aplikaciji putem poveznice, prvo što vidi je početna stranica (*Home Page*). Početna stranica sastoji se od navigacijske trake i glavnog dijela stranice. Na slici 3.1. prikazan je izgled početne stranice.



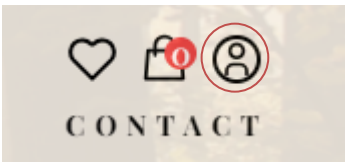
Slika 3.1. – Izgled početne stranice

Navigacijska traka sadrži logotip vinarije u lijevom kutu, koji istovremeno služi kao poveznica za povratak na početnu stranicu, te poveznice na desnoj strani. Poveznice na desnoj strani vode redom na različite stranice, kao što je prikazano na slici 3.2. Na primjer, klikom na poveznicu *SHOP*, korisnik će biti preusmjeren na *URL* adresu */shop,* odnosno stranicu na kojoj su izlistani svi proizvodi dostupni za kupnju.



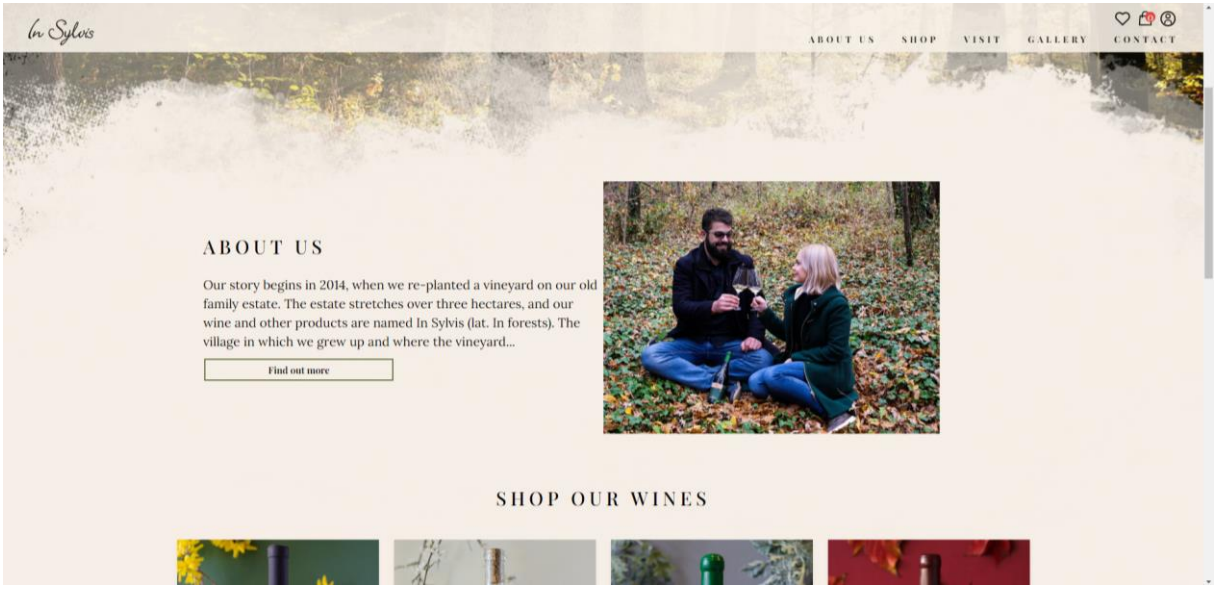
Slika 3.2. – Poveznice navigacijske trake

Klik na ikonu u krajnjem desnom kutu odjavljuje korisnika ako je prethodno bio prijavljen, a ako nije, preusmjeruje ga na *URL* adresu */register*. Na slici 3.3. crvenom je bojom zaokružena spomenuta ikona.



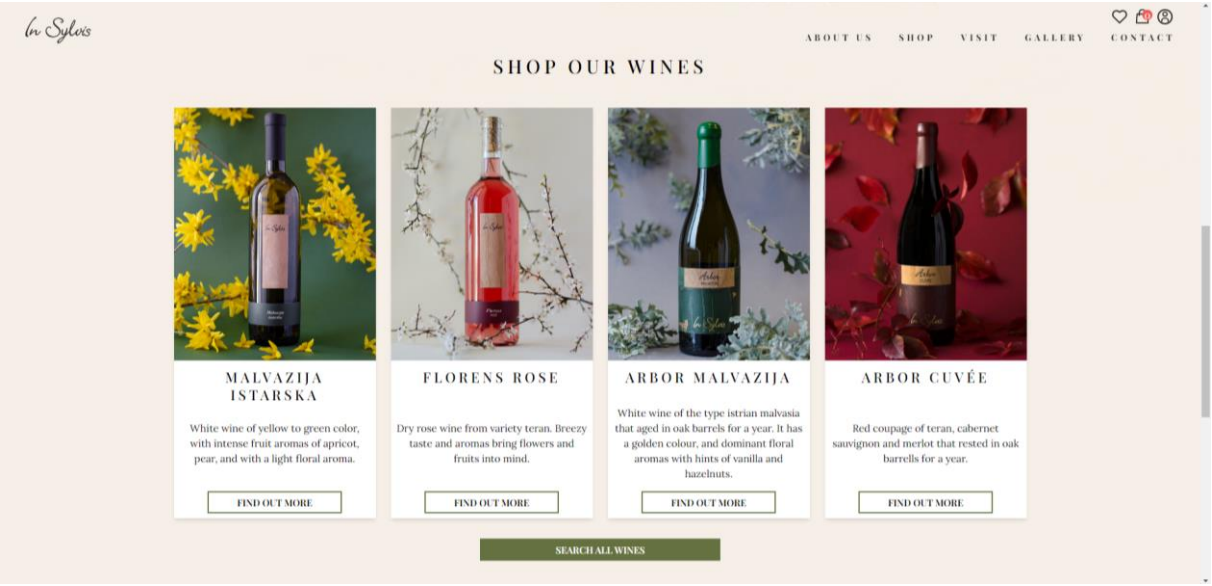
Slika 3.3. – Poveznica na URL adresu /register

Pomicanjem prema dolje, korisnik može vidjeti nekoliko dijelova početne stranice. Prvo što će vidjeti je dio *ABOUT US* u kojem je ukratko predstavljena vinarija InSylvis. Na slici 3.4. prikazan je *ABOUT US* dio. Klikom na gumb *Find out more*, korisnik je preusmjeren na *URL* adresu */aboutUs* gdje može saznati više o vinariji.



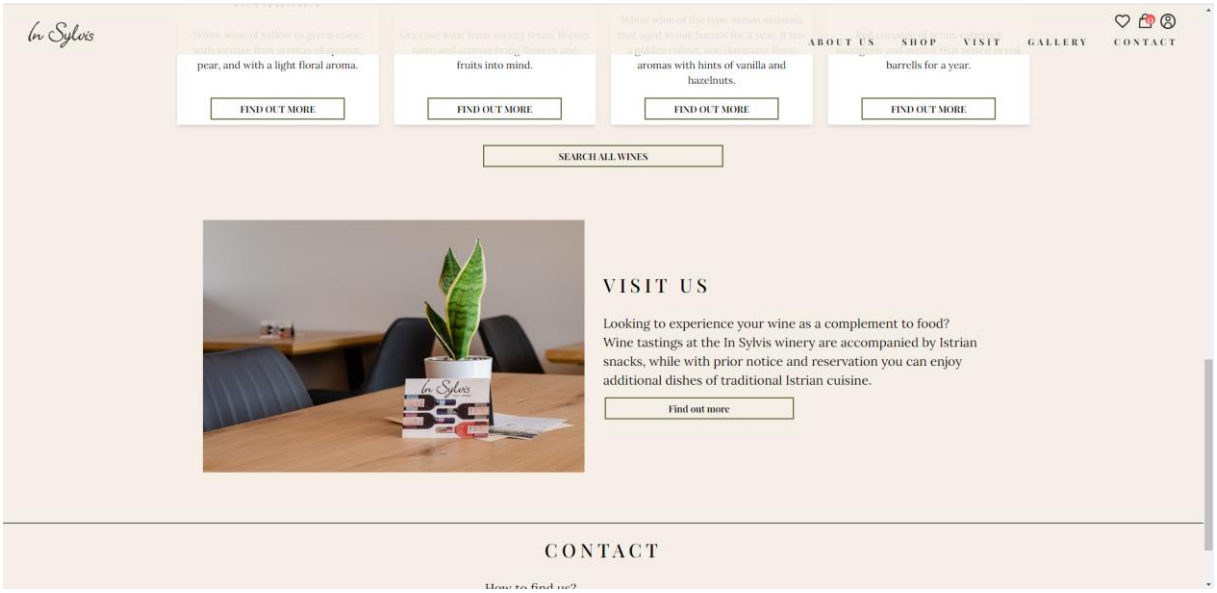
Slika 3.4. – About us dio

Nakon dijela *About us*, korisnik može vidjeti sekciju *SHOP OUR WINES* u kojoj su ukratko predstavljena vina. Klikom na gumb *SEARCH ALL WINES*, korisnik je preusmjeren na *URL* adresu */shop* gdje su detaljnije predstavljena sva vina. Uz svako vino nalazi se i gumb *FIND OUT MORE* koji vodi na detaljniji prikaz tog konkretnog vina s mogućnošću njegovog dodavanja u košaricu. Na slici 3.5. je prikazan je *SHOP OUR WINES* dio.



Slika 3.5. – Shop our wines dio

Daljnjim pomicanjem prema dolje, korisnik može vidjeti dio *VISIT US* u kojem se korisnika poziva da posjeti vinariju te klikom na gumb *Find out more*, sazna više o tome kako to i učiniti. Na slici 3.6. prikazan je *VISIT US* dio.



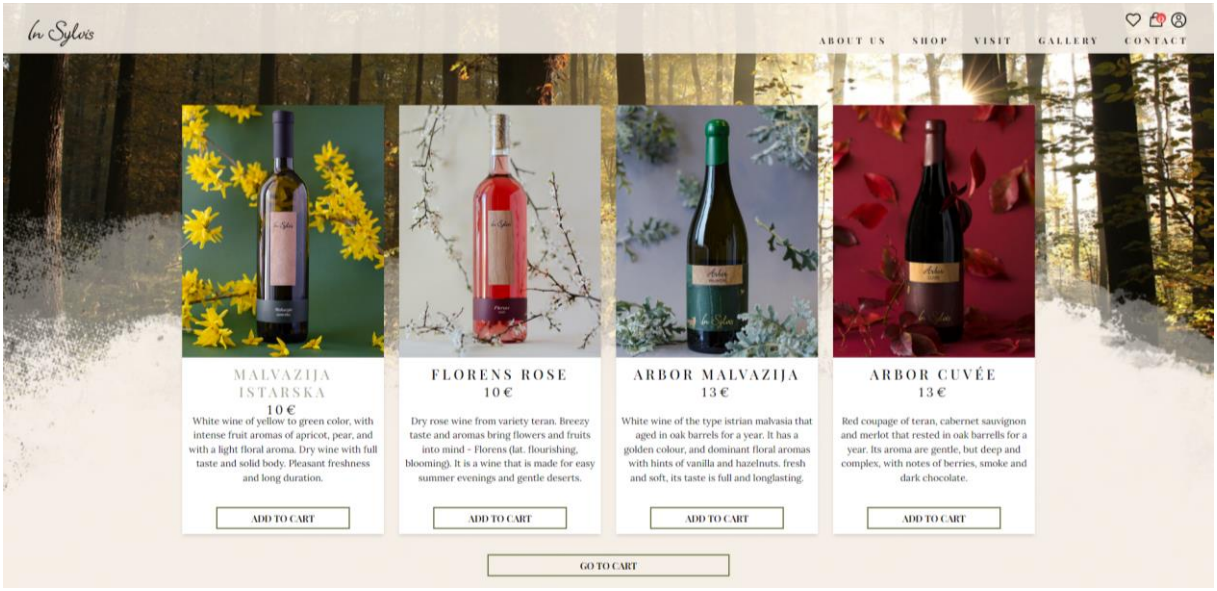
Slika 3.6. – Visit us dio

Na dnu početnog zaslona nalazi se dio s kontaktnim informacijama: poveznica koja vodi na Google Maps navigaciju do vinarije, *e-mail* adresa vinarije, te brojevi telefona vlasnika vinarije na koje se može kliknuti i odmah nazvati. Na slici 3.7. prikazane su kontaktne informacije na dnu početnog zaslona.

Slika 3.7. – Kontaktne informacije

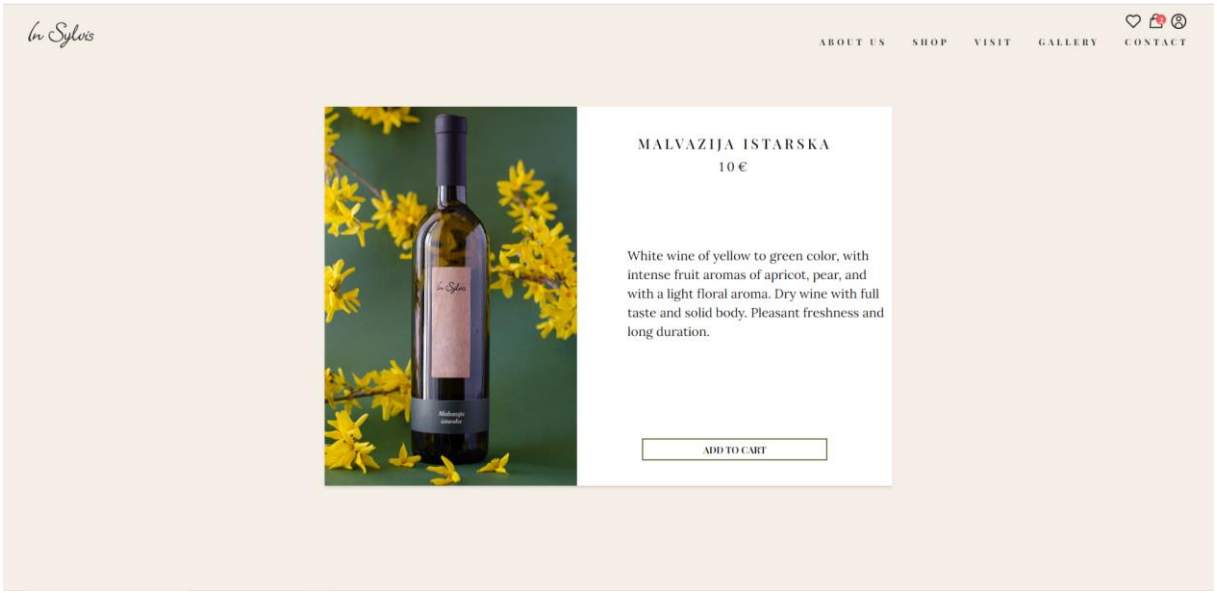
3.2 Trgovina

Trgovina koja se nalazi na *URL* adresi */shop* najvažniji je dio ove *web* aplikacije (slika 3.8.). Prikazuje cijelu ponudu vina, a svako je vino opisano slikom, nazivom, cijenom i opisom.



Slika 3.8. – Izgled trgovine

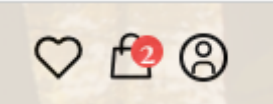
Klikom na naziv vina, korisnik može detaljnije proučiti to konkretno vino kao što je prikazano na slici 3.9.



Slika 3.9. – Detaljniji prikaz vina

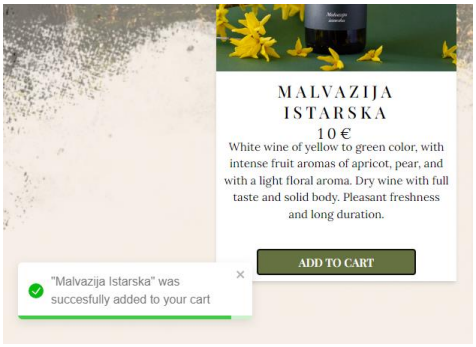
Klikom na gumb *ADD TO CART*, korisnik može dodati to vino u košaricu. Kada korisnik klikne na gumb *ADD TO CART*, događa se nekoliko stvari:

- Vino se dodaje u košaricu.
- Broj artikala u košarici se povećava. Na slici 3.10. vidljiv je uvećani broj artikala koji se nalazi unutar crvenog kružića.



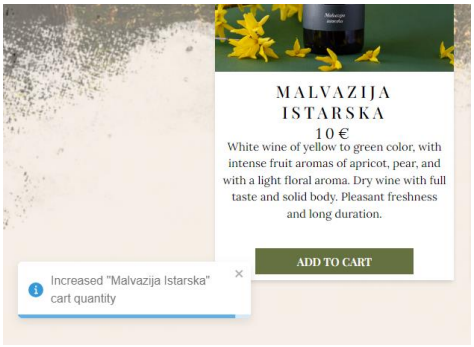
Slika 3.10. –Broj artikala u košarici se povećava

- Korisnik dobiva poruku s potvrdom uspješnog dodavanja vina u košaricu. Na slici 3.11. vidljiva je zelena poruka koja se pojavljuje na ekranu ukoliko je ta vrsta vina po prvi put dodana u košaricu.



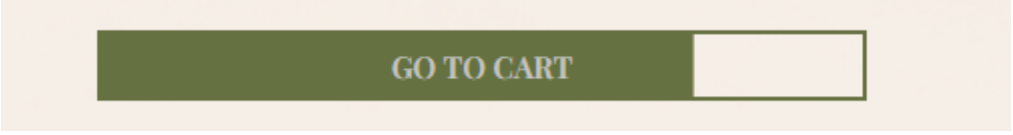
Slika 3.11. – Obavijest o uspješnom dodavanju vina u košaricu

Na slici 3.12. vidljiva je plava poruka koja se pojavljuje na ekranu ukoliko je ta vrsta vina već u košarici. Poruka obavještava korisnika da je broj vina te vrste u košarici uvećan.



Slika 3.12. – Obavijest o uspješnom povećanju količine vina u košarici

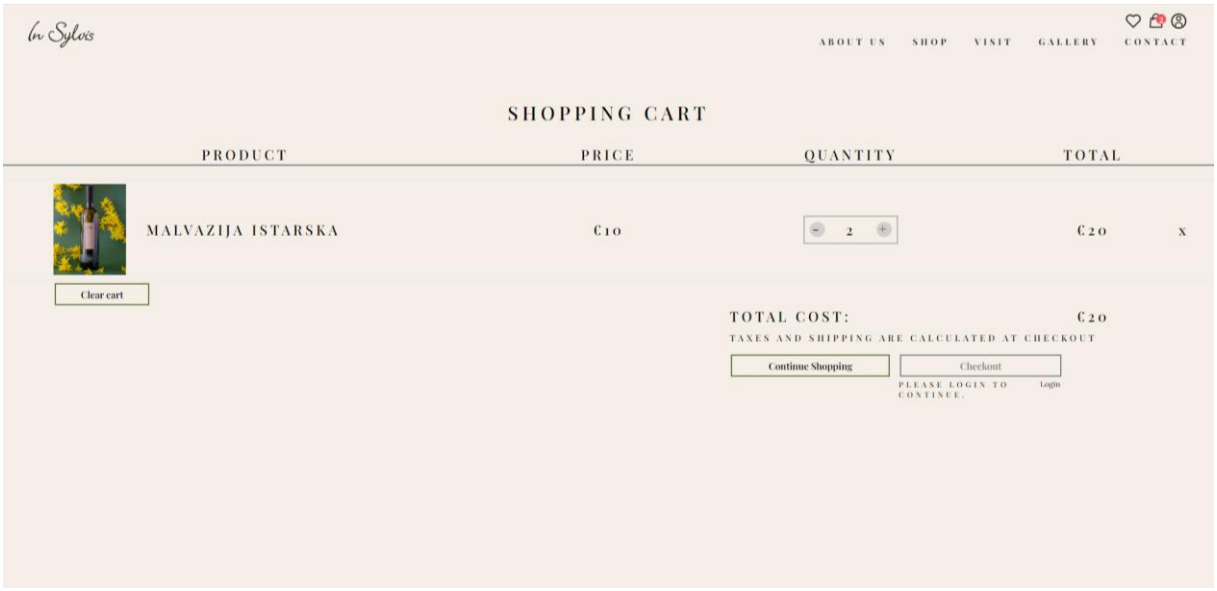
Slika 3.13. prikazuje *GO TO CART* gumb. Gumb se nalazi na dnu trgovine i korisnika vodi do košarice.



Slika 3.13. – Go to cart gumb

3.3 Košarica

Košarica se nalazi na *URL* adresi */cart*. Na slici 3.14. prikazan je izgled košarice. U košarici su unutar tablice prikazana vina koja je korisnik odabrao. Za svako vino prikazana je njegova slika, ime, cijena, odabrana količina i ukupni iznos (umnožak cijene i odabrane količine). Svako vino se može maknuti iz košarice klikom na gumb *x*, a količina se može podešavati klikom na gumbove - ili +. Na dnu tablice prikazan je ukupan iznos košarice, odnosno zbroj iznosa za svako vino.

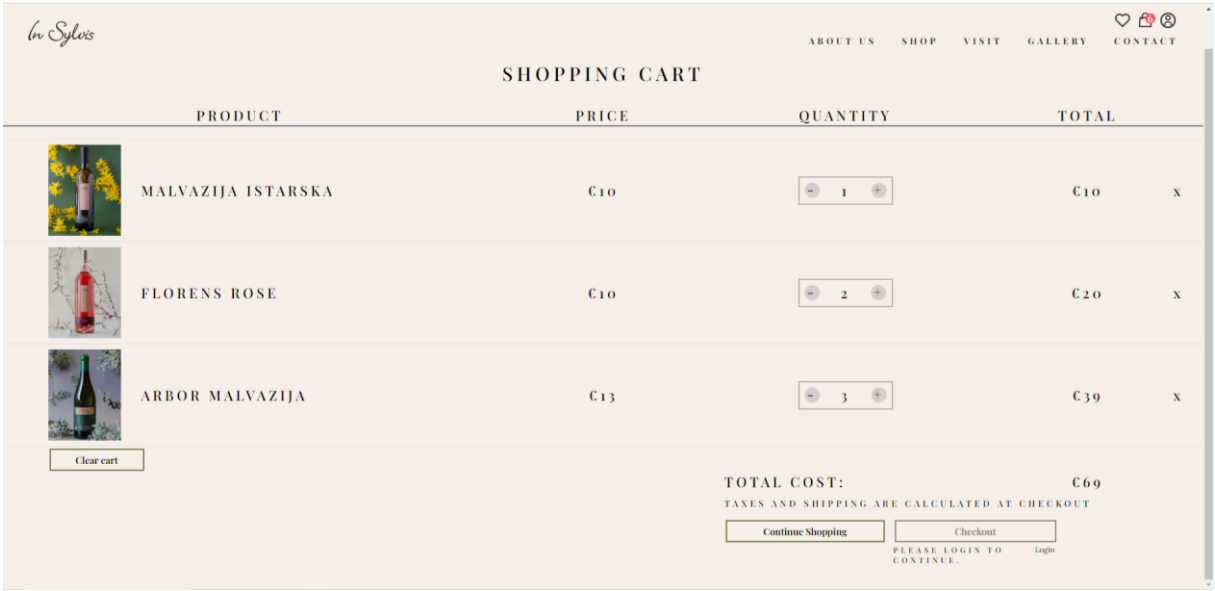


Slika 3.14. – Izgled košarice

Klikom na gumb *Clear cart*, korisnik može ukloniti sva vina iz košarice. U tom se trenutku na zaslonu pojavljuje poruka *Cart cleared* koja potvrđuje da je košarica ispražnjena, a zaslon izgleda kao na slici 3.15. Na zaslonu se nalazi poruka koja upućuje na to da je košarica prazna te gumb *start shopping* koji vodi korisnika natrag u trgovinu.

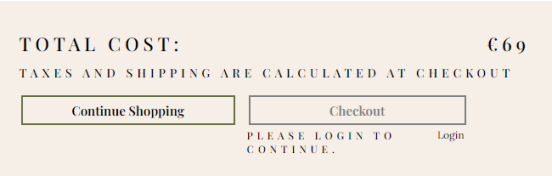
Slika 3.15. – Izgled prazne košarice

Na slici 3.16. vidljiv je ekran nakon što je korisnik ponovno dodao željena vina u košaricu. Na dnu trgovine nude se dvije opcije: klikom na gumb *Continue shopping* korisnik može nastaviti kupnju tako da se vraća natrag u trgovinu, a klikom na gumb *Checkout* može nastaviti s kupnjom odabranih proizvoda.



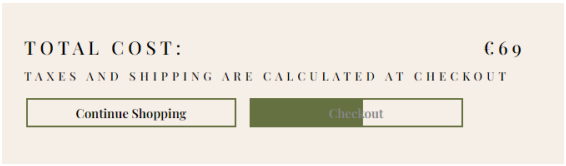
Slika 3.16. – Izgled pune košarice

Na slici 3.17. prikazan je onemogućeni gumb *Checkout*. Gumb će biti onemogućen dok god korisnik nije prijavljen.



Slika 3.17. – Checkout gumb je blokiran jer korisnik nije prijavljen

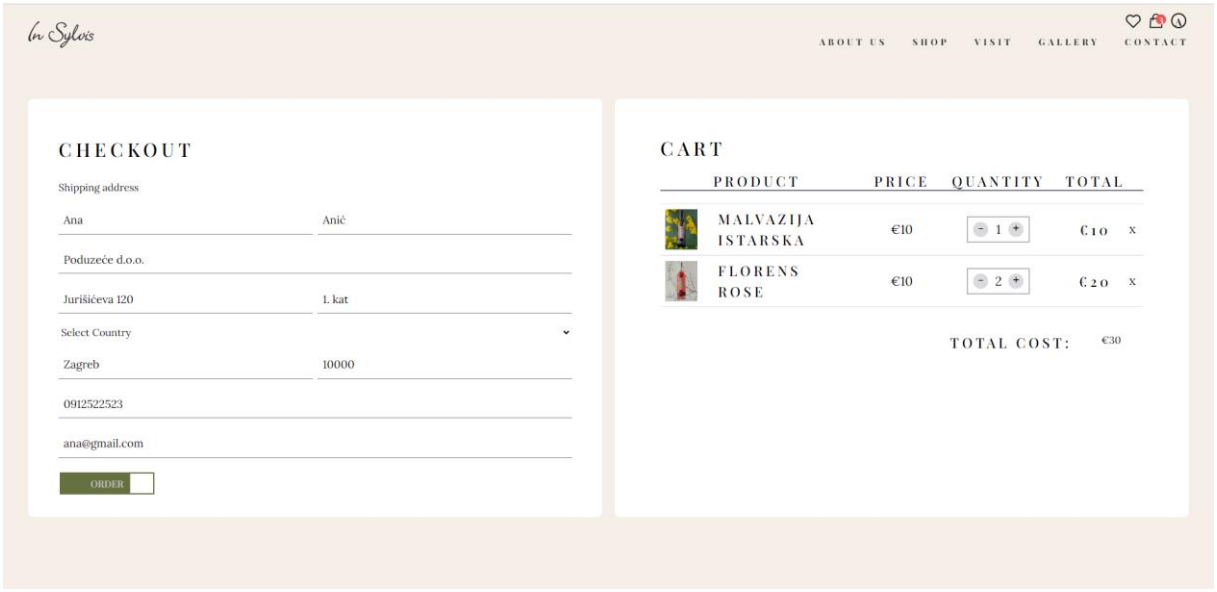
Nakon prijave, gumb je moguće kliknuti te on vodi na *URL* adresu */checkout*. Na slici 3.18. prikazan je gumb na koji je moguće kliknuti.



Slika 3.18. – Checkout gumb je moguće kliknuti jer je korisnik prijavljen

3.4 Završetak kupovine

Na *URL* adresi */checkout* nalazi se završetak kupovine (*checkout*), gdje korisnik može unijeti informacije o adresi za dostavu s lijeve strane ekrana, dok na desnoj strani može pregledati odabrane proizvode. Također, korisnik može mijenjati količinu proizvoda ili ih u potpunosti ukloniti iz košarice, jednako kao u prethodnom koraku. Na slici 3.19. prikazan je izgled završetka kupovine.



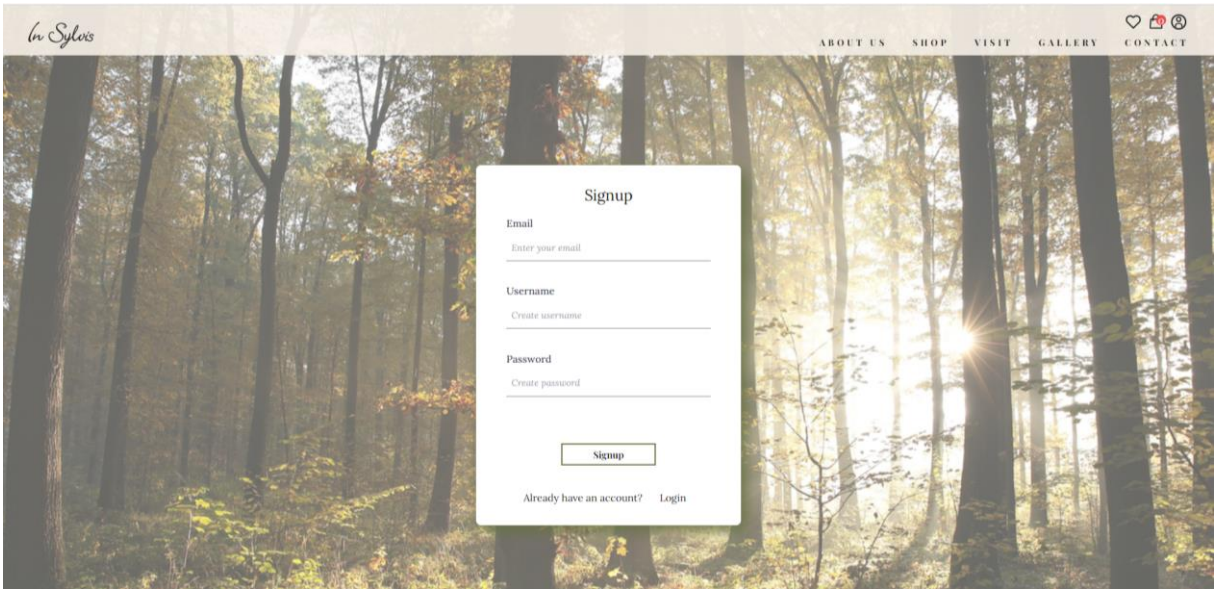
Slika 3.19. – Izgled završetka kupovine

Klikom na gumb *ORDER* korisnik je usmjeren na stranicu */checkout-success* na kojoj su prikazani detalji uspješne narudžbe. Na slici 3.20. prikazan je primjer potvrde narudžbe i poruke u gornjem desnom kutu koja potvrđuje uspješnost narudžbe. Potvrda narudžbe sadrži broj narudžbe, popis naručenih proizvoda, cijenu naručenih proizvoda te podatke za slanje.

Slika 3.20. – Izgled potvrde narudžbe

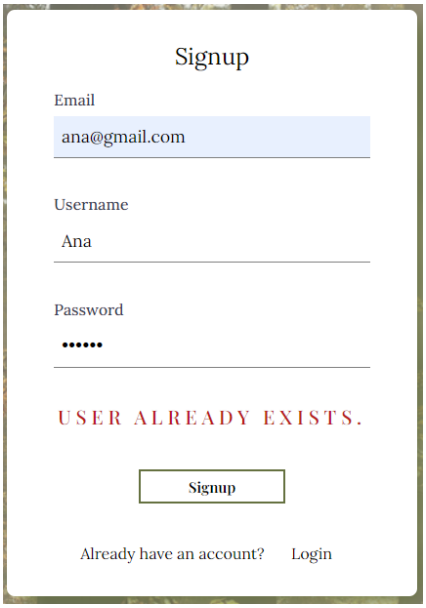
3.5 Registracija

Na slici 3.21. vidljiv je izgled ekrana prilikom registracije na *URL* adresi */register*. Prikazan je obrazac za registraciju korisnika koji zahtijeva unos *e-mail* adrese, korisničkog imena i zaporke. Klikom na gumb *Signup*, korisnik se registrira i preusmjerava na početnu stranicu *web shopa*. Ako korisnik već ima račun, može kliknuti na gumb *Login* kako bi bio preusmjeren na stranicu */login*.



Slika 3.21. – Izgled stranice za registraciju

U slučaju da već postoji korisnik s istom *e-mail* adresom, pojavit će se poruka upozorenja. Potrebno je unijeti važeću *e-mail* adresu, a zaporka mora sadržavati barem 6 znakova. U suprotnom, registracija će biti onemogućena. Na slici 3.22. vidljiva je poruka upozorenja prilikom pokušaja registracije korisnika koji već postoji.

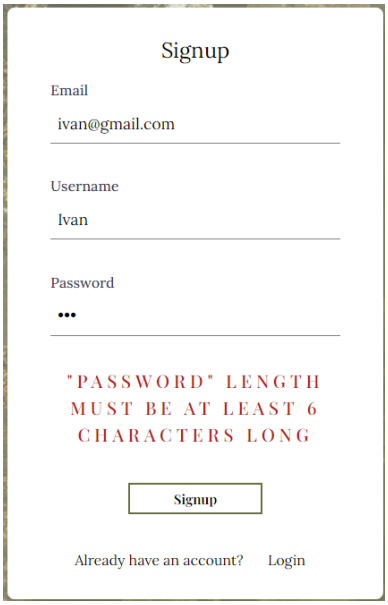


Slika 3.22. – Poruka upozorenja prilikom pokušaja registracije korisnika koji već postoji

Slika 3.23. prikazuje poruku upozorenja prilikom unosa nevažeće e-mail adrese.

Slika 3.23. – Poruka upozorenja prilikom unosa nevažeće e-mail adrese

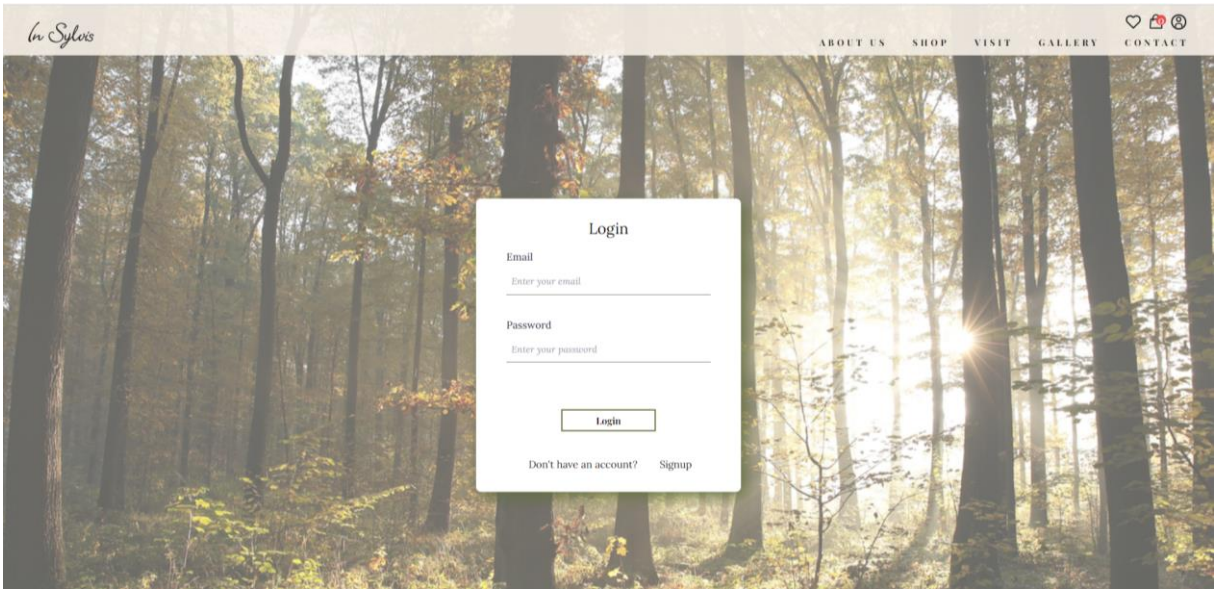
Na slici 3.24. vidljiva je poruka upozorenja prilikom unosa zaporke koja sadrži manje od 6 znakova.



Slika 3.24. – Poruka upozorenja prilikom unosa zaporke koja sadrži manje od 6 znakova

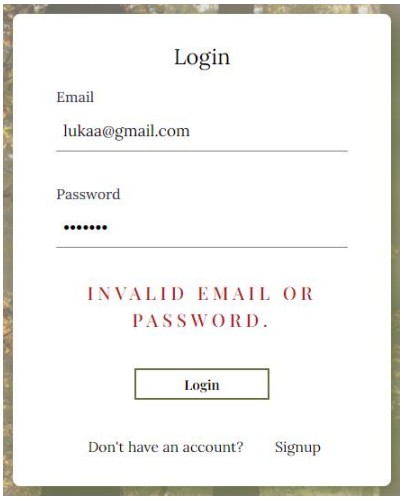
3.6 Prijava

Ukoliko korisnik već ima račun, može se prijaviti na *URL* adresi */login*, gdje se nalazi obrazac za prijavu korisnika koji zahtijeva unos *e-mail* adrese i zaporke. Na slici 3.25. prikazan je izgled stranice prilikom prijave korisnika. Klikom na gumb *Login*, korisnik se prijavljuje i preusmjerava na početnu stranicu. Na dnu obrasca nalazi se i poveznica *Signup* za one koji još uvijek nemaju račun.



Slika 3.25. – Izgled stranice za prijavu

U slučaju unosa krive *e-mail* adrese ili zaporke, pojavit će se poruka upozorenja i prijava neće biti uspješna. Slika 3.26. prikazuje poruku upozorenja prilikom unosa krive *e-mail* adrese ili zaporke.



Slika 3.26. – Poruka upozorenja prilikom unosa krive e-mail adrese ili zaporke

3.7 Stranica za administratore

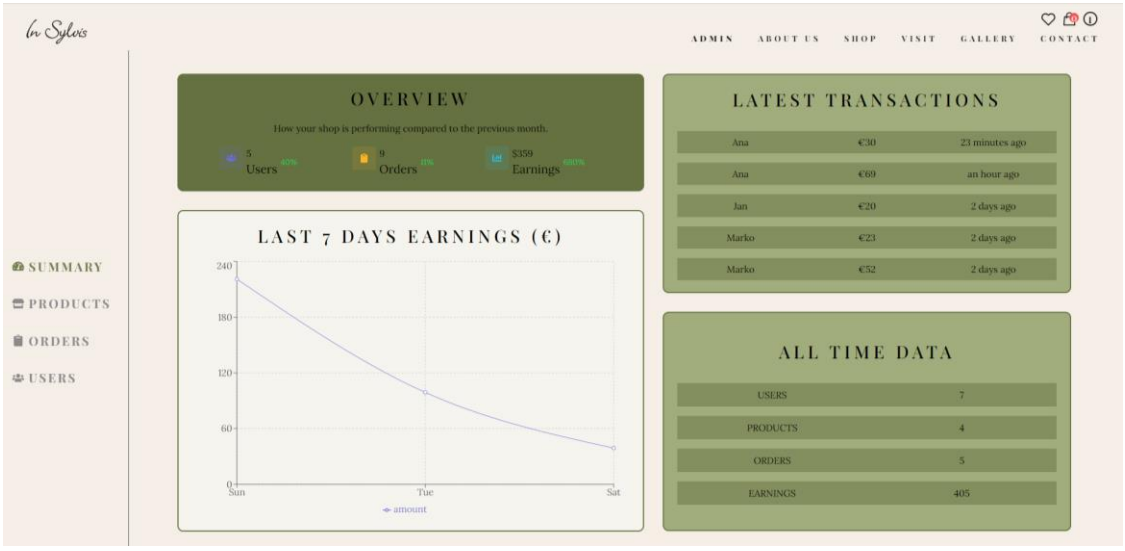
Nakon prijave, administratoru su vidljivi svi sadržaji kao i ostalim korisnicima, ali dodatno ima pristup administratorskoj strani. Na slici 3.27. prikazana je navigacijska traka kakvu vidi administrator. Administratorska stranica dostupna je putem poveznice *ADMIN* koja se nalazi u navigacijskoj traci i nije dostupan običnim korisnicima.

Slika 3.27. – Navigacijska traka vidljiva administratoru

Klikom na poveznicu *ADMIN*, administrator je preusmjeren na stranicu s *URL* adresom */admin/summary*. Uz lijevi rub ekrana vidljiva je bočna navigacijska traka koja sadrži poveznice do različitih URL adresa: *SUMMARY* vodi na */admin/summary*, *PRODUCTS* na */admin/products*, *ORDERS* na */admin/orders* i *USERS* na */admin/users*.

3.7.1 Sažetak

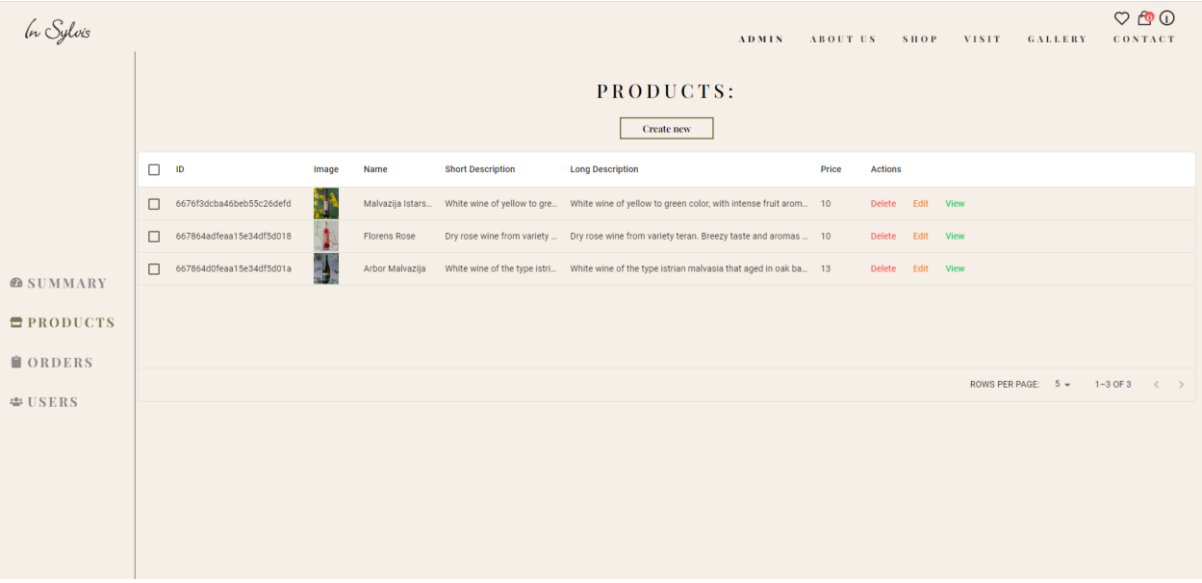
Slika 3.28. prikazuje sažetak napretka trgovine koji se nalazi na *URL* adresi */admin/summary*. Na vrhu sažetka nalazi se pregled napretka trgovine u odnosu na prethodni mjesec. Pregled prikazuje broj registriranih korisnika, broj naručenih narudžbi, ostvarenu zaradu te postotke koji pokazuju koliki je napredak ostvaren. Ispod pregleda nalazi se grafički prikaz zarade u posljednjih 7 dana, iskazan u eurima. Na desnoj strani sažetka prikazane su dvije tablice: gornja tablica koja prikazuje zadnjih 5 narudžbi te donja tablica koja prikazuje trenutne podatke, uključujući broj registriranih korisnika, broj proizvoda u ponudi, broj uspješnih narudžbi te ukupnu zaradu.



Slika 3.28. – Izgled sažetka napretka trgovine

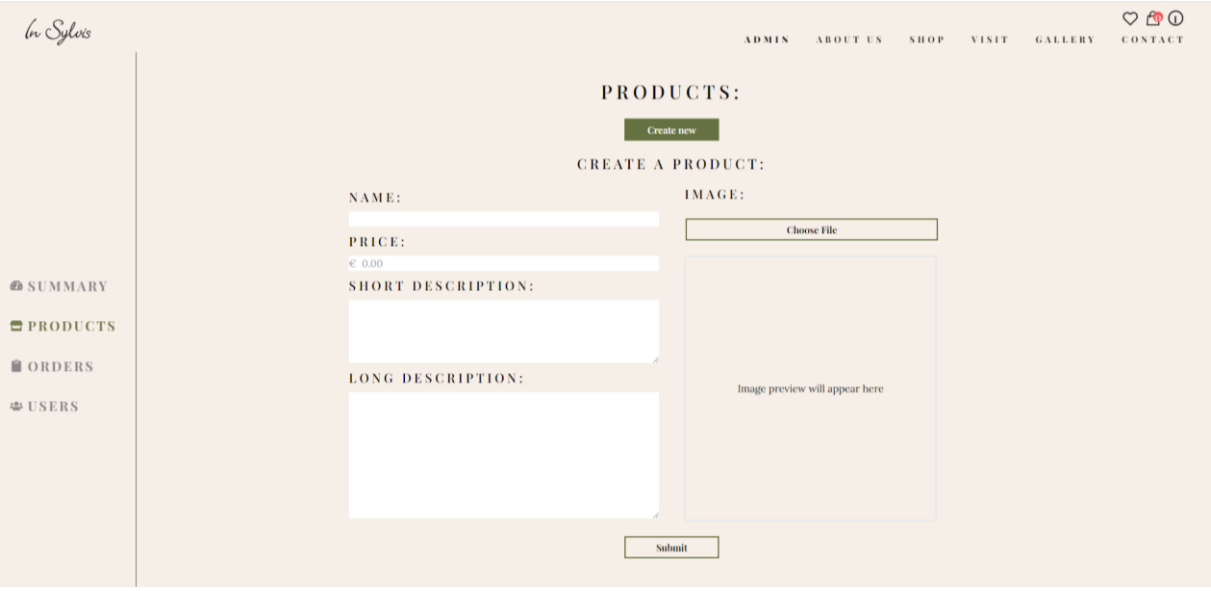
3.7.2 Proizvodi

Slika 3.29. prikazuje stranicu koja se otvara pritiskom na poveznicu *PRODUCTS*. Stranica se nalazi na *URL* adresi *admin/products* te sadrži gumb *Create new* i tablicu svih proizvoda koji su trenutno u ponudi.



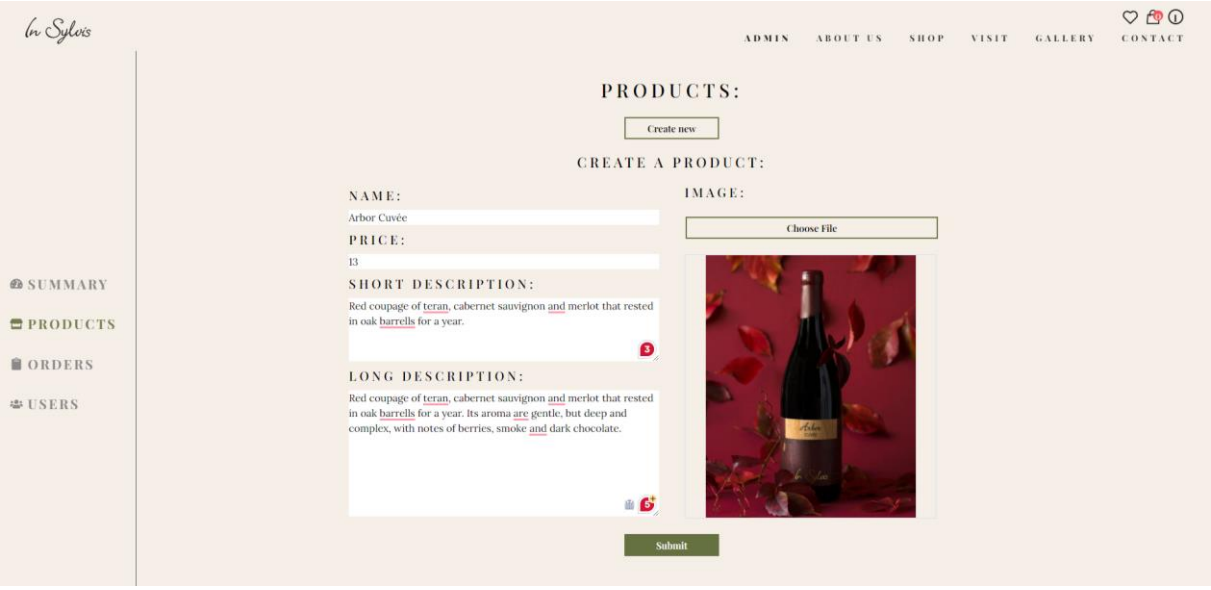
Slika 3.29. – Izgled stranice Proizvodi

Klikom na gumb *Create new* administrator je usmjeren na *URL* adresu *admin/products/create-product* gdje ima mogućnost dodati novi proizvod u trgovinu. Na slici 3.30. je vidljiv izgled te stranice.



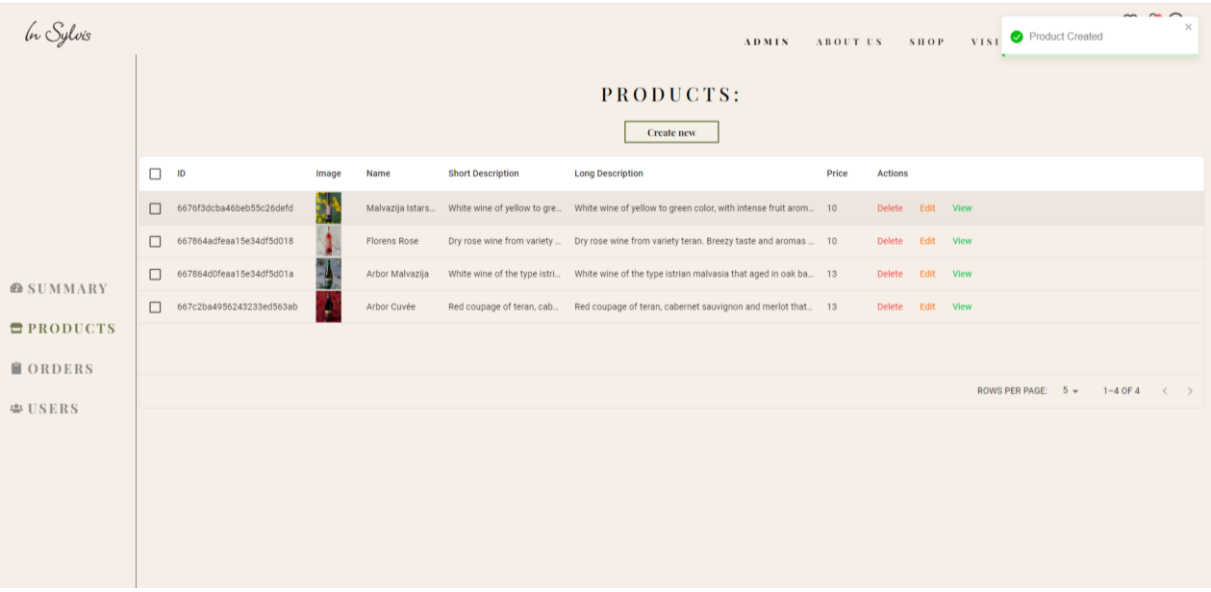
Slika 3.30. – Izgled stranice koja omogućuje administratoru dodavanje novog proizvoda u trgovinu

Slika 3.31. prikazuje primjer unosa podataka o vinu Arbor Cuvée. Administrator je unio ime, cijenu, kratki opis te dugi opis vina. Također, dodao je i sliku vina sa svog računala.



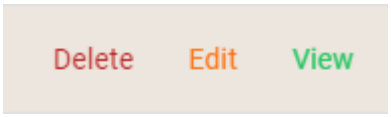
Slika 3.31. – Primjer unosa podataka o vinu Arbor Cuvée

Nakon unosa podataka o proizvodu te dodavanja slike sa svog računala, klikom na *Submit* administrator može dodati proizvod. Slika 3.32. prikazuje ekran nakon uspješnog dodavanja proizvoda. U gornjem desnom kutu vidi se poruka *Product Created* koja potvrđuje da je proizvod dodan u trgovinu, a vino Arbor Cuvée je sada vidljivo i u tablici.



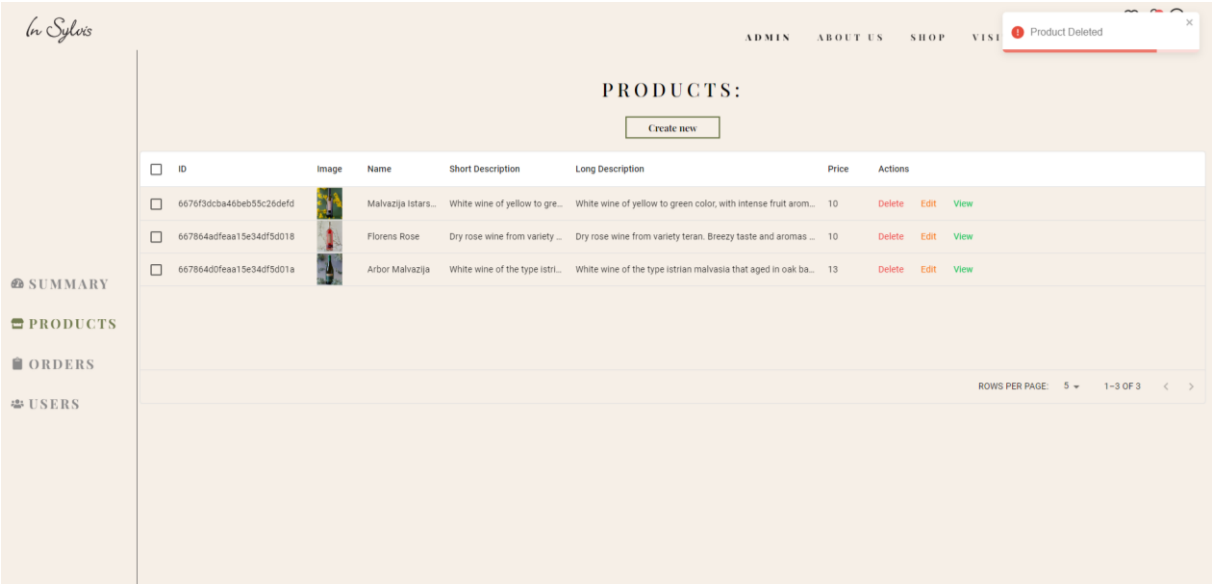
Slika 3.32. – Vino Arbor Cuvée je uspješno dodano

Tablica proizvoda prikazuje ID, sliku, ime, kratki opis, duži opis i cijenu svakog proizvoda. Također, uz svaki proizvod nalaze se akcijski gumbovi *Delete*, *Edit* i *View*. Na slici 3.33. prikazani su spomenuti gumbovi.



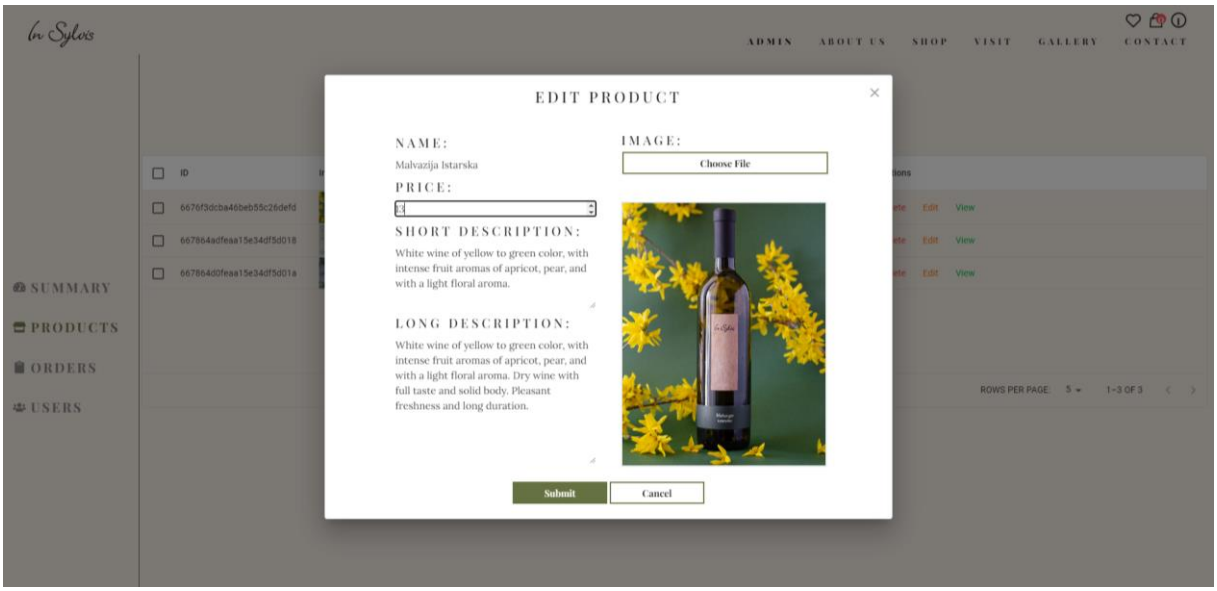
Slika 3.33. – Akcijski gumbovi

Klikom na gumb *Delete*, odabrani proizvod se briše iz trgovine i tablice. Slika 3.34. prikazuje brisanje vina Arbor Cuvée. Na slici se vidi da proizvoda više nema u tablici, a u gornjem desnom kutu vidi se poruka *Product Deleted* koja potvrđuje da je brisanje uspješno obavljeno.



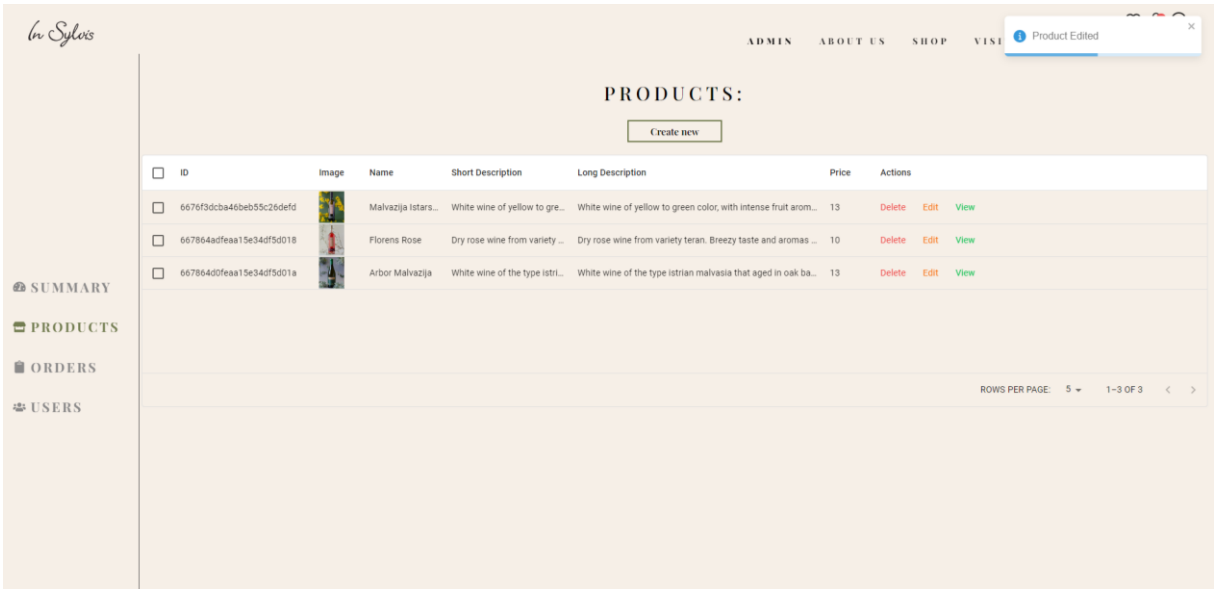
Slika 3.34. – Vino Arbor Cuvée uspješno je izbrisano

Klikom na gumb *Edit* otvara se prozor koji prikazuje trenutne podatke o proizvodu uz mogućnost njihovih uređivanja te trenutnu sliku proizvoda uz mogućnost njezine promjene. Nakon unosa promjena, administrator može kliknuti gumb *Cancel* kako bi odustao od promjena ili gumb *Submit* kako bi potvrdio promjene. Nakon toga, prikazuje se poruka *Product Edited* koja potvrđuje da je proizvod uspješno uređen. Slika 3.35. prikazuje trenutne podatke o vinu Malvazija Istarska te promijenjenu cijene iz 10 eura u 13.



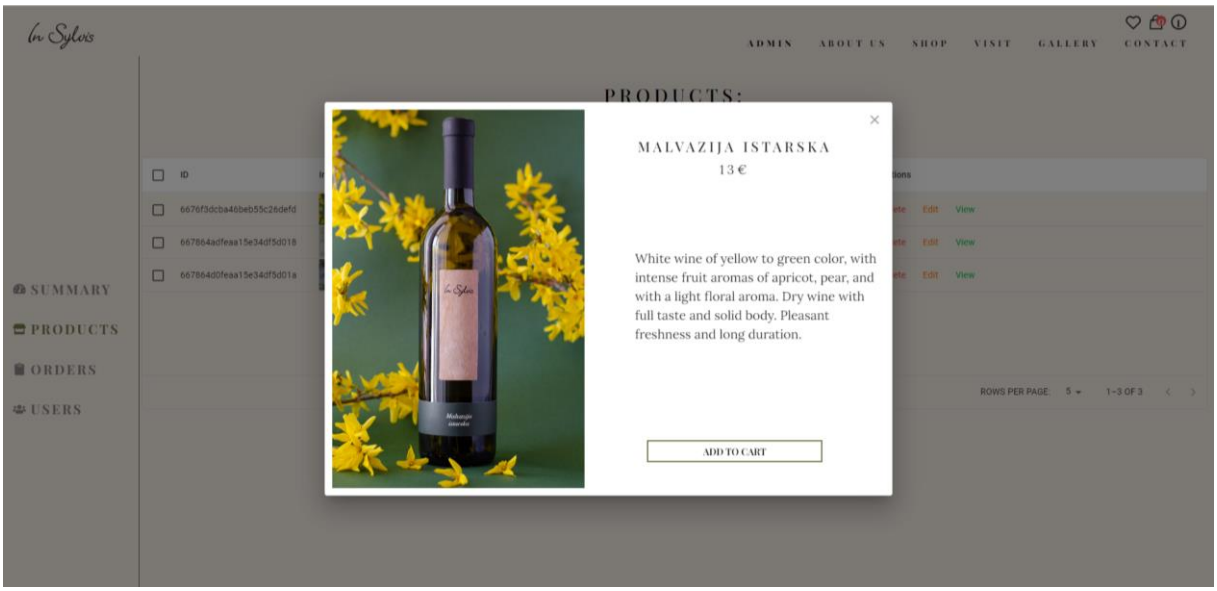
Slika 3.35. – Primjer uređenja podataka o vinu Malvazija Istarska

Na slici 3.36. vidljiva je promjena cijene i poruka potvrde o promjeni u gornjem desnom kutu.



Slika 3.36. – Vino Malvazija Istarska je uspješno uređeno

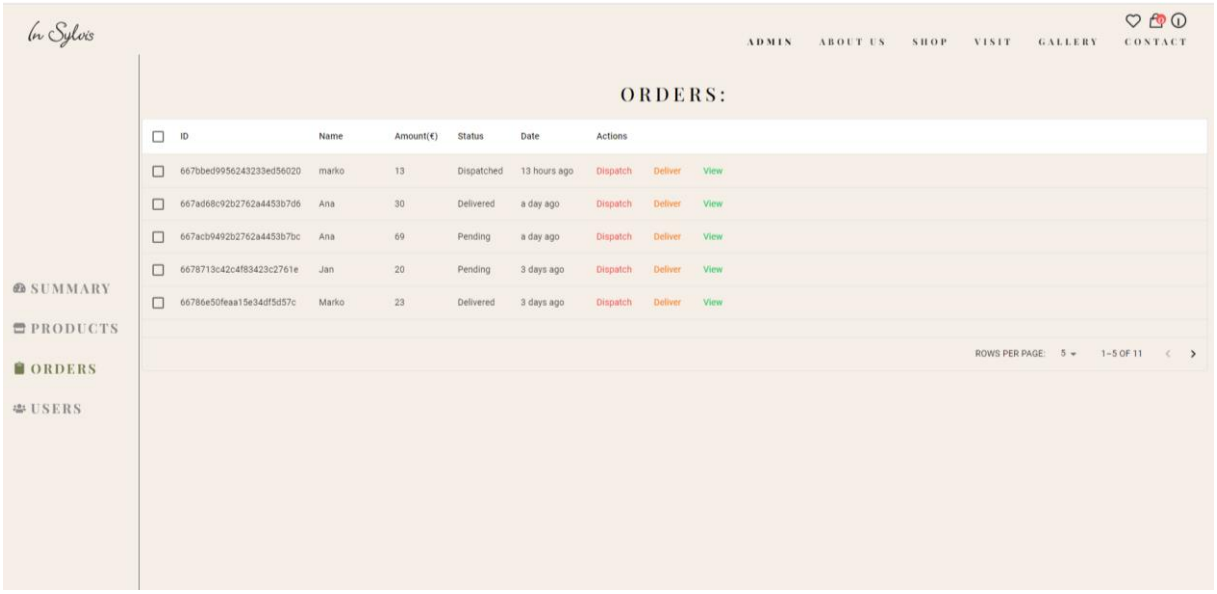
Klikom na gumb *View* otvara se prozor koji detaljnije prikazuje odabrani proizvod i sve informacije vezane uz njega. Slika 3.37. prikazuje detaljan prikaz vina Malvazija Istarska.



Slika 3.37. – Detaljan prikaz vina

3.7.3 Narudžbe

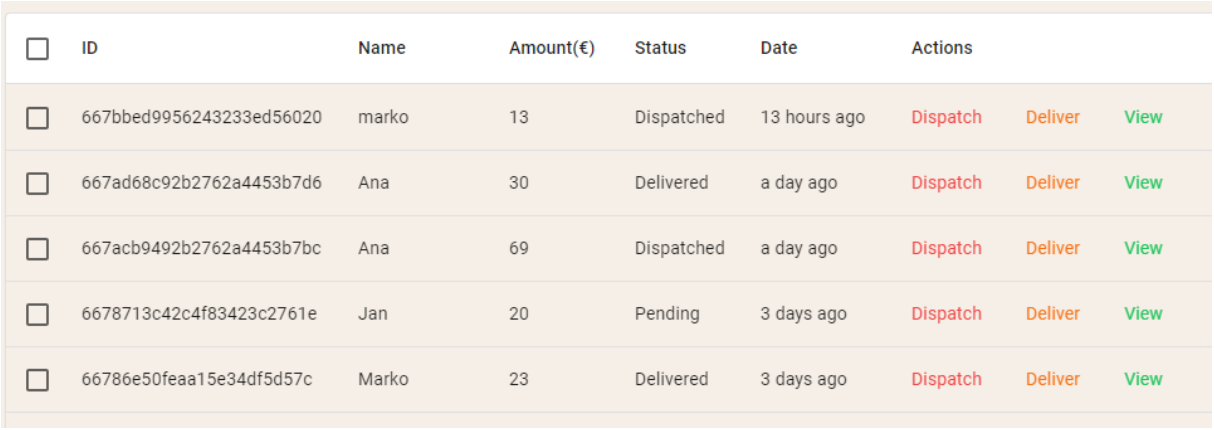
Slika 3.38. prikazuje stranicu koja se otvara pritiskom na poveznicu *ORDERS*. Stranica se nalazi na *URL* adresi /admin/orders i prikazuje tablicu s podacima o svim uspješnim narudžbama, uključujući ID narudžbe, ime naručitelja, iznos, status narudžbe te datum narudžbe. Također, uz svaku narudžbu nalaze se gumbi *Dispatch*, *Deliver* i *View*.



Slika 3.38. – Izgled stranice Narudžbe

Status svake narudžbe u početku je postavljen na *Pending*, što znači da je narudžba u tijeku. Gumb *Dispatch* mijenja status odabrane narudžbe u *Dispatched*, što znači da je

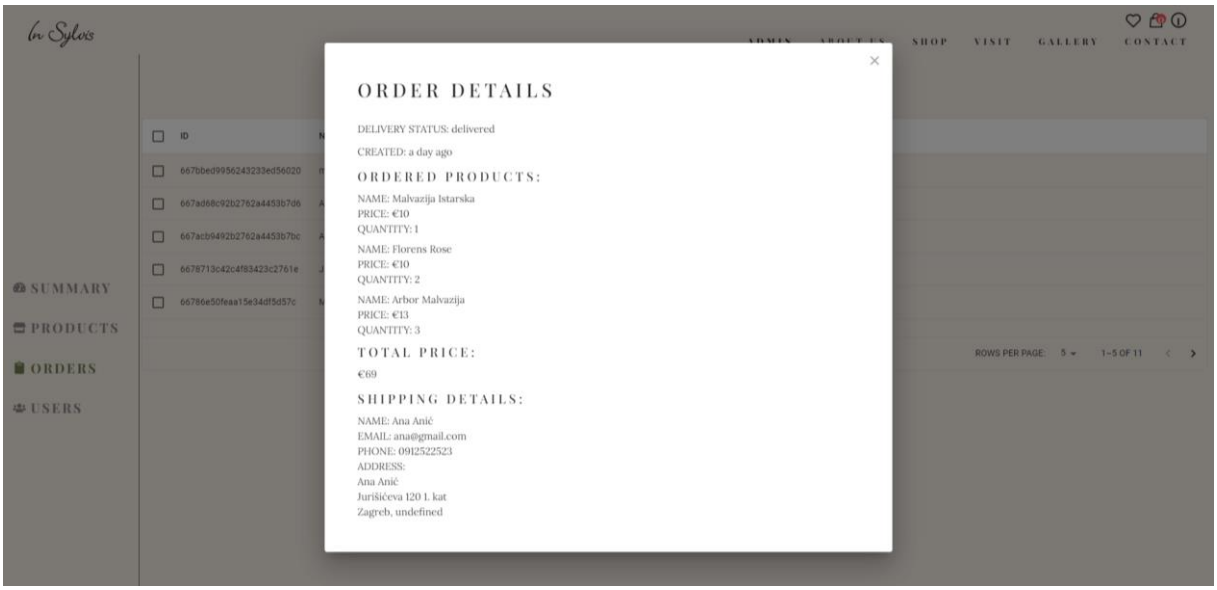
narudžba otpremljena, dok gumb *Deliver* mijenja status u *Delivered*, što znači da je narudžba dostavljena. Na taj način administratoru je olakšana briga i organizacija narudžbi. Na slici 3.39. prikazana je promjena statusa narudžbe naručitelja pod imenom Ana u *Dispatched*.



ID	Name	Amount(€)	Status	Date	Actions		
667bbed9956243233ed56020	marko	13	Dispatched	13 hours ago	Dispatch	Deliver	View
667ad68c92b2762a4453b7d6	Ana	30	Delivered	a day ago	Dispatch	Deliver	View
667acb9492b2762a4453b7bc	Ana	69	Dispatched	a day ago	Dispatch	Deliver	View
6678713c42c4f83423c2761e	Jan	20	Pending	3 days ago	Dispatch	Deliver	View
66786e50feaa15e34df5d57c	Marko	23	Delivered	3 days ago	Dispatch	Deliver	View

Slika 3.39. – Promjena statusa narudžbe naručitelja pod imenom Ana u Dispatched

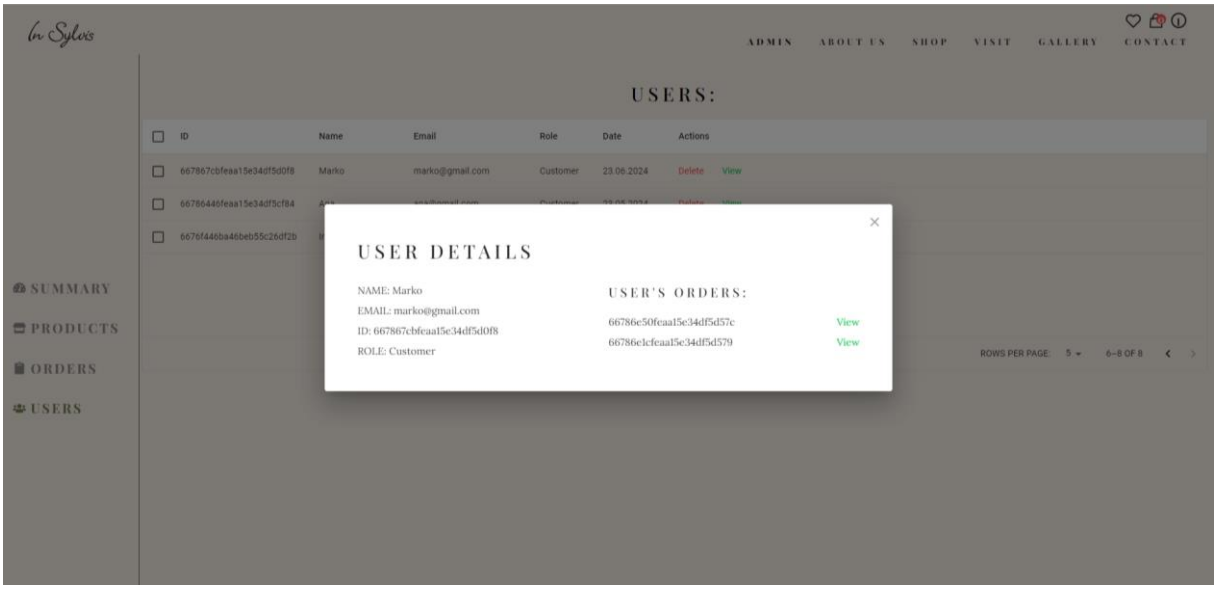
Klikom na gumb *View* otvara se prozor koji prikazuje sve detalje odabrane narudžbe, uključujući njezin status i vrijeme nastanka, popis naručenih proizvoda i podatke za slanje. Slika 3.40. prikazuje primjer detalja narudžbe naručitelja pod imenom Ana.



Slika 3.40. – Detaljan prikaz narudžbe

3.7.4 Korisnici

Klikom na link *USERS* administrator je usmjeren na *URL* adresu /admin/users, gdje je prikazana tablica registriranih korisnika. Slika 3.41. prikazuje izgled te tablice. U tablici se za svakog korisnika nalaze ID oznaka, ime, *e-mail* adresa, uloga (administrator ili običan korisnik) te datum registracije. Također, uz svakog korisnika dostupni su gumbovi *Delete* i *View*. Gumb *Delete* briše korisnika na isti način kao što istoimeni gumb briše proizvode u tablici proizvoda. Gumb *View* otvara prozor s detaljima o korisniku. Slika 3.41. prikazuje detaljan prikaz korisnika pod imenom Marko. Detaljan prikaz korisnika uključuje njegovo ime, *e-mail* adresu, ID, ulogu te broj svih narudžbi tog korisnika. Svaki broj narudžbi ima gumb *View* koji vodi na detalje te narudžbe.



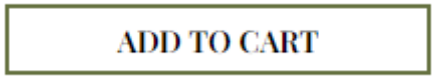
Slika 3.41. – Detaljan prikaz korisnika

4 OPIS FUNKCIONALNOSTI

U ovom poglavlju detaljno su opisane dvije ključne funkcionalnosti za svaku e-trgovinu. Prva funkcionalnost omogućava korisnicima jednostavno i intuitivno odabiranje proizvoda koje žele kupiti i dodavanje istih u virtualnu košaricu. Druga funkcionalnost obuhvaća dodavanje proizvoda u web shop putem administratorskog sučelja, što uključuje unos novih proizvoda, njihove karakteristike, cijene i slično. Kroz ovo poglavlje istražuje se kako se ove značajke implementiraju na razini koda, uključujući *backend* i *frontend* komponente, te kako se prezentiraju korisniku na ekranu.

4.1 Dodavanje proizvoda u košaricu

Na slici 4.1. prikazan je izgled gumba *ADD TO CART* koji se nalazi na početnoj stranici i u trgovini. Klikom na taj gumb proizvod se dodaje u košaricu, a na zaslonu se prikazuje poruka koja to i potvrđuje.



Slika 4.1. – Add to cart gumb

Na slici 4.1. prikazan je isječak programskog koda koji prikazuje kod *ADD TO CART* gumba. Klikom na gumb poziva se funkcija *handleAddToCart()* i šalje joj se odabrani proizvod.

```
<Button
  onClick={() => handleAddToCart(product)}
  style={{ width: "70%" }}
>
  ADD TO CART
</Button>
```

Slika 4.1. – Isječak koda Add to cart gumba

Na slici 4.2. prikazano je kako funkcija *handleAddToCart()* poziva dvije *dispatch* funkcije. Funkcija *dispatch()* je dio Redux biblioteke i koristi se za slanje akcija (*actions*) prema Redux *storeu* kako bi se ažuriralo stanje aplikacije. Funkcija *dispatch(addToCart(product))* šalje akciju *addToCart()* zajedno s informacijama o proizvodu prema Redux *storeu*. *Reducer* u Redux *storeu* obrađuje ovu akciju i ažurira

stanje košarice tako da uključi novi proizvod. Nakon toga *dispatch(getTotals())* šalje akciju *getTotals()* prema Redux *storeu* te *reducer* u redux *storeu* obrađuje ovu akciju i ažurira broj ukupnih proizvoda u košarici te ukupan novčani iznos proizvoda koji su u košarici.

```
const handleAddToCart = (product) => {
  dispatch(addToCart(product));
  dispatch(getTotals());
};
```

Slika 4.2. – Isječak koda koji prikazuje handleAddToCart funkciju

Na slici 4.3. prikazan je simbol košarice i crveni krug koji se nalaze u navigacijskoj traci. Unutar kruga nalazi se broj ukupnih proizvoda koji se nalaze u košarici.

Slika 4.3. – Simbol košarice i broj proizvoda unutar košarice

Na slici 4.4. je prikazan dio *cartSlicea*, Redux *slicea* kreiranog korištenjem funkcije *createSlice()* iz Redux Toolkita. Unutar objekta *reducers* nalazi se i *reducer addToCart()* koji služi za dodavanje proizvoda u košaricu. *Reduceri* su funkcije koje određuju kako će se stanje promijeniti kao odgovor na određene akcije. U ovom slučaju *reducer* addToCart mijenja stanje košarice. Funkcija *addToCart()* prima dva argumenta: argument *state* odnosno trenutačno stanje košarice te argument *action* odnosno akciju koja se prethodno otpremila.

```
const cartSlice = createSlice({
  name: "cart",
  initialState,
  reducers: {
    addToCart(state, action) {
      const itemIndex = state.cartItems.findIndex(
        (item) => item._id === action.payload._id
      );

      if (itemIndex >= 0) {
        state.cartItems[itemIndex].cartTotalQuantity += 1;

        toast.info(
          `Increased "${state.cartItems[itemIndex].name}" cart quantity`,
          {
            position: "bottom-left",
          }
        );
      } else {
        const tempProduct = { ...action.payload, cartTotalQuantity: 1 };
        state.cartItems.push(tempProduct);

        toast.success(
          `"${action.payload.name}" was succesfully added to your cart`,
          {
            position: "bottom-left",
          }
        );
      }

      localStorage.setItem("cartItems", JSON.stringify(state.cartItems));
    },
```

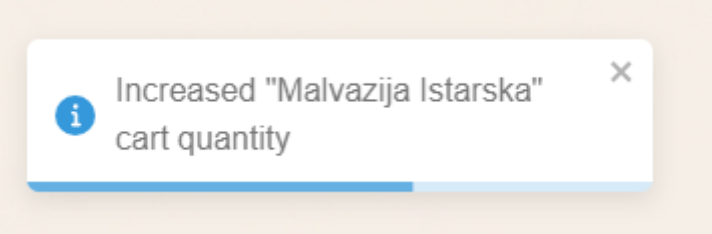
Slika 4.4. – Isječak koda koji prikazuje dio cartSlicea

Na slici 4.5. vidljiv je djelić koda zaslužan za pronalaženje proizvoda u košarici. Funkcija *findIndex()* pronalazi indeks proizvoda u nizu *cartItems* koji ima isti *_id* kao proizvod u *action.payload* odnosno proizvod koji želimo dodati u košaricu. U kodu se vidi uvjet koji provjerava je li indeks proizvoda (*itemIndex*) veći ili jednak nuli. Ako je uvjet ispunjen, to znači da je proizvod već prisutan u košarici i izvodi se kod unutar tog uvjeta. Količina proizvoda s indeksom *itemIndex* se povećava za 1. Nakon ažuriranja količine, prikazuje se informativna poruka korisniku koristeći biblioteku *toast* za poruke obavijesti.

```
if (itemIndex >= 0) {
  state.cartItems[itemIndex].cartTotalQuantity += 1;

  toast.info(
    `Increased "${state.cartItems[itemIndex].name}" cart quantity`,
    {
      position: "bottom-left",
    }
  );
```

Slika 4.5. – Isječak koda zaslužan za pronalaženje proizvoda u košarici

Slika 4.6. prikazuje kako korisnik vidi *toast* poruku.



Slika 4.6. – Toast poruka vidljiva na ekranu korisnika

Na slici 4.7. prikazan je *else* blok. Ovaj blok koda se izvršava ako proizvod nije pronađen u košarici (*itemIndex* je manji od 0). U ovom dijelu koda prvo se stvara novi objekt *tempProduct* koji sadrži sve podatke iz *action.payload*, zajedno s dodatnim svojstvom *cartTotalQuantity* koje se postavlja na 1. Korištenjem operatora razlaganja (eng. *spread operator*), svi podaci iz *action.payload* (kao što su *_id*, *name*, *price* i drugi) kopiraju se u novi objekt *tempProduct*. Dodavanjem svojstva *cartTotalQuantity* postavlja se početna količina proizvoda u košarici na 1. Nakon stvaranja novog objekta *tempProduct*, taj objekt se dodaje na kraj niza *cartItems* pomoću metode *push()*. Ovo osigurava da se novi proizvod pravilno dodaje u košaricu. Kao posljednji korak, koristi se *toast.success()* za prikaz obavijesti korisniku. Ova funkcija prikazuje poruku koja informira korisnika da je novi proizvod uspješno dodan u košaricu.

```
} else {
  const tempProduct = { ...action.payload, cartTotalQuantity: 1 };
  state.cartItems.push(tempProduct);

  toast.success(
    `"${action.payload.name}" was succesfully added to your cart`,
    {
      position: "bottom-left",
    }
  );
}
```

Slika 4.7. – Isječak koda koji prikazuje else blok zaslužan za dodavanje proizvoda u cartItems

Na slici 4.8. je prikazana linija koda koja sprema trenutno stanje košarice u *localStorage* preglednika. *LocalStorage* je ugrađeni objekt u *web* preglednicima koji omogućuje pohranjivanje podataka na strani klijenta. Podaci pohranjeni u *localStorage* ostaju sačuvani čak i nakon što korisnik zatvori preglednik.

```
localStorage.setItem("cartItems", JSON.stringify(state.cartItems));
```

Slika 4.8. – Linija koda koja sprema trenutno stanje košarice u localStorage

Slika 4.9. prikazuje *reducer getTotals()* koji prima *state* i *action* kao parametre. Svrha funkcije je izračunati ukupnu količinu proizvoda i ukupni iznos u košarici za kupovinu (*cartItems*), koji su pohranjeni u *state*. Funkcija koristi *reduce* metodu na *state.cartItems* kako bi iterirala kroz svaki element u listi *cartItems*, što su proizvodi u košarici. Objekt se inicijalizira s varijablama *total* (ukupni iznos) i *quantity* (ukupna količina proizvoda) postavljenim na početne vrijednosti 0. Za svaki *cartItem* u *state.cartItems*, funkcija izračunava *itemTotal* kao umnožak cijene proizvoda (*price*) i količine tog proizvoda (*cartTotalQuantity*). Tijekom iteracije, *reduce* funkcija ažurira *total* dodajući *itemTotal* na trenutni *total*, te ažurira *quantity* dodajući *cartTotalQuantity* na trenutni *quantity*. Nakon završetka iteracije, varijable *total* i *quantity* sadrže konačne vrijednosti ukupnog iznosa i ukupne količine svih proizvoda u košarici. Te konačne vrijednosti se zatim dodjeljuju objektu *state* pod ključevima *cartTotalAmount* i *cartTotalQuantity*, ažurirajući tako stanje aplikacije s novim vrijednostima. Ovaj

postupak osigurava da nakon izvršenja funkcije *getTotals()*, objekt *state* sadrži točne podatke o ukupnom iznosu i količini svih proizvoda koji su trenutno u košarici za kupovinu.

```
  getTotals(state, action) {
    let { total, quantity } = state.cartItems.reduce(
      (cartTotal, cartItem) => {
        const { price, cartTotalQuantity } = cartItem;
        const itemTotal = price * cartTotalQuantity;

        cartTotal.total += itemTotal;
        cartTotal.quantity += cartTotalQuantity;

        return cartTotal;
      },
      {
        total: 0,
        quantity: 0,
      }
    );

    state.cartTotalQuantity = quantity;
    state.cartTotalAmount = total;
  },
},
```

Slika 4.9. – Isječak koda koji prikazuje getTotals funkciju

Dio koda na slici 4.10. odnosi se na izvoz akcija i *reducer* funkcija koje su definirane unutar *cartSlicea*. Linija *„export const { addToCart, removeFromCart, decreaseCart, clearCart, getTotals } = cartSlice.actions;“* omogućuje izvoz svih definiranih akcija unutar *cartSlicea*, uključujući prethodno objašnjene akcije *addToCart()* i *getTotals()*. Također, linija *„export default cartSlice.reducer;“* izvozi glavnu *reducer* funkciju koja je također definirana unutar *cartSlicea*.

```
export const { addToCart, removeFromCart, decreaseCart, clearCart, getTotals } =
  cartSlice.actions;

export default cartSlice.reducer;
```

Slika 4.10. – Isječak koda koji prikazuje izvoz akcija i reducer funkcija

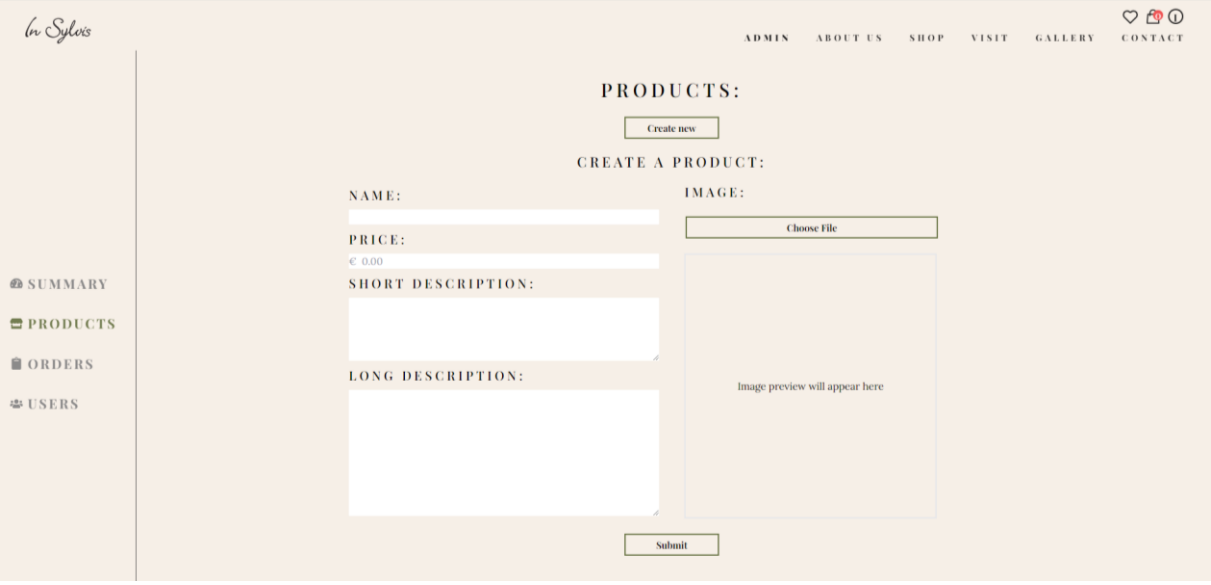
Na slici 4.11. nalazi se kod koji konfigurira centralno stanje aplikacije (*store*) korištenjem funkcije *configureStore()*. Definirani su različiti *reduceri* za upravljanje proizvodima, narudžbama, korisnicima, autentifikacijom i košaricom. Ovaj ulomak koda koristi *ReactDOM.createRoot()* za postavljanje korijenske komponente aplikacije na elementu s *ID-om root*. Nakon toga, koristi se *root.render()* za renderiranje aplikacije unutar tog korijenskog elementa. U okviru renderiranja, koristi se *<React.StrictMode>* kao omotač koji pomaže u otkrivanju potencijalnih problema u aplikaciji tijekom razvoja. Unutar *<React.StrictMode>*, koristi se *<Provider>* komponenta kako bi se aplikacija povezala s globalnim stanjem (*store*) putem *React Context API-ja*, omogućujući komponentama pristup stanju i akcijama koje modificiraju stanje.

```
const store = configureStore({
  reducer:{
    products: productsReducer,
    orders: ordersReducer,
    users: usersReducer,
    cart: cartReducer,
    auth: authReducer,
    [productsApi.reducerPath]: productsApi.reducer,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(productsApi.middleware),

});

store.dispatch(productsFetch());
store.dispatch(ordersFetch());
store.dispatch(usersFetch())
store.dispatch(getTotals());
store.dispatch(loadUser(null));

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store = {store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

Slika 4.11. – Centralno stanje aplikacije

4.2 Dodavanje proizvoda u trgovinu

Na administratorskoj strani aplikacije, točnije na *URL* adresi */admin/products/create-product*, admin ima mogućnost dodavanja proizvoda u *web shop*. Slika 4.12. prikazuje obrazac vidljiv administratoru prilikom dodavanja proizvoda.



Slika 4.12. – Izgled obrasca za dodavanje novog proizvoda

Slika 4.13. i slika 4.14 zajedno prikazuju isječak koda tog obrasca. Ovaj kod definira *HTML* formu koja se koristi za unos podataka o novom proizvodu. Forma je stilizirana pomoću Tailwind CSS-a i sastoji se od dva glavna dijela koji su raspoređeni u dva stupca.

```
<form
  className="flex flex-col w-full px-20 items-center gap-5 "
  style={{ height: "750x", width: "1200px" }}
  onSubmit={handleSubmit}
>
  <div className="flex w-full gap-10 ">
    <div className="  flex flex-col items-left text-left pl-2 justify-center gap-2 w-1/2">
      <p>name:</p>
      <input
        className="input"
        type="text"
        required
        placeholder=""
        onChange={(e) => setName(e.target.value)}
      />
      <p>price:</p>
      <input
        className="input"
        type="number"
        required
        placeholder="€  0.00"
        onChange={(e) => setPrice(e.target.value)}
      />
      <p>short description:</p>
      <textarea
        style={{ height: "100px", boxSizing: "border-box" }}
        className="input"
        type="text"
        placeholder=""
        onChange={(e) => setShortDesc(e.target.value)}
      />
      <p>long description:</p>
      <textarea
        style={{ height: "200px" }}
        className="input"
        type="text"
        placeholder=""
        onChange={(e) => setLongDesc(e.target.value)}
      />
    </div>
```

Slika 4.13. – Prvi dio isječka koda obrasca za dodavanje novog proizvoda

```
    <div className=" flex  w-1/2 h-full items-center">
      <div className="gap-5 flex flex-col">
        <p>image:</p>
        <label className="button" style={{ width: "400px" }}>
          <input
            type="file"
            accept="image/*"
            onChange={handleProductImageUpload}
            style={{ display: "none" }}
            required
          />
          Choose File
        </label>

        <div
          className=" flex align-center justify-center ■border-gray-200 border-2 "
          style={{ height: "420px", width: "400px" }}
        >
          {productImg ? (
            <div>
              <img
                src={productImg}
                alt="product"
                className="w-full h-full object-cover"
              />
            </div>
          ) : (
            <div className=" justify-center items-center flex flex-col">
              Image preview will appear here
            </div>
          )}
        </div>
      </div>
    </div>
  </div>

  <div className="items-center">
    <Button type="submit">Submit</Button>
  </div>
</form>
```

```
.object-cover {
    object-fit: cover;
}
```

Slika 4.14. – Drugi dio isječka koda obrasca za dodavanje novog proizvoda

Prvi stupac forme za unos proizvoda sastoji se od nekoliko polja koja omogućavaju administrativnim korisnicima unos osnovnih informacija o proizvodu. Svako od ovih polja je povezano s odgovarajućim stanjem u React komponenti, što omogućava dinamičko ažuriranje podataka dok korisnici unose informacije. Na primjer, stanje *name* se koristi za čuvanje naziva proizvoda, *price* za cijenu, *shortDesc* za kratki opis, i *longDesc* za dugi opis proizvoda. Na slici 4.15. vidljiva je inicijalizacija tih stanja pomoću *useState Hooks*, gdje se početne vrijednosti postavljaju na prazne *stringove* ("").

```
const [productImg, setProductImg] = useState("");
const [name, setName] = useState("");
const [price, setPrice] = useState("");
const [shortDesc, setShortDesc] = useState("");
const [longDesc, setLongDesc] = useState("");
```

Slika 4.15. – Inicijalizacija stanja pomoću useState Hooks

Funkcije poput *setName(), setPrice(), setShortDesc(), setLongDesc()* se koriste za ažuriranje tih stanja u odgovarajućim *onChange* događajima unutar *HTML input* i *textarea* elementima. Na taj način, komponenta može pratiti i reagirati na promjene koje korisnici unose u formu.

Drugi stupac uključuje dio za unos slike proizvoda. Koristi se *label* element stiliziran kao gumb (*button*), unutar kojeg je skriveni *input* element tipa *file* koji omogućuje odabir slike proizvoda. Kada se odabere slika, funkcija *handleProductImageUpload()* se poziva kako bi se obrađena slika prikazala u *preview* dijelu ispod. *Preview* dijelom upravlja se prikaz slike proizvoda ako je odabrana (*productImg*), inače se prikazuje poruka *Image preview will appear here*. Slika 4.16. prikazuje dvije funkcije koje se koriste za upravljanje učitavanjem i transformacijom slike proizvoda. Prva funkcija, *handleProductImageUpload()*, poziva se kada korisnik odabere sliku proizvoda putem *input* elementa tipa *file*. Ona prima *event e*, iz kojeg se dobiva pristup odabranim datotekama preko *e.target.files[0]*. Zatim se poziva funkcija *TransformFile()* s odabranom datotekom radi daljnje obrade. Druga funkcija, *TransformFile()*, je funkcija koja prima datoteku (*file*) kao argument. Koristi se *FileReader* objekt za čitanje sadržaja odabrane datoteke. Ako je datoteka definirana (nije *null* ili *undefined*), *reader.readAsDataURL(file)* pokreće čitanje datoteke kao *Data URL*. Kada se čitanje završi (*reader.onloadend*), postavlja se stanje *productImg* na rezultat čitanja (*reader.result*), što je *Data URL* slike. Ako datoteka nije definirana, stanje *productImg* se postavlja na prazan *string* (""), što rezultira brisanjem trenutne slike iz prikaza.

Ove funkcije omogućavaju dinamičko učitavanje slika proizvoda i prikazivanje njihovih pregleda unutar web sučelja za administraciju proizvoda. Na taj način, korisnici mogu vizualno pregledati i odabrati slike proizvoda prije nego što ih konačno potvrde za unos u *web shop*.

```
const handleProductImageUpload = (e) => {
  const file = e.target.files[0];

  console.log(file);

  TransformFile(file);
};

const TransformFile = (file) => {

  const reader = new FileReader();

  if (file) {
    reader.readAsDataURL(file);

    reader.onloadend = () => {
      setProductImg(reader.result);
    };
  } else {
    setProductImg("");
  }
};
```

Slika 4.16. – Isječak koda koji prikazuje funkcije handleProductImageUpload i TransformFile

Forma se zatvara gumbom za podnošenje (*Button* komponenta s atributom *type="submit"*), koji aktivira funkciju *handleSubmit()* kada je forma spremna za slanje. Slika 4.17. prikazuje funkciju *handleSubmit()*, koja se koristi kao rukovatelj događaja za događaj "*submit*" forme. Kada se forma pošalje, ova funkcija se automatski poziva i prima objekat događaja *e* kao argument. Prvo, unutar funkcije *handleSubmit()*, poziva se metoda *e.preventDefault()*, koja sprječava zadanu radnju slanja forme. Inače, kada se forma pošalje, preglednik automatski šalje podatke forme na server i osvježava stranicu. Korištenjem *e.preventDefault()*, sprječava se ovo ponašanje, što omogućava izvršenje prilagođenih radnji prije slanja podataka. Nakon što je zadana radnja spriječena, funkcija *handleSubmit()* koristi *dispatch()* metodu za slanje akcije *productsCreate()*.

```
const handleSubmit = (e) => {
  e.preventDefault();
  dispatch(
    productsCreate({
      name,
      price,
      shortDesc,
      longDesc,
      image: productImg,
    })
  );
};
```

Slika 4.17. – Isječak koda koji prikazuje funkciju handleSubmit

Na slici 4.18. prikazana je akcija *productsCreate()*. Ova akcija, definirana kao *createAsyncThunk* funkcija, prima objekat s informacijama o proizvodu kao što su *name*, *price*, *shortDesc*, *longDesc* i *image* (koja je povezana s varijablom *productImg*). Funkcija *productsCreate()* šalje ove podatke putem *HTTP POST* zahtjeva na server koristeći axios biblioteku, a *URL* i zaglavlja su definirani pomoću *setHeaders()* funkcije. Unutar *productsCreate()*, *axios.post()* šalje zahtjev na *${url}/products* s proslijeđenim values. Ako je zahtjev uspješan, vraća se *response.data*, koji sadrži podatke odgovora servera. Ako se dogodi greška, ona se bilježi u konzolu i prikazuje se greška korisniku putem *toast* obavijesti.

```
export const productsCreate = createAsyncThunk(
  "products/productsCreate",

  async (values) => {
    try {
      const response = await axios.post(
        `${url}/products`,
        values,
        setHeaders()
      );
      return response.data;
    } catch (error) {
      console.log(error);
      toast.error("an error occured");
    }
  }
);
```

Slika 4.18. – Isječak koda koji prikazuje akciju productsCreate

Na *backend* strani, Express.js *router* je konfiguriran za rukovanje *POST* zahtjevom za stvaranje novog proizvoda. Na slici 4.19. prikazan je isječak koda u kojem je napisan taj *router*. Rukovatelj ruta prima podatke o proizvodu iz *req.body*, uključujući *name*, *shortDesc*, *longDesc*, *price* i *image*. Ako je slika prisutna, koristi se Cloudinary za prijenos slike, koja se zatim pohranjuje s *URL*-om i javnim ID-om. Ako je prijenos slike uspješan, kreira se novi *Product* objekt s proslijeđenim podacima i prenesenom slikom, te se taj objekt pohranjuje u bazu podataka pomoću *Mongoose ORM-a*. Ako je pohrana uspješna, vraća se spremljeni proizvod kao odgovor s *HTTP* statusom 200. U slučaju greške, greška se bilježi u konzolu i vraća se kao odgovor s *HTTP* statusom 500.

```
const router = express.Router();

//CREATE
router.post("/", isAdmin, async (req, res) => {
  const { name, shortDesc, longDesc, price, image } = req.body;

  try {
    if (image) {
      const uploadRes = await cloudinary.uploader.upload(image, {
        upload_preset: "webShop", //images are saved in folder onlineShop
      });

      if (uploadRes) {
        const product = new Product({
          name,
          shortDesc,
          longDesc,
          price,
          image: {
            url: uploadRes.secure_url,
            public_id: uploadRes.public_id,
          },
        });

        const savedProduct = await product.save();

        res.status(200).send(savedProduct);
      }
    }
  } catch (error) {
    console.log(error);
    res.status(500).send(error);
  }
});
```

Slika 4.19. – Isječak koda koji prikazuje router konfiguriran za rukovanje POST zahtjevom za stvaranje novog proizvoda

Na slici 4.20. prikazan je model za proizvode (*productSchema*) definiran pomoću *Mongoosea* koji se također nalazi na *backend* strani. Ovaj model uključuje podatke o proizvodu koji omogućuju pohranu i upravljanje proizvodima u bazi podataka.

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

const productSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    shortDesc: { type: String },
    longDesc: { type: String },
    price: { type: Number, required: true },
    image: { type: Object, required: true },
  },
  { timestamps: true }
);

const Product = mongoose.model("Product", productSchema);

exports.Product = Product;
```

Slika 4.19. – Isječak koda koji prikazuje model za proizvode

5 ZAKLJUČAK

Ovaj završni rad opisuje razvoj i implementaciju *web shop* aplikacije za vinariju In Sylvis, pružajući detaljan uvid u tehničke aspekte i funkcionalnosti aplikacije. Korištenjem MERN Stack tehnologije (MongoDB, Express.js, React.js i Node.js) postignuta je robusna, skalabilna i efikasna aplikacija. MERN Stack omogućuje brzi razvoj i jednostavno održavanje, čineći ga idealnim izborom za *full stack* aplikacije poput ove.

Aplikacija omogućava korisnicima pregled proizvoda, točnije vina, te njihovo dodavanje i uklanjanje iz košarice te kupnju nakon registracije ili prijave. Trenutno ne podržava elektroničko plaćanje, ali je planirano njegovo uvođenje u budućnosti, zajedno sa slanjem potvrda narudžbi putem elektroničke pošte. Administratorima je omogućeno upravljanje proizvodima, korisnicima i narudžbama kroz *admin* stranicu, što osigurava efikasno vođenje poslovanja.

Ovaj rad detaljno prikazuje sve aspekte razvoja aplikacije, uključujući korištene tehnologije i funkcionalnosti. U budućnosti se planiraju dodatne funkcionalnosti kako bi se unaprijedilo korisničko iskustvo i poslovni procesi vinarije In Sylvis.

LITERATURA

[1] „Mongo DB“, s interneta, https://www.mongodb.com, 30. lipnja 2024.

[2] „Express“, s interneta, https://expressjs.com, 30. lipnja 2024.

[3] „React“, s interneta, https://react.dev, 30. lipnja 2024.

[4] „Node“, s interneta, https://nodejs.org/en, 30. lipnja 2024.

[5] „bcryptjs“, s interneta, https://www.npmjs.com/package/bcryptjs, 30. lipnja 2024.

[6] „joi“, s interneta, https://joi.dev, 30. lipnja 2024.

[7] „jsonwebtoken“, s interneta, https://jwt.io, 30. lipnja 2024.

[8] „cloudinary“, s interneta, https://cloudinary.com, 30. lipnja 2024.

[9] „dotenv“, s interneta, https://www.npmjs.com/package/dotenv, 30. lipnja 2024.

[10] „axios“, s interneta, https://www.axios.com, 30. lipnja 2024.

[11] „jwt-decode“, s interneta, https://www.npmjs.com/package/jwt-decode, 30. lipnja 2024.

[12] „Moment.js“, s interneta, https://momentjs.com, 30. lipnja 2024.

[13] „tailwindcss“, s interneta, https://tailwindcss.com, 30. lipnja 2024.

[14] „Redux Toolkit“, s interneta, https://redux-toolkit.js.org, 30. lipnja 2024.

[15] „MERN Stack Explained“, s interneta, https://www.mongodb.com/resources/languages/mern-stack, 30. lipnja 2024.

[16] „What is React and Why should you learn it in 2022-23?“, s interneta, https://aboutreact.com/reactjs, 30. lipnja 2024.

[17] „MUI“, s interneta, https://mui.com, 30. lipnja 2024.

[18] „react-dom“, s interneta, https://www.npmjs.com/package/react-dom, 30. lipnja 2024.

[19] „React Icons“, s interneta, https://www.npmjs.com/package/react-icons, 30. lipnja 2024.

[20] „React Redux“, s interneta, https://react-redux.js.org, 30. lipnja 2024.

[21] „React Router“, s interneta, https://reactrouter.com/en/main, 30. lipnja 2024.

[22] „React-Toastify“, s interneta, https://www.npmjs.com/package/react-toastify, 30. lipnja 2024.

[23] „Recharts“, s interneta, https://recharts.org/en-US, 30. lipnja 2024.

[24] „Semantic UI“, s interneta, https://semantic-ui.com, 30. lipnja 2024.

[25] „npm“, s interneta, https://www.npmjs.com, 30. lipnja 2024.

[26] „Frequently Asked Questions“, s interneta, https://cloudinary.com/faq, 30. lipnja 2024.

SAŽETAK

U ovom radu izrađena je *web* aplikacija za *online* prodaju koja omogućuje jednostavno i efikasno predstavljanje proizvoda, olakšava proces naručivanja i unapređuje komunikaciju s kupcima. Aplikacija korisnicima omogućuje pregled vina, dodavanje i uklanjanje proizvoda iz košarice, te kupnju nakon registracije ili prijave uz generiranje potvrde narudžbe. Administrator aplikacije može vidjeti sažetak poslovanja te upravljati proizvodima, korisnicima i narudžbama. Ova aplikacija izrađena je korištenjem MERN tehnološkog skupa (MongoDB, Express.js, React, Node.js) i drugih tehnologija kao što su Cloudinary i Tailwind CSS. MERN tehnološki skup je odabran jer omogućava brzi razvoj, lako održavanje i skalabilnost aplikacija, čineći ga idealnim za moderne web projekte.

Ključne riječi: web aplikacija, online prodaja, proces naručivanja, administrator, MERN tehnološki skup, Tailwind CSS, Cloudinary

ABSTRACT

In this thesis, a web application for online sales has been developed, which enables simple and efficient product presentation, facilitates the ordering process, and improves communication with customers. The application allows users to browse wines, add and remove products from the cart, and make purchases after registering or logging in, with order confirmation generation. The application administrator can view business summaries and manage products, users, and orders. This application was created using the MERN technology stack (MongoDB, Express.js, React, Node.js) and other technologies such as Cloudinary and Tailwind CSS. The MERN technology stack was chosen because it enables rapid development, easy maintenance, and scalability of applications, making it ideal for modern web projects.

Keywords: web application, online sales, ordering process, administrator, MERN technology stack, Tailwind CSS, Cloudinary