Implementacija SPA arhitekture primjenom Angular razvojnog okriva

Magdić, Iva

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Diplomski studij informatike

Poslovna informatika

Iva Magdić

Implementacija SPA arhitekture primjenom Angular razvojnog okvira

Diplomski rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, rujan 2023.

Sveučilište u Rijeci
FIDIT Fakultet informatike i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 25.4.2023.

Zadatak za diplomski rad

Pristupnik: Iva Magdić

Naziv diplomskog rada: Implementacija SPA arhitekture primjenom Angular razvojnog okriva

Naziv diplomskog rada na eng. jeziku: Implementing SPA architecture using the Angular development framework

Sadržaj zadatka: Zadatak završnog rada je izraditi web aplikaciju SPA arhitekture pomoću Angular razvojnog okvira. Prikazat će se sve faze razvoja web aplikacije koje uključuju razvoj početne ideje, dizajn korisničkih sučelja, implementaciju frontend i backend rješenja te završnu fazu postavljanja aplikacije na web. U radu će se također opisati sve tehnologije koje će se koristiti prilikom izrade web aplikacije te opisati funkcionalnosti izrađene aplikacije.

Mentor:

Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditeljica za diplomske radove:

Prof. dr. sc. Ana Meštrović

Zadatak preuzet: 26.4.2023.

(potpis pristupnika)

Iva Magdić

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: +385(0)51 584 700
E-mail: ured@inf.uniri.hr
OIB: 64218323816
IBAN: HR1524020061400006966

uniri

Sažetak

U suvremenom svijetu, web aplikacije postaju nezaobilazni alati za pružanje informacija i usluga globalnoj publici. Ovaj rad bavi se detaljnim prikazom procesa izrade jednostranične web aplikacije na praktičnom primjeru. Kroz analizu, istražuje se kako tehnološki napredak i softver otvorenog koda olakšavaju i ubrzavaju razvoj web aplikacija. Prikazane su sve faze razvoja koje uključuju razvoj početne ideje, dizajn korisničkih sučelja, implementaciju frontend i backend rješenja te završnu fazu postavljanja aplikacije na web. Na ovom praktičnom primjeru koristi se Angular razvojni okvir koji služi za razvoj frontend rješenja, te Firebase usluge kao backend rješenje. Rad stavlja poseban naglasak na važnost korisničkog iskustva, estetike i interaktivnosti u kreiranju aplikacija koje zadovoljavaju potrebe današnjih korisnika. Kroz praktični primjer, demonstrirane su tehnike i alati koji omogućuju izradu funkcionalnih, responzivnih i korisnički orijentiranih web aplikacija. Zaključno, rad pruža uvid u buduće mogućnosti i potencijalne nadogradnje u svijetu web razvoja.

Ključne riječi: Web aplikacija, Jednostranična Web aplikacija, Angular, TypeScript, Git, Github, Firebase, HTML, SCSS, NPM, Node.js, Frontend development, Backend development, SPA, UI/UX dizajn, web dizajn

Sadržaj

1. Uvod

U suvremenom svijetu, tehnološki napredak donio je revoluciju u načinu na koji pristupamo informacijama i uslugama. Brzina i jednostavnost s kojom se danas obavljaju različite aktivnosti rezultat su razvoja weba i web aplikacija koje su postale neizostavan dio svakodnevnog života. Sada se zadaci koji su nekada trajali satima mogu obaviti jednim klikom, bila to kupovina robe, rezerviranje autobusne karte, plaćanje računa i slično. Za ovakav napredak zaslužni su mnogi faktori, a neki od njih su široka dostupnosti brzog interneta i ljudska inovativnost.

Veliku ulogu u načinu kupovine imaju informacije dostupne na webu. Kupci danas koriste internet kao primarni alat za istraživanje proizvoda ili usluga prije donošenja odluke o kupnji. Detaljne specifikacije proizvoda, recenzije korisnika, usporedbe cijena i video demonstracije postali su standardni sadržaji koje potrošači očekuju pronaći online. Ova promjena u ponašanju potrošača potaknula je trgovce da prilagode svoje online strategije. Web stranice su postale interaktivnije, pružajući korisnicima mogućnost postavljanja pitanja, čitanja iskustava drugih korisnika ili čak virtualnog isprobavanja proizvoda. Međutim, s ovom digitalnom evolucijom dolazi i izazov transparentnosti. Dok trgovci na svojim web stranicama imaju mogućnost upravljanja recenzijama koje im korisnici ostavljaju, postoji potencijalna opasnost od selektivnog prikazivanja samo pozitivnih povratnih recenzija, dok se negativne mogu izostaviti ili ukloniti.

Ideja ovog rada je pružiti platformu gdje korisnici mogu pronaći i ostavljati recenzije proizvoda koje će biti vjerodostojne. Dodatno, platforma će se bazirati isključivo na proizvode hrvatskih kozmetičkih brendova, čime se želi promovirati domaća kozmetička industrija i dati prilika manje poznatim brendovima da se istaknu na tržištu. Osim toga, aplikacija će korisnicima omogućiti da istraže različite vrste proizvoda, saznaju više o njihovim sastojcima, upotrebi i korisničkim iskustvima. Svoje omiljene proizvode koje su pronašli moći će sačuvati u vlastitoj listi želja.

Cilj rada je prikazati proces izrade jednostranične aplikacije koristeći Angular kao razvojni okvir i Firebase kao backend rješenje. Osim tehničke implementacije, rad će se osvrnuti na važnost korisničkog iskustva na web aplikacijama.

Prvi dio ovog rada opisuje što su to web aplikacije te daje kratki pregled njihove povijesti. Osvrće se i na dva glavna modela po kojima se web aplikacije mogu razlikovati. Poseban naglasak stavljen je na opis jednostraničnih aplikacija, poznatijih kao SPA, i različitih okvira dostupnih za njihovu izradu.

Završni segment rada detaljno razrađuje sve faze razvoja web aplikacije pod nazivom „Croatian Cosmetics“. Nakon uvodnog pregleda korištenih tehnologija, rad se detaljno bavi implementacijom aplikacije, od faze dizajna, preko razrade frontenda, sve do backenda, a završava postupkom postavljanja aplikacije na web.

2. Web aplikacije

2.1. Kratka povijest

World Wide Web (WWW) je mrežni navigacijski servis koji je inicijalno nastao kao alat za pronalaženje resursa na Internetu. Nastao je iz ideje kreiranja univerzalne informacijske baze podataka na mreži koja je globalno dostupna te u kojoj se informacije lako pretražuju. WWW, koji je zaživio 1989. godine zahvaljujući britanskom znanstveniku Tom Berners-Leeju iz CERN-a (Europska organizacija za nuklearna istraživanja), zamišljen je kao alat za jednostavniji prijenos informacija unutar te organizacije. Ovaj sustav za razmjenu informacija temelji se na, tada novom, protokolu za razmjenu hiperteksta (eng. *HyperText Transfer protocol*, *HTTP*) [1]. Ključna organizacija koja definira web standarde, uključujući HTML, CSS i JavaScript, je The World Wide Web Consortium (W3C). Njihovi standardi, poznati kao W3C preporuke, oblikuju način kreiranja i interpretiranja web sadržaja [2].

Web stranica i web aplikacija dva su ključna koncepta unutar World Wide Weba, no imaju distinktivne karakteristike. Web stranica je prvenstveno digitalni dokument dostupan na internetu, često sastavljen od statičkih elemenata kao što su tekst i slike. Glavna svrha web stranice je informiranje, gdje korisnici pasivno konzumiraju sadržaj. Interakcija je minimalna, te se najčešće svodi samo na pregledavanje sadržaja.

S druge strane, web aplikacija je interaktivno softversko rješenje koje korisnicima omogućava obavljanje određenih zadataka ili pristup specifičnim funkcionalnostima putem internetskog preglednika. Dok web stranica služi kao izlog informacija, web aplikacija nudi dinamično korisničko iskustvo, gdje korisnici mogu unositi, mijenjati ili upravljati informacijama. Web aplikacije često zahtijevaju autentifikaciju korisnika, dok je pristup većini web stranica otvoren svima [3]. Razlika između web aplikacija i tipične web stranice ponekad nije očita na prvi pogled. Web aplikacija upravlja korisničkim zahtjevima pomoću poslužitelja i prikazuje informacije pomoću kombinacije skriptiranja na strani klijenta i poslužitelja. S druge strane, web stranica je grupa povezanih web stranica. Ona ne mora biti unaprijed kompilirana prije postavljanja na poslužitelj, dok web aplikacija mora. Web stranicu može svatko vidjeti bez provjere autentičnosti, dok web aplikacija često to zahtjeva.

Web aplikacije su postale ključna komponenta suvremenog digitalnog svijeta. U velikoj mjeri zamjenjuju stare desktop aplikacije jer su praktičnije za upotrebu, lako se ažuriraju i nisu vezane uz jedan uređaj. Omogućuju korisnicima pristupanje različitim uslugama, funkcijama i sadržajima putem internetskog preglednika bez potrebe za prethodnim instaliranjem softvera na svoj uređaj. Iako se korisnici postupno prebacuju s web aplikacija na mobilne aplikacije, potražnja za složenim i sofisticiranim aplikacijama već je ogromna te i dalje raste. Njihova pristupačnost, fleksibilnost i široka primjena u različitim industrijama čine ih neizostavnim alatom za olakšavanje komunikacije, trgovine, edukacije i poslovanja u globalnom okruženju [3].

2.2. Što je SPA (Single Page Application)?

Postoje dva glavna modela web aplikacija: višestranične web aplikacije (eng. *Multi page applications, MPA*) i jednostranične web aplikacije (eng. *Single page applications, SPA*). Oba modela imaju svoje prednosti i nedostatke.

2.2.1. Višestranična aplikacija

Višestranična aplikacija je tradicionalni model aplikacija koji ima više od jedne stranice sa statičkim podacima (tekst, slike itd.), a svaka stranica odgovara različitom URL-u. Svaka promjena zahtijeva

ponovno renderiranje nove stranice s poslužitelja u pregledniku. Drugim riječima, svaki prelazak s jedne stranice na drugu unutar web aplikacije zahtjeva učitavanje sadržaja stranice i preuzimanja resursa, čak i ako se komponente ponavljaju na svim stranicama (poput zaglavlja i podnožja) [4]. Ovakva arhitektura aplikacije, obzirom da zahtijeva stalnu interakciju s poslužiteljem, rezultira sporijim vremenom učitavanja, posebice ako se korisnik nalazi na sporijoj mrežnoj vezi.
Višestranične aplikacije obično imaju veliku količinu podataka i kompleksnu arhitekturu. Zbog obilja sadržaja, ove aplikacije imaju mnoge razine korisničkog sučelja [4].

Neke od prednosti višestranične web aplikacije su [4]:

- Visoka skalabilnost: ne postoji ograničenje broja stranica koje se mogu dodati postojećoj aplikaciji
- Optimiziran SEO[1] - višestranične web aplikacije najbolje odgovaraju uslugama ili proizvodima s ciljem boljeg rangiranja za različite ključne riječi, budući da svaka stranica aplikacije može biti indeksirana i optimizirana za pretraživanje
- Analitika korisnika - programi poput Google Analytics-a mogu pružiti različite izvještaje o performansama različitih stranica u višestraničnoj aplikaciji. Ovi izvještaji omogućuju dublji uvid u to koje stranice privlače najviše posjetitelja, kako korisnici komuniciraju s pojedinim stranicama te koliko vremena provode na njima. Takve informacije mogu biti korisne u svrhe poboljšanja pojedinih stranica kako bi se njihova vidljivost povećala

Neki od nedostataka višestraničnih aplikacija su [4]:

- Sporija izvedba - višestranična aplikacija se ponovno učitava svaki put kada korisnik klikne novu stranicu. Resursi poput HTML-a, CSS-a i JavaScripta obnavljaju se svaki put kada se izvrši neka akcija što utječe na brzinu i izvedbu web aplikacije
- Vrijeme razvoja - u usporedbi s jednostraničnim aplikacijama, višestranične aplikacije imaju veći broj značajki čime njihov razvoj zahtijeva više truda i resursa
- Teže održavanje - razvojni tim mora održavati svaku stranicu višestranične aplikacije zasebno i redovito. Potrebno je i pojedinačno osigurati svaku stranicu, što povećava ukupni teret razvoja ovakvih aplikacija

2.2.2. Jednostranična aplikacija

Jednostranična aplikacija, SPA, je implementacija aplikacije koja učitava samo jedan web dokument te dohvaća i ažurira specifične dijelove podataka kada je potrebno, bez ponovnog učitavanja cijele stranice [4]. Ova postupna izgradnja na strani klijenta značajno ubrzava vrijeme učitavanja za korisnike i istovremeno smanjuje opterećenje poslužitelja, čineći proces ekonomičnijim. Korištenje web aplikacije ili web stranice koja komunicira s korisnicima putem dinamičkog ažuriranja trenutačne stranice, umjesto potpuno novog učitavanja cijelih stranica s poslužitelja, znatno poboljšava korisničko iskustvo [4].

S tehničke strane, jedna od tehnologija na koju se SPA oslanja za dohvat podataka je AJAX[2] (eng. *Asynchronus JavaScript and XML*), što omogućuje ažuriranje dijelova stranice bez potrebe za osvježavanjem cijele stranice. To znači da kada korisnik obavlja neku akciju, poput klika na gumb ili

[1] SEO (Search Engine Optimization) je proces optimizacije web stranica s ciljem poboljšanja njihove vidljivosti na tražilicama za određene ključne riječi.
[2] Ajax je skup tehnika razvoja weba koji koristi različite web tehnologije na klijentskoj strani kako bi stvorio asinkrone web aplikacije [5].

pretraživanje, JavaScript koji se izvodi na strani klijenta komunicira s poslužiteljem asinkrono, dohvaća potrebne podatke i ažurira određeni dio stranice bez potrebe za osvježavanjem cijelog sučelja [5].
Za razliku od tradicionalnih web aplikacijama koje zahtijevaju od korisnika čekanje na odgovor s poslužitelja prilikom prijelaza na novu stranicu, AJAX omogućava korisnicima da komuniciraju sa stranicom tijekom tog procesa. Ključni elementi SPA tehnologija temelje se na komponentama poput HTML-a, CSS-a, DOM-a, XMLHttpRequest objekta i JavaScripta. Svaka od ovih komponenti ima svoju ulogu: HTML i CSS koriste se za standardno prikazivanje sadržaja na webu, DOM (eng. *Document Object Model*) omogućava dinamičko prikazivanje i interakciju, XMLHttpRequest se koristi za razmjenu podataka, dok JavaScript djeluje kao vezivno tkivo i pruža logiku aplikaciji. Sve ove komponente su široko podržane u svim glavnim web preglednicima [6].

Neke od prednosti jednostraničnih aplikacija su [4]:

- Brža izvedba - svi resursi se učitavaju tijekom jedne sesije, a zatim se prilikom interakcije mijenjaju samo potrebni podaci
- Pohrana podataka - nakon prvog zahtjeva prema poslužitelju, svi potrebni podaci pohranjuju se na uređaju korisnika što omogućuje rad bez internetske veze. Primjer ovakvog slučaja je Google Docs u offline načinu rada
- Ubrzani proces ispravljanja grešaka - Većina jednostraničnih aplikacija razvijena je uz korištenje tehnologija kao što su React, Vue.js i Angular, temeljenih na Google Chromeu. Ovo omogućuje praćenje mrežnih operacija, analizu elemenata na stranici i povezanih podataka, značajno pojednostavljujući proces ispravljanja eventualnih grešaka

Neki od glavnih nedostataka jednostraničnih aplikacija (SPA) su [4]:

- SEO optimizacija – SEO kod jednostraničnih aplikacija je izazov koji leži u načinu na koji one učitavaju svoj sadržaj, bez potrebe za ponovnim učitavanjem cijele stranice, i to samo kad korisnik to zahtijeva. Kao rezultat toga, ne postoje posebni URL-ovi optimizirani za pretraživače. Ipak, ovaj problem može biti prevladan primjenom serverskog generiranja stranica, što omogućuje poboljšanu vidljivost na webu
- Održavanje JavaScripta stalno aktivnim - ukoliko korisnik onemogući JavaScript u svom pregledniku, to može dovesti do nemogućnosti pravilnog prikazivanja aplikacije i njezinih funkcionalnosti
- Pitanje sigurnosti - u usporedbi s tradicionalnim aplikacijama, jednostranične aplikacije smatraju se manje sigurnima, djelomično zbog ranjivosti na Cross-Site Scripting-u (XSS). Ova ranjivost omogućava potencijalnim napadačima umetanje zlonamjernih skripti na strani klijenta u web aplikaciju drugih korisnika, što može predstavljati sigurnosni rizik

Neki od primjera jednostraničnih web aplikacija su Google Maps, Gmail, Airbnb, Netflix, Pinterest i mnoge druge.

Unatoč standardiziranim komponentama weba, web aplikacija se može izgraditi na mnogo različitih načina. Na razvojnom programeru je da odabere razvojno okruženje i tehnologije s kojima će raditi, a prednost je u tome što nema značajnih ograničenja pri oblikovanju arhitekture aplikacija [6].

2.3. Okviri za razvoj SPA

S razvojem interneta i promjenjivim zahtjevima korisnika, web aplikacije postale su sve sofisticiranije i dinamičnije. Kako bi se postiglo optimalno korisničko iskustvo uz minimalno čekanje, koncept višestraničnog pristupa web dizajnu sve više se zamjenjuje jednostraničnim aplikacijama. Ove

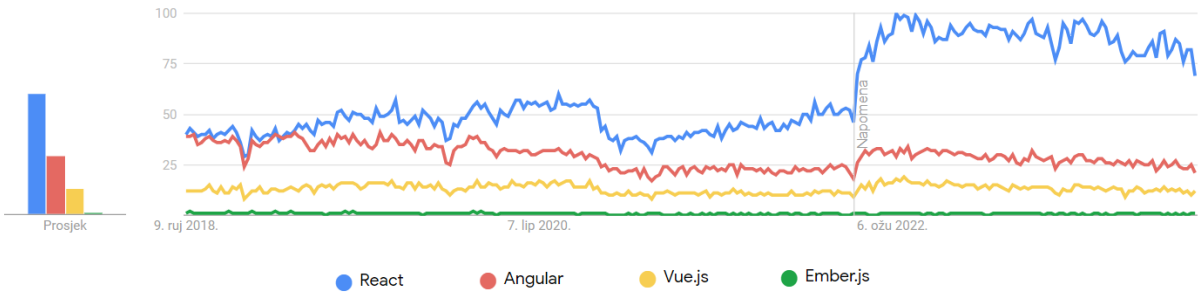
aplikacije nude brz i fluidan doživljaj, sličan radu izvornih[3] aplikacija na mobilnim uređajima. Da bi se postigle brze performanse aplikacija, SPA se oslanjaju na moćne razvojne okvire koji olakšavaju njihov razvoj i održavanje. Neki od najpopularnijih razvojnih okvira su Ember, React, Vue te Angular koji je korišten u praktičnom dijelu ovog rada te će biti opisan detaljnije u nastavku.

Ember je JavaScript okvir otvorenog koda čija se arhitektura temelji na komponentama. Omogućuje razvojnim programerima stvaranje skalabilnih jednostraničnih web aplikacija uključivanjem uobičajenih idioma, najboljih praksi i uzoraka iz drugih obrazaca ekosustava jednostraničnih aplikacija u okvir. Ember.js omogućuje strukturiranje aplikacije po MVC (eng. *Model-View-Controller*) modelu [7]. Model je odgovoran za upravljanje podacima i poslovnom logikom aplikacije. Prikaz (eng. *view*) predstavlja korisničko sučelje aplikacije, ono što korisnik vidi i s čime komunicira. Kontrolor (eng. *controller*) upravlja logikom između modela i prikaza. Kontrolori u Ember.js-u često upravljaju korisničkim unosom, događajima i drugim akcijama koje utječu na model ili prikaz [7].

React je JavaScript razvojni okvir otvorenog koda kojeg je razvio Meta (nekadašnji Facebook), a njegovo inicijalno izdanje bilo je 2013. godine [8]. Okvir se ističe svojim vlastitim virtualnim DOM-om, koji garantira stabilne performanse aplikacija kako se one razvijaju i šire. Njegova arhitektura temeljena je na komponentama. React se fokusira na „V“ u MVC modelu, no uz biblioteke ili okvire poput Reduxa ili MobXa može se upravljati i model i kontrolor segmentima [9].

Vue je također JavaScript okvir otvorenog koda. Ima inkrementalno prilagodljivu arhitekturu koja se fokusira na deklarativno prikazivanje i sastav komponenti. Osnovna biblioteka fokusirana je na prezentacijski sloj. Vue komponente proširuju osnovne HTML elemente za rezimiranje koda koji se može ponovno koristiti. Vue se također fokusira na „V“ u MVC modelu, no uz korištenje Vuex biblioteke podržava segmente modela i kontrolora [10].

Na slici 1 prikazana je popularnost navedenih razvojnih okvira tokom posljednjih 5 godina. Jasno je vidljivo kako je React najpopularniji, posebice posljednjih godinu dana, a odmah iza njega je Angular.



Slika 1 Popularnost razvojnih okvira posljednjih 5 godina [11]

3. Razvoj aplikacije

U sklopu praktičnog dijela ovog rada izrađena je web aplikacija pod nazivom „Croatian Cosmetics“. Svrha ove aplikacije je pružiti korisnicima sveobuhvatnu platformu posvećenu kozmetičkim

[3] izvorna aplikacija je aplikacija koja je napisana specifično za određeni operativni sustav čime se ostvaruje bolja integracija sa operativnim sustavom, visoke performanse te visoka razina sigurnosti.

proizvodima hrvatskih brendova. Da bi se postigla intuitivnost i dinamičnost, za izradu aplikacije korišten je Angular razvojni okvir.

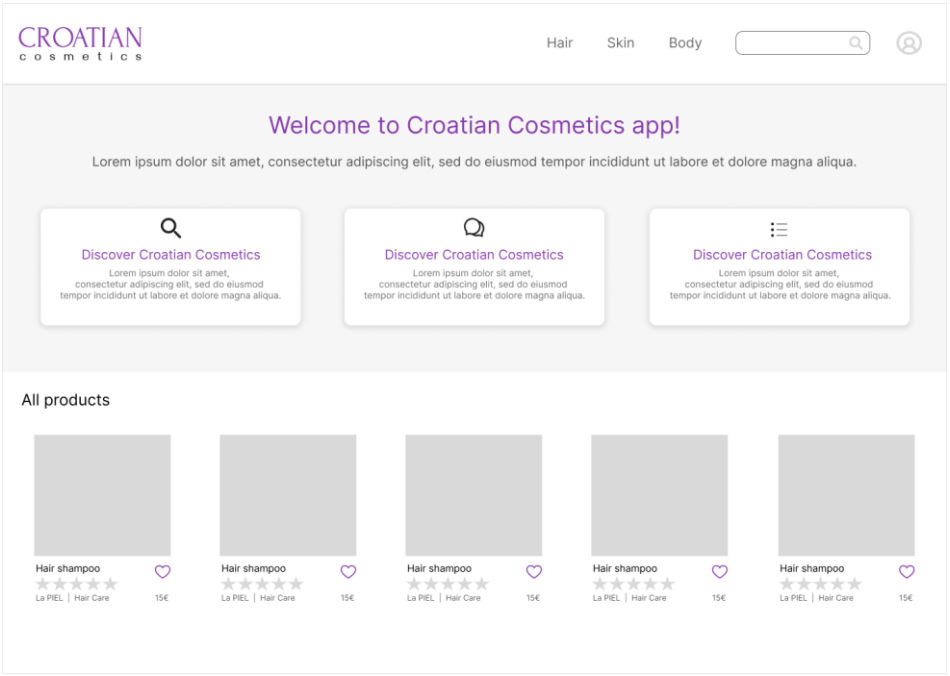
„Croatian Cosmetics“ nudi korisnicima mogućnost pretrage proizvoda kroz različite kategorije: proizvodi za kosu, proizvodi za tijelo te proizvodi za lice. Uz kategoriziranu pretragu, implementirana je i tražilica koja omogućuje korisnicima pretraživanje proizvoda po njihovom nazivu.
Osim same pretrage, aplikacija pruža interaktivnost korisnicima kroz mogućnost ocjenjivanja proizvoda. Na ovaj način korisnici koji pretražuju proizvode mogu vidjeti kako su ih drugi korisnici ocijenili. Korisnicima je također pružena mogućnost izrade vlastite liste želja, gdje samo jednim klikom mogu dodati željeni proizvod u svoju listu. Za ocjenjivanje proizvoda i kreiranje liste želja korisnik se mora registrirati u web aplikaciju.

Kreiranje multimedijskih i hipermedijskih aplikacija, uključujući web sjedišta, zahtijeva pažljivu specifikaciju, odabir tima, razvoj koda i dokumentacije te testiranje. Ključni aspekt je prikupljanje i izrada multimedijskih sadržaja poput grafike, animacija i videa te uspostava hiperveza. Dizajn hipermedijske aplikacije često zahtijeva više vremena nego samo kodiranje. Posebna pažnja posvećuje se hipertekstu i navigaciji. Postoje različiti modeli za razvoj softvera, poput vodopadnog modela, agilne metode i drugih. ADDIE model (eng. *Analysis, Design, Development, Implementation, Evaluation*) je primjeren za razvoj hipermedijskih aplikacija i koristan za jednostavnije projekte, a posebno za početnike [12].

Za potrebe izrade praktičnog dijela ovog rada pratile su se tri faze razvoja, koje su vodile od konceptualne ideje do funkcionalnog digitalnog rješenja.

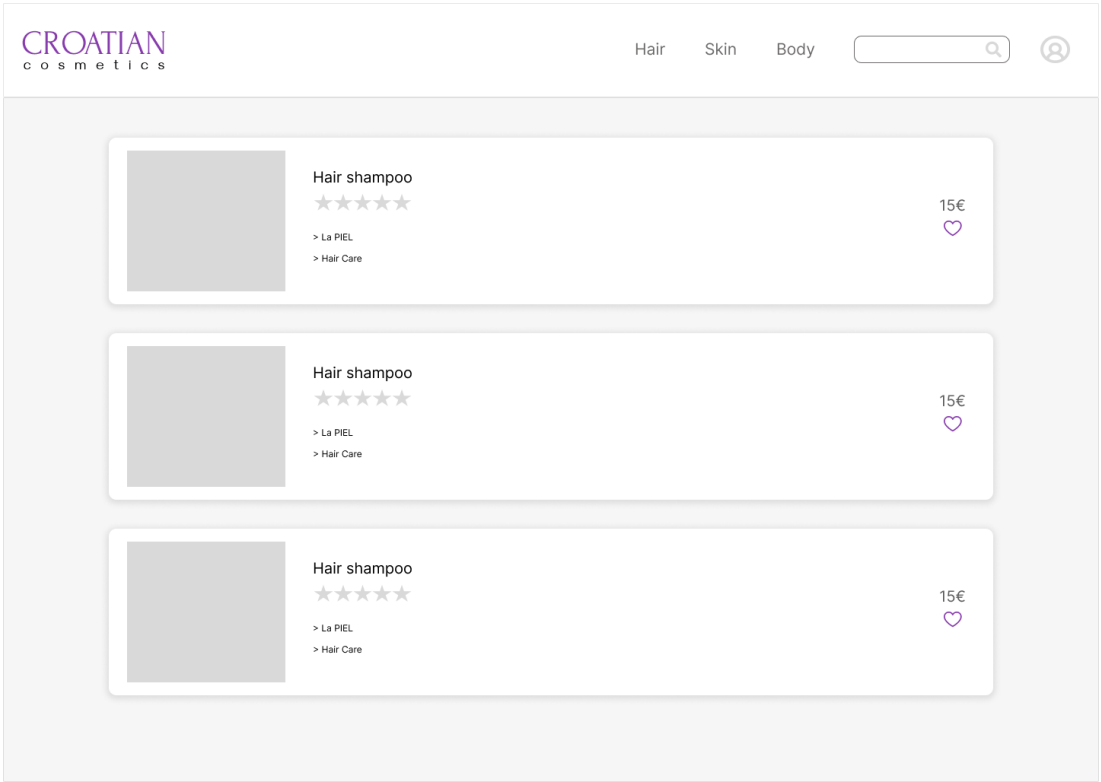
Prva faza bila je faza dizajna. Tokom ove faze osmišljena je tema i konceptualna vizija aplikacije. S obzirom na to da je dizajn ključna komponenta koja određuje korisničko iskustvo, posebna pažnja usmjerena je prema kreiranju glavnih korisničkih sučelja koje će aplikacija sadržavati. Izrađeni su „mockupovi“ unutar online alata Figma. Figma je besplatan alat dostupan kao web aplikacija, a služi za dizajn korisničkog sučelja i korisničkog iskustva te omogućuje suradnju između više dizajnera istovremeno [13]. Mockup je skica koja vjerno prikazuje pravi raspored stranice, sa svim elementima dizajna. U ovakvoj skici često se koristi izmišljeni tekst koji započinje s „Lorem ipsum...“ [12].

Na slici 2 prikazan je mockup početne stranice web aplikacije „Croatian Cosmetics“. Na njoj su vidljivi svi glavni elementi stranice, poput zaglavlja, pozdravne poruke te lista proizvoda. Zaglavlje aplikacije koje sadržava navigaciju i tražilicu moguće je vidjeti unutar svih stranica aplikacije, odnosno na svim mockupovima.



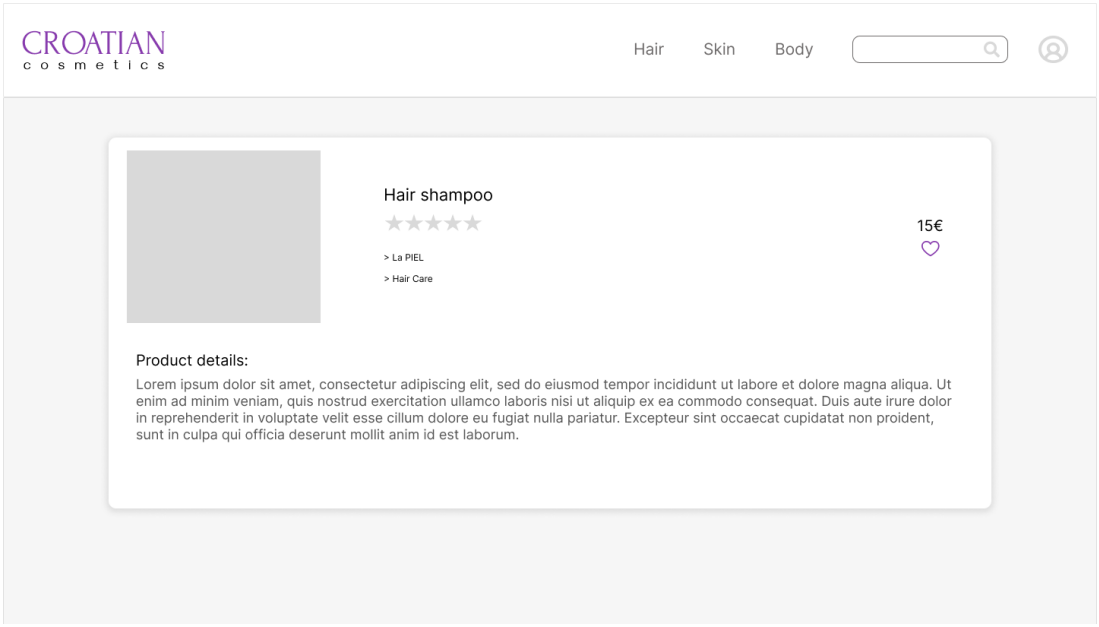
Slika 2 Mockup početne stranice web aplikacije "Croatian Cosmetics"

Slika 3 prikazuje stranicu liste želja. Na njoj je moguće uočiti razliku između načina prikazivanja rasporeda proizvoda u odnosu na početnu stranicu.



Slika 3 Mockup stranice liste želja web aplikacije "Croatian Cosmetics"

Na slici 4 prikazana je stranica detalja proizvoda gdje je moguće saznati nešto više o pojedinačnom proizvodu nego što je to vidljivo na ostalim stranicama aplikacije.



Slika 4 Mockup stranice detalja proizvoda web aplikacije "Croatian Cosmetics"

Na svim prikazanim korisničkim sučeljima korisnik ima mogućnost pregleda ocjene proizvoda i mogućnost ocjenjivanja proizvoda. Obje funkcionalnosti dostižu se kroz simbolični prikaz zvjezdica, gdje svaka zvjezdica predstavlja određeni broj ocjene (1-5) zadovoljstva proizvodom.

Nakon faze dizajna, uslijedila je faza frontend razvoja aplikacije. Frontend je razvoj koji se bavi korisničkim sučeljem i korisničkim iskustvom. Odnosi se na razvoj svega onoga što korisnici vide te kako komuniciraju s aplikacijom. Početne verzije aplikacije „Croatian Cosmetics“ temeljile su se na „hardcodiranim“ podacima, odnosno privremenim podacima implementiranih u „lažni“ servis unutar samog koda. To je omogućilo testiranje funkcionalnosti prije implementacije dinamičkog sadržaja i utvrđivanje njihovih ispravnosti.

Završna faza obuhvaćala je backend. Backend razvoj se bavi aspektima aplikacije koje korisnik ne vidi. Drugim riječima, bavi se aplikacijskom logikom i njenom komunikacijom s bazama podataka. Za pohranjivanje i upravljanje podacima unutar „Croatian Cosmetics“ aplikacije odabran je Firebase Cloud Firestore kao baza podataka. Povezivanje frontenda i backenda omogućilo je dinamičke interakcije s podacima, što pruža korisnicima pristup ažurnim informacijama i omogućujući razmjenu podataka u stvarnom vremenu.

3.1. Korištene tehnologije

Kako bi bolje razumjeli praktični dio izrade web aplikacije prvo je potrebno proučiti sve tehnologije koje se koriste u izradi “Croatian Cosmetics” aplikacije i što svaka omogućuje u tom procesu.

3.1.1. Angular

Angular je suvremeni razvojni okvir otvorenog koda koji je posebno dizajniran za izradu jednostraničnih web aplikacija i dinamičkih web sučelja. Iza njegovog razvoja stoji Google, a predstavlja evoluciju i potpunu preradu svog prethodnika nazvanog AngularJS. Temelji se na TypeScriptu, što je nadograđena verzija JavaScripta koja uključuje statičke tipove i druge napredne značajke [14].

Arhitektura Angulara temelji se na konceptu modularnosti. Aplikacije se razvijaju kroz module koji omogućuju organiziranje funkcionalnosti na visokoj razini. Svaki modul može sadržavati komponente, servise, direktive i cijevi (eng. *pipes*) koji rade zajedno kako bi pružili željenu funkcionalnost. Svaka komponenta se sastoji od HTML predloška koji definira kako će komponenta izgledati, CSS datoteke za stiliziranje HTML dijela, TypeScript koda koji definira ponašanje komponente, te datoteke za testiranje komponente također pisane u TypeScriptu [15].

Angular se ističe svojim bogatim značajkama kao što su dvosmjerna veza podataka (eng. *two-way data binding*), kontrola zavisnosti (eng. *dependency injection*) i snažna podrška za modularni razvoj. Proširuje HTML sintaksu kako bi omogućio dinamičko renderiranje i manipulaciju komponenti. Jedna od glavnih prednosti Angulara je njegova sposobnost dvosmjerne veze podataka. Primjerice, moguće je koristiti *if* uvjete, *for* petlje i lokalne varijable izravno u HTML-u. To omogućuje automatsko ažuriranje korisničkog sučelja svaki put kada se podaci promijene, i obrnuto (kada korisnik promijeni podatke) [15].

Unutar ekosustava Angulara, razvijen je niz alata i biblioteka koje olakšavaju razvoj. Angular CLI je alat koji omogućuje generiranje, testiranje i izgradnju aplikacija. Ima ugrađenu podršku za komunikaciju s pozadinskim poslužiteljem, što web aplikacijama olakšava dohvaćanje i objavljivanje podataka ili izvršavanje poslovne logike na strani poslužitelja.

3.1.2. TypeScript

Typescript je programski jezik koji je nadogradnja ili nadskup jezika JavaScript. TypeScript pruža statički tipizacijski sustav i objektne orijentirane značajke povrh standardnog JavaScripta. Dok JavaScript kao dinamički tipizirani jezik omogućuje fleksibilnost u radu s varijablama i funkcijama, to može dovesti do nepredvidivih grešaka tijekom izvođenja koda. Koristeći TypeScript, mnoge od tih grešaka se identificiraju već u fazi kompilacije, prije nego što se kod stvarno izvede. To može biti korisno za rano otkrivanje potencijalnih grešaka u kodu i omogućuje razvoj stabilnijih i pouzdanijih aplikacija. Kod napisan u TypeScriptu se mora kompilirati u JavaScript kako bi se izvršavao u web preglednicima [16].

Neke od prednosti TypeScripta su [16]:

- Autocomplete - funkcionalnost koja olakšava razvoj na način da predlaže različite metode ili varijable prilikom pisanja koda
- Refaktoriranje - zbog statičke provjere tipova, mijenjanje imena varijable ili funkcije u jednom dijelu koda automatski će se ažurirati kroz cijeli projekt, smanjujući rizik od grešaka
- Statičko tipiziranje - omogućuje eksplicitno definiranje tipova podataka, što pruža dodatnu razinu sigurnosti prilikom pisanja koda

Postoje i neki nedostaci [16]:

- Kompilacija - iako kompilacija ima svoje prednosti, proces kompiliranja može usporiti razvojni proces
- Trajanje pisanja koda - zbog potrebe za definiranjem tipova, pisanje koda može biti sporije u usporedbi s JavaScriptom

3.1.3. HTML i SCSS

HTML (eng. *HyperText Markup Language*) je prezentacijski jezik za izradu web stranica. Ne smatra se programskim jezikom i ne omogućuje izvođenje operacija poput zbrajanja ili oduzimanja. Njegova

glavna svrha je uputiti web pregledniku kako da prikaže hipertekstualni dokument, s ciljem da dokument izgleda konzistentno bez obzira na vrstu web preglednika. Osnovni elementi HTML-a su oznake (eng. *tags*) koje opisuju kako će se sadržaj prikazati u web pregledniku. Poveznice unutar HTML dokumenta omogućuju povezivanje dokumenata u strukturiranu hijerarhiju [17].

SCSS (eng. *Sassy CSS*) je preprocesorski skriptni jezik koji je nadskup CSS-a. Služi stiliziranju web stranica s većom fleksibilnošću i modularnošću od regularnog CSS-a. Sintaksa SCSS-a vrlo je slična CSS-u, ali omogućuje korištenje varijabli, ugnježđivanja, mješavina (eng. *mixins*) i drugih programskih konstrukcija. SCSS ima ekstenziju datoteke *.scss* [18].

3.1.4. Firebase

Firebase je platforma za razvoj aplikacija koju je razvio Google. Pruža različite usluge za izgradnju aplikacija, a neki od ključnih su hosting, autentifikacija, baze podataka u stvarnom vremenu, analitika i testiranje. Firebase je BaaS (eng. *Backend as a Service*), što znači da pruža gotove backend usluge koje se mogu integrirati u aplikacije bez potrebe za izgradnjom i održavanjem vlastite backend infrastrukture [19].

U ovom radu korištene su Cloud Firestore, Firebase Storage, Authentication te Firebase hosting usluge. Svaka od njih opisana je u nastavku.

Cloud Firestore je jedna od usluga baza podataka koje nudi Firebase. Kao što naziv sugerira, Cloud Firestore je nerelacijska baza podataka bazirana u oblaku. Ima dokumentno-orijentiranu strukturu, što znači da podaci nisu pohranjeni u tablicama s redovima i stupcima kao u relacijskim bazama podataka, već u kolekcijama dokumenata. Svaki dokument može sadržavati različite setove podataka, uključujući i složene objekte [19]. Prednost Firestorea je mogućnost ažuriranja podataka unutar kolekcija baze u stvarnom vremenu. Kada se podaci promijene, sve klijentske aplikacije povezane s bazom automatski primaju ažuriranja. Firestore SDK pruža i offline podršku, odnosno omogućuje aplikacijama da funkcioniraju i bez internetske veze, a sinkroniziraju promjene nakon što se veza ponovno uspostavi [19].

Firebase Storage omogućuje pohranjivanje datoteka poput slika ili videozapisa u oblaku. Uz Firebase Storage datoteke je moguće pohraniti u sigurnom i skalabilnom okruženju [19]. U ovom radu Firebase Storage korišten je u kombinaciji s Cloud Firestore bazom podataka za prikaz slika svakog proizvoda unutar aplikacije.

Firebase Authentication pruža backend podršku, SDK-ove (eng. *Software Development Kit*) koji su laki za uporabu i gotove biblioteke korisničkih sučelja za autentifikaciju korisnika u aplikaciji [19]. Ova usluga integrirana je u „Croatian Cosmetics“ aplikaciju u svrhu registracije i prijave korisnika u aplikaciju za potrebe ostavljanja recenzija i kreiranja lista želja, koje moraju biti povezane s određenim korisnikom.

Firebase Hosting predstavlja hosting uslugu za web sadržaj kojom razvojni tim može na brz način postaviti i pružiti statički i dinamički sadržaj putem globalnog CDN-a[4] (eng. *content delivery network*). Jedan od ključnih prednosti Firebase Hostinga je sigurnost koju pruža zahvaljujući ugrađenom SSL-u[5] (eng. *Secure Socket Layer*) [19].

[4] Globalni CDN je grupa geografski raspoređenih poslužitelja koji ubrzavaju isporuku web sadržaja

[5] SSL je standardna tehnologija za osiguranje internetske veze šifriranjem podataka koji se šalju između web stranice i preglednika.

3.1.5. Node.js

Node.js je poslužiteljsko okruženje otvorenog koda koje omogućuje izvršavanje JavaScript koda izvan konteksta web preglednika. U kontekstu web razvoja, tradicionalno se JavaScript koristio isključivo na klijentskoj strani, odnosno unutar web preglednika, za manipulaciju DOM-om i interakciju s korisnikom. Uz pomoć Node.js-a, JavaScript je proširen na poslužiteljsku stranu, odnosno služi kao backend rješenje koje upravlja bazama podataka [20].

Uz instalaciju Node.js-a dolazi i NPM (eng. *Node Package Manager*). Npm je upravitelj paketa koji omogućuje preuzimanje, instalaciju i upravljanje vanjskim bibliotekama i alatima potrebnima za razvoj aplikacije. Biblioteke mogu biti od jednostavnih alata do složenih okvira koji pružaju funkcionalnosti poput rutiranja (eng. *routing*), upravljanja bazama podataka ili autentifikacije korisnika [21].

3.1.6. Git i Github

Git je besplatni distribuirani sustav kontrole verzija otvorenog koda dizajniran za brzu i učinkovitu obradu svih informacija. Razvijen je 2005. godine od strane Linusa Torvaldsa, tvorca Linux operativnog sustava, s ciljem olakšavanja suradnje među programerima i efikasnijeg upravljanja izvornim kodom [22]. U procesu razvoja koda važno je pratiti verzije istog kako bi se osigurala konzistentnost, omogućilo vraćanje na prethodna stanja u slučaju grešaka i konflikata, te kako bi se olakšala suradnja programera koji istovremeno rade na istom projektu. Git, osim što bilježi promjene u kodu, bilježi i vrijeme kada je promjena napravljena te tko ju je napravio.

Prednost korištenja Gita je njegova podrška za nelinearno razvijanje što omogućava razvojnom timu paralelan rad na različitim dijelovima koda koje se naknadno mogu spojiti s glavnim kodom. Takvi dijelovi koda razvijaju se na tzv. granama (eng. *branches*), te se spajaju sa glavnom granom kada je to potrebno. Također, Git pruža mogućnost integracije s online platformama kao što su GitHub, GitLab i BitBucket kako bi razvojni tim mogao surađivati na projektima u oblaku [22].

Za potrebe praktičnog dijela ovog rada korišteno je Git verzioniranje uz online platformu GitHub. Nakon što je kreiran Angular projekt unutar lokalnog repozitorija, kreiran je repozitorij unutar GitHuba. Nakon toga, prvi korak za korištenje Git verzioniranja bio je inicijalizirati ga unutar lokalnog repozitorija projekta kroz naredbeni redak, korištenjem naredbe *git init*. Sljedeći korak bio je spojiti lokalni repozitorij s GitHub online repozitorijem naredbom *git remote add origin [URL]*, gdje je [URL] stvarna adresa repozitorija na GitHubu. U ovom projektu korištena je samo jedna, odnosno glavna *master* grana. Nakon uspostave veze između lokalnog i online repozitorija, sve promjene napravljene lokalno mogle su se prenijeti na online repozitorij nizom *commit* i *push* operacija uz odgovarajući opis za svaki prijenos.

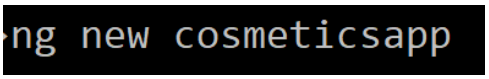
3.2. Priprema radnog okruženja

Nakon određivanja tehnologija s kojima će se raditi za razvoj web aplikacije, potrebno je odabrati i softverski program unutar kojeg će se aplikacija razvijati. U ovom radu korišten je Microsoftov Visual Studio Code, moćan uređivač koda dostupan kao desktop aplikacija za Windows, Linux i macOS. Dolazi s ugrađenom podrškom za programske jezike JavaScript, TypeScript i Node.js, no pruža i mogućnost proširenja za druge jezike i okruženja kao što su C++, C#, Java, Python, PHP, Go i .Net [23].

Nakon što su pripremljeni alati za izradu aplikacije, poput instalacije Visual Studio Code-a i Node.js-a, potrebno je bilo instalirati Angular CLI (eng. *Command Line Interface*) korištenjem naredbe „npm install

-g @angular/cli“ unutar lokalnog repozitorija. Angular CLI je alat za sučelje naredbenog retka koji se koristi za inicijalizaciju, razvoj i održavanje Angular aplikacija [15].

Kreiranje nove Angular aplikacije izvršava se jednostavnom naredbom *ng new naziv*, gdje je naziv u ovom slučaju *cosmeticsapp*, kako je prikazano na sljedećoj slici.



Slika 5 Naredba za kreiranje Angular projekta

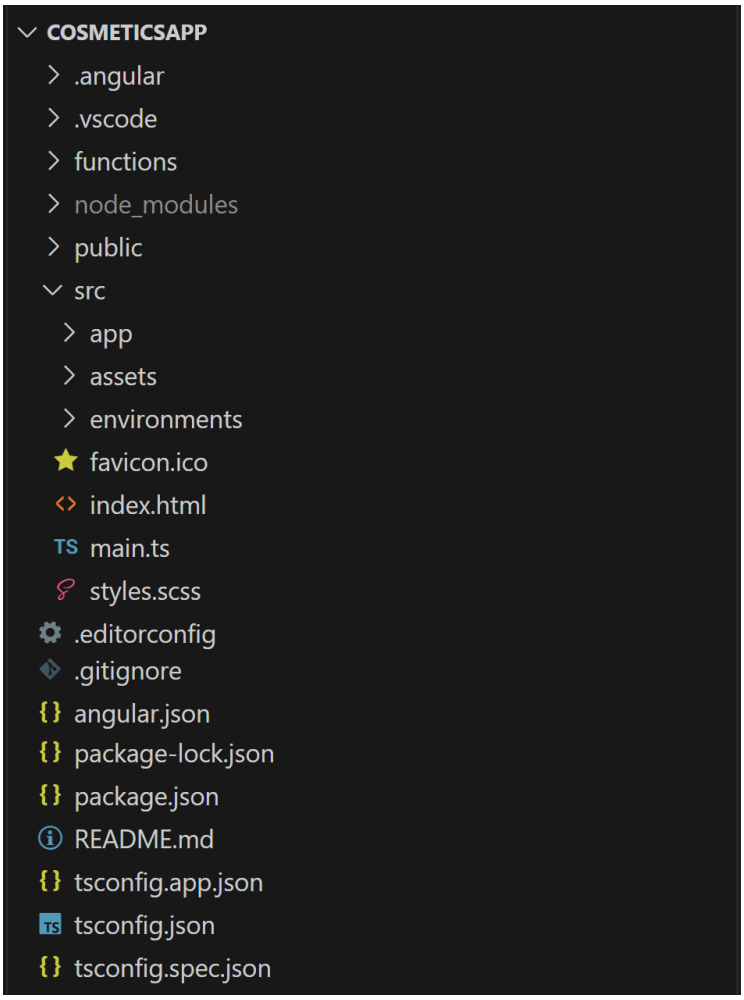
Pokretanjem prethodne naredbe kreirala se dodatna datoteka unutar lokalnog repozitorija koja sadrži sve datoteke i resurse projekta.

Sljedeće naredbe unutar naredbenog retka služile su za lociranje unutar novokreirane datoteke i njeno otvaranje unutar Visual Studio Code programa.

Slika 6 Otvaranje rezitorija unutar Visual Studio Code aplikacije

Početni izgled Angular repozitorija prikazan je na slici 7, a njegovi glavni dijelovi su:

- node_modules – direktorij koji sadrži sve vanjske biblioteke i module potrebne za rad Angular aplikacije. Svi paketi instalirani putem npm-a pohranjuju se u ovom direktoriju
- src – glavni direktorij koji sadrži osnovne komponente aplikacije poput glavne komponente, modula i početne stranice
- assets – direktorij za pohranu statičkih resursa poput slika ili ikona
- environments – sadrži konfiguracijske datoteke za različita okruženja razvoja (npr. testno okruženje, produkcijsko okruženje)
- angular.json – konfiguracijska datoteka za Angular CLI koja služi za definiranje npr. stilova ili skripti
- package.json – datoteka koja sadrži meta podatke o projektu i popis zavisnosti potrebnih za rad aplikacije
- README.md – osnovna dokumentacijska datoteka koja obično sadrži informacije o projektu, kako ga postaviti, pokrenuti i slično



Slika 7 Početni izgled angular repozitorija

Nakon postavljanja osnovne strukture projekta, sljedeći korak bio je dodavanje komponenti i modula koji će činiti osnovu aplikacije. Unutar *app* direktorija sadržane su sve datoteke specifične za ovu aplikaciju. Korištenjem Angular CLI, moguće je na brz način generirati komponente, servise, direktive i druge bitne dijelove aplikacije. Na primjer, za kreiranje nove komponente koristi se naredba *ng*

generate component ime-komponente, ili skraćeno *ng g c ime-komponente*, a ona će stvoriti datoteke prikazane na slici 8.

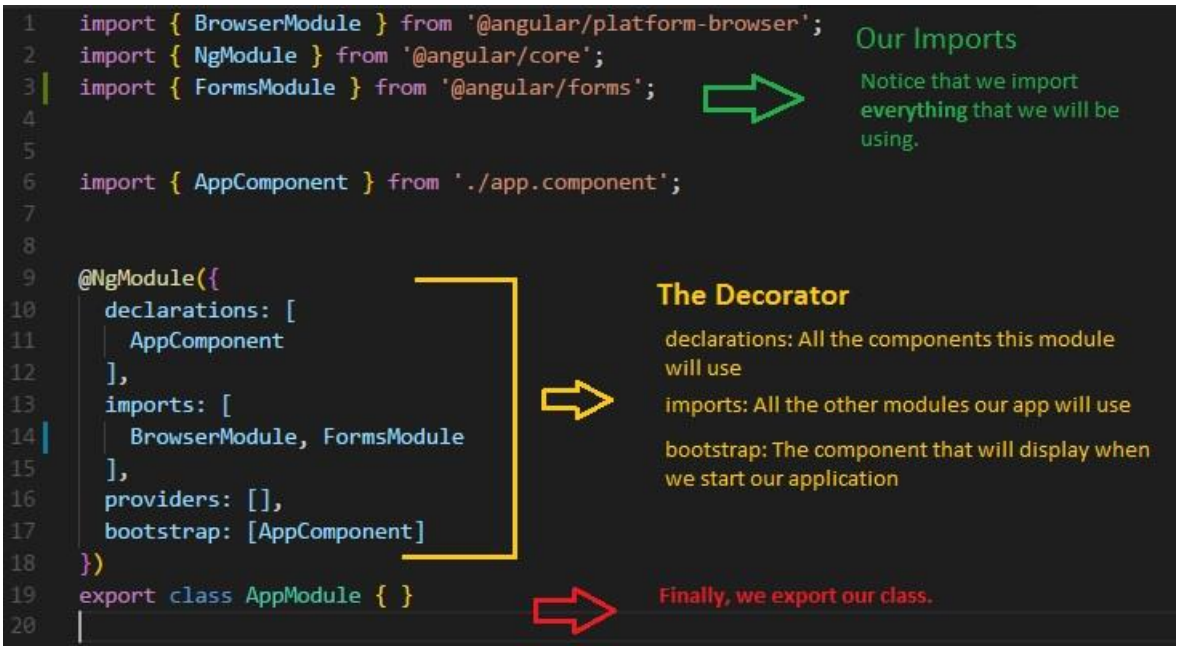
Slika 8 Dijelovi komponente projekta u Angularu

Ime-komponente.component.ts je glavna TypeScript datoteka komponente gdje se definira njena logika. HTML predložak definira se u *ime-komponente.component.html* datoteci, a stiliziranje u *ime-komponente.component.scss* datoteci. Generiranjem komponente pomoću Angular CLI-ja, generirat će se i *ime-komponente.component.spec.ts* datoteka koja služi za pisanje jediničnih testova koji provjeravaju ispravnost te komponente. Iako postoje automatski generirane *spec.ts* datoteke, u ovom radu testiranje se nije izvršavalo na ovaj način već ručnim testiranjem tokom cijelog procesa razvoja aplikacije.

3.2.1. Izrada komponente

Vodeći se mockup dizajnom aplikacije, stvorene su određene komponente od kojih svaka ima svoju ulogu, a detaljnije su objašnjene u nastavku rada. Svaku komponentu potrebno je registrirati unutar glavnog modula aplikacije koji se obično naziva AppModule, kao što je to i u ovom slučaju.

Ovaj modul služi kao središnja točka koja definira komponente, direktive, cijevi i servise koji su korišteni unutar aplikacije. Slika 9 prikazuje primjer izgleda AppModule-a, u kojem postoji nekoliko različitih stavki. U prvoj sekciji dekoratora *@NgModule* (označeno žutom bojom na slici), unutar *declarations* liste navode se sve komponente, direktive i cijevi. U *imports* sekciji navode se svi moduli koji se uvoze u aplikaciju, poput uvođenja formularne funkcionalnosti *FormsModule*. Ovdje se također mogu uvesti i moduli trećih strana, poput PrimeNg-a čija uloga se spominje kasnije u ovom radu. U *providers* sekciji navode se servisi koji će biti dostupni kroz cijelu aplikaciju. Kada se servis registrira na razini AppModule-a, on postaje *singleton*, što znači da postoji samo jedna instanca tog servisa kroz cijelu aplikaciju. *Bootstrap* sekcija definira koja će komponenta biti pokrenuta prva kada se aplikacija učita. U većini slučajeva je to AppComponent, kao i u ovom radu. Iako je AppModule glavni modul u većini Angular aplikacija, veće i složenije aplikacije često koriste dodatne module kako bi bolje organizirale svoj kod. Ovi moduli, poznati kao feature moduli, omogućuju razdvajanje funkcionalnosti aplikacije u manje dijelove što olakšava razvoj, testiranje i održavanje aplikacije.



Slika 9 Izgled glavnog modula, AppModule [24]

Kada se nova komponenta generira pomoću Angular CLI, automatski se dodaje u *declarations* polje AppModule-a, a ručnim dodavanjem komponente potrebno je i ručno dodati referencu unutar ovog modula. Klasa postaje Angular komponenta kada joj se definiraju meta podaci. Meta podaci se dodaju klasi pomoću dekoratora u TypeScript datoteci. Dekorator sadrži različite konfiguracijske opcije koje određuju kako će se komponenta ponašati i kako će biti prikazana. Za definiranje klase kao komponente koristi se oznaka *@Component*. Pri tome je potrebno uvesti *Component* dekorator iz *@angular/core* modula. Na slici 10 prikazan je osnovni izgled TypeScript datoteke komponente koji sadržava: selektor, predložak (eng. *template*) i stilove. Navedeni osnovni izgled dio je spomenutog dekoratora. Selektor određuje HTML oznaku kojom se komponenta može referencirati unutar HTML datoteka drugih komponenti. Primjerice, ako je selektoru postavljen naziv *'app-root'* kao na slici 10, tada se *<app-root> </app-root>* oznaka može koristiti unutar HTML-a neke druge komponente da bi se u njoj prikazala. *TemplateUrl* ukazuje na putanju do HTML predloška koji definira izgled komponente. *StyleUrls* ukazuje na putanju do SCSS datoteke ili datoteka koje definiraju stilove za ovu komponentu.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'cosmeticsapp';
}
```

Slika 10 Izgled komponente

3.3. Integracija komponenti

Nakon što su komponente registrirane u glavnom modulu aplikacije (AppModule), mogu se koristiti unutar drugih komponenti ili predložaka. Kako bi se strukturirala ova aplikacija, koristi se Angular Router modul. Angular Router omogućuje navigaciju iz jednog pogleda na drugi dok korisnici izvode zadatke kao što su klik na link, ispunjavanje formi ili pretraživanje sadržaja. Router je ključan za jednostranične aplikacije gdje sve stranice postoje unutar jedne glavne datoteke, ali se dinamički mijenjaju bez potrebe za ponovnim učitavanjem cijele stranice. Da bi se koristio Angular Router potrebno je unutar app-routing.module.ts datoteke definirati rute aplikacije. Svaka ruta povezana je s određenom komponentom koja će se prikazati kada se korisnik navigira na određeni URL. Sve rute, odnosno URL-ovi koji su definirani u ovoj aplikaciji prikazani su na slici 10 unutar *Routes* liste objekata. Tako je, primjerice, za početnu stranice aplikacije (HomeComponent) definirana ruta s praznim putem (*path: ' '*), što znači da će se HomeComponent prikazati kada korisnik posjeti osnovni URL aplikacije (npr. http://domena.com/). Za prikaz detalja proizvoda koristi se ProductDetailsComponent, a ruta je definirana s *path: 'product/:id'*. Ovdje je *:id* dinamički segment rute koji omogućuje da se prenese određeni identifikator proizvoda kao parametar u URL-u. Na primjer, ako korisnik posjeti http://domena.com/product/5, ProductDetailsComponent će se prikazati s detaljima proizvoda s ID-jem 5. Slično tome, ruta *path: 'category/:categoryName'* omogućuje dinamičko prenošenje imena kategorije kao dijela URL-a, što će rezultirati prikazom *CategoryComponent* komponente s proizvodima te kategorije. Da bi Angular Router mogao prepoznati i koristiti ove rute, potrebno je dodati RouterModule u *imports* sekciju AppRoutingModule-a i konfigurirati ga s *routes* listom koristeći *RouterModule.forRoot(routes).* U glavnom modulu aplikacije (AppModule) potrebno je uvesti ovaj routing modul, odnosno *AppRoutingModule*.

app-routing.module.ts

src > app > app-routing.module.ts > ...

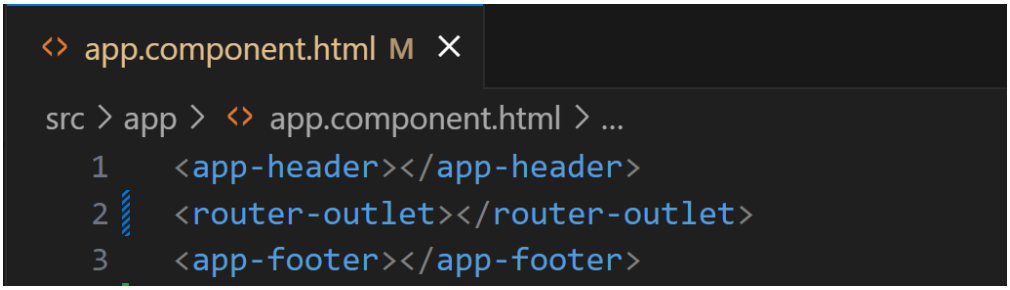
```
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3  import { HomeComponent } from './home/home.component';
4  import { ProductDetailsComponent } from './product-details/product-details.component';
5  import { WishlistPageComponent } from './wishlist-page/wishlist-page.component';
6  import { LoginComponent } from './login/login.component';
7  import { RegistrationComponent } from './registration/registration.component';
8  import { CategoryComponent } from './product-category/category/category.component';
9
10 const routes: Routes = [
11   {path:'', component: HomeComponent},
12   {path:'product/:id', component: ProductDetailsComponent},
13   {path:'wishlist',component: WishlistPageComponent},
14   {path:'login', component: LoginComponent},
15   {path:'register', component: RegistrationComponent},
16   {path:'category/:categoryName', component: CategoryComponent}
17 ];
18
19 @NgModule({
20   imports: [RouterModule.forRoot(routes)],
21   exports: [RouterModule]
22 })
23 export class AppRoutingModule { }
24
```

Slika 11 Angular Router modul

Također, potrebno je napomenuti kako modul nije isto što i komponenta. Tako se ovdje može vidjeti *@NgModule* dekorator koji označava da se ovdje radi o modulu. Modul je zapravo skup funkcionalnosti koji je grupiran zajedno. U Angularu, moduli se koriste za organiziranje komponenata, servisa, direktiva i drugih dijelova aplikacije u koherentne cjeline za modularnost i lakše upravljanje funkcionalnostima aplikacije.

Nakon što su rute definirane i konfigurirane, može se koristiti Angular direktiva *<router-outlet></router-outlet>* ovog modula unutar glavnog HTML predloška kako bi se dinamički prikazale komponente na temelju trenutnog URL-a. Glavni HTML predložak je *app.component.html,* uz kojeg postoje i pripadne TypeScript, SCSS i SPEC datoteke.

Na slici 12 prikazan je app.component.html, u kojem *<app-header></app-header>* predstavlja zaglavlje aplikacije koje se prikazuje na svim stranicama aplikacije. <router-outlet></router-outlet> je direktiva koja omogućava Angular Routeru da dinamički prikazuje komponente aplikacije na temelju trenutne rute, a koje su opisane kasnije. Na primjer, ako korisnik posjeti početnu stranicu, komponenta koja je povezana s tom rutom (u ovom slučaju HomeComponent) će se automatski prikazati na mjestu gdje se nalazi <router-outlet></router-outlet> direktiva. Posljednja direktiva na slici, *<app-footer></app-footer>* predstavlja podnožje aplikacije koje se prikazuje na svim stranicama aplikacije, kao i zaglavlje. Ovaj pristup omogućuje centraliziranu organizaciju i dinamičko prikazivanje sadržaja na jednoj stranici, što je ključno za jednostranične aplikacije. Korištenjem ove strukture, mogu se lako dodavati, mijenjati ili uklanjati komponente i rute bez potrebe za velikim preinakama u osnovnom predlošku aplikacije.



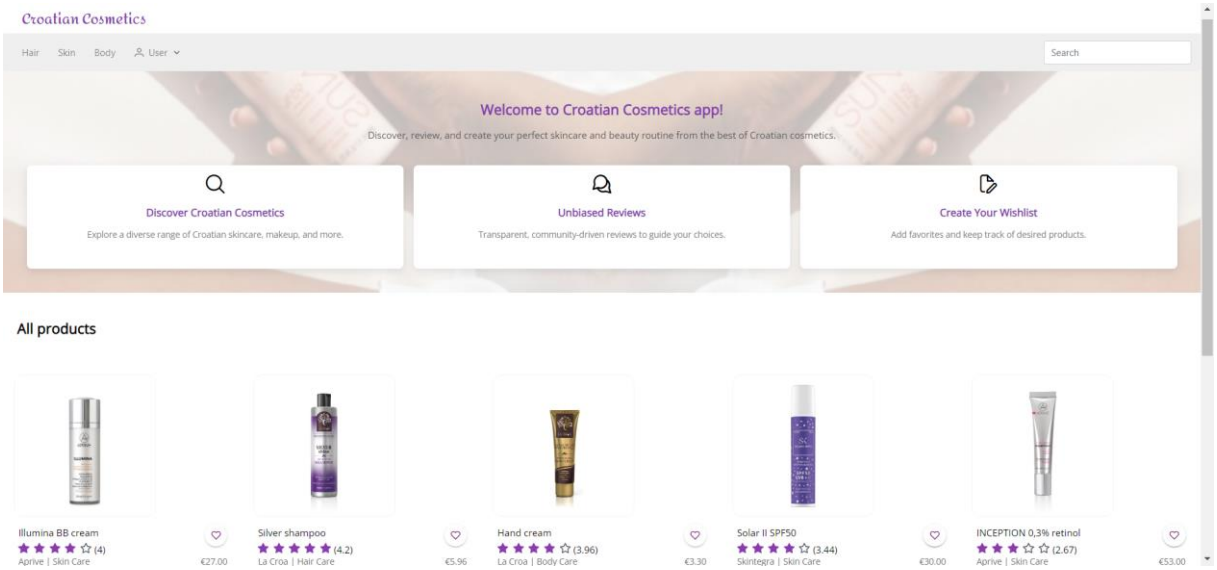
Slika 12 HTML predložak komponente AppComponent, glavne komponente aplikacije

Komponenta podnožja aplikacije kreirana je zbog vizualno ugodnijeg doživljaja aplikacije. Njena svrha nije funkcionalnost, već estetika. U korisničkom sučelju ona sadržava minimalistički dizajn prikaza naziva aplikacije, „Croatian Cosmetics“, jednak dizajnu naslova u zaglavlju aplikacije.

3.3.1. Glavne komponente

Komponenta početne stranice

Jedva od glavnih komponenti „Croatian Cosmetics“ aplikacije je Home komponenta. Kada korisnik pristupi osnovnom URL-u aplikacije, Home komponenta je prva koja se prikazuje. Ova komponenta služi kao početna točka za korisnike, pružajući im pregled onoga što aplikacija nudi i sve proizvode koji se na njoj mogu pronaći. Prvo što će korisnik primijetiti na početnoj stranici aplikacije je uvodna sekcija. Ova sekcija sadrži naslov koji pozdravlja korisnike i kratki opis koji ih informira o svrsi aplikacije. Korisničko sučelje početne aplikacije prikazano je na sljedećoj slici.



Slika 13 Korisničko sučelje Home komponente

Ispod uvodne sekcije, korisnici mogu pregledavati sve dostupne proizvode. Svaki proizvod prikazan je s njegovom slikom, imenom, kategorijom, cijenom i prosječnom ocjenom. Korisnici također mogu dodati proizvode na svoju listu želja ili ocijeniti proizvode izravno s početne stranice.

Kada korisnik klikne na sliku proizvoda, bit će preusmjeren na stranicu s detaljima proizvoda gdje može saznati više o odabranom proizvodu. Ovo je ostvareno pomoću Angular Routera koji omogućuje programsku navigaciju koristeći *Router* servis i deklarativnu navigaciju koristeći *<a [routerLink]* direktivu prikazanu na slici 14. Na dnu sekcije s proizvodima nalazi se paginator koji omogućuje korisnicima da prelaze između stranica proizvoda. Paginator je dinamički i prilagođava se ukupnom broju dostupnih proizvoda. Ovo je postignuto korištenjem *<p-paginator></p-paginator>* PrimeNg oznake koju je prethodno bilo potrebno uvesti i konfigurirati unutar aplikacije. Paginator omogućuje korisnicima prilagodbu broja prikazanih proizvoda po stranici, koji u ovom slučaju mogu biti 10, 20 ili 30.

```
<h2 style="padding: 1.5rem;">All products</h2>
<ul class="image-grid">
    <li *ngFor="let product of displayedProducts">
        <a [routerLink]="['/product', product.id]"><img src="{{product.imageUrl}}" alt="product images"></a>
        <div class="content">
            <div class="name">
                {{product.name}}
            </div>
            <div class="favorite">
                <app-wishlist-button [product]="product"></app-wishlist-button>
            </div>
            <div class="star">
                <app-rating [productId]="product.id"
                    (ratingChange)="handleRatingChange(product.id, $event)"></app-rating>
            </div>
            <div class="product-item-footer">
                <div class="category">{{product.brand}} | {{product.category}}
                </div>
                <div class="price">{{product.price | currency:'EUR'}}</div>
            </div>
        </div>
    </li>
</ul>
<div class="card flex justify-content-center">
    <p-paginator (onPageChange)="onPageChange($event)" [first]="first" [rows]="rows" [totalRecords]="products.length"
        [rowsPerPageOptions]="[10, 20, 30]"></p-paginator>
</div>
```

Slika 14 Dio HTML predloška početne stranice

PrimeNg je popularna biblioteka komponenata korisničkih sučelja za Angular aplikacije koja nudi širok spektar gotovih komponenata poput tablica, dijaloga, grafova i gumbova. Da bi se PrimeNg komponente koristile, prvo je bilo potrebno instalirati PrimeNg paket i dodati ga u Angular projekt. Nakon instalacije, potrebno je uvesti željeni modul iz PrimeNg-a u glavni modul aplikacije (AppModule).

Home komponenta koristi *ProductsService* servis za dohvaćanje svih proizvoda koji se zatim prikazuju korisnicima. Kada korisnik mijenja stranicu putem paginatora, prikazani proizvodi se ažuriraju kako bi odražavali odabrane proizvode za tu stranicu. Važno je napomenuti da svaka komponenta u Angularu ima svoj životni ciklus, a postoji niz metoda životnog ciklusa koje se mogu koristiti kako bi se izvršile određene akcije u različitim fazama životnog ciklusa komponente, poput ngOnInit, ngOnChanges i ngOnDestroy. Metoda ngOnInit se koristi za postavljanje početnih vrijednosti i dohvaćanje podataka s poslužitelja. Na slici 15 prikazana je *ngOnInit* metoda unutar *Home* komponente koja se koristi za dohvaćanje proizvoda iz servisa *ProductsService*. Kada se podaci dohvate, popis proizvoda se ažurira i zatim se poziva metoda *updateDisplayedProducts* kako bi se ažurirala lista proizvoda unutar paginatora.

Također, komponenta koristi *RatingService* servis za dohvaćanje prosječnih ocjena za svaki proizvod. Kada korisnik ocijeni proizvod, prosječna ocjena za taj proizvod se ažurira u stvarnom vremenu.

```typescript
ngOnInit(): void {
  this.productsService.getProducts().subscribe((products) => {
    this.products = products;
    this.updateDisplayedProducts();
  });
}

viewProductDetails(productId: string) {
  this.router.navigate(['/products', productId]);
}

onPageChange(event: any) {
  console.log(event);
  this.first = event.first;
  this.rows = event.rows;
  this.updateDisplayedProducts();
}

updateDisplayedProducts() {
  this.displayedProducts = this.products.slice(this.first, this.first + this.rows);
}

handleRatingChange(productId: string, newRatingValue: number): void {
  console.log(`Product with ID: ${productId} received a new rating of ${newRatingValue}`);
  this.ratingService.getAverageProductRating(productId).subscribe(avgRating => {
    this.averageRatings[productId] = avgRating;
  });
}
```

Slika 15 TypeScript kod Home komponente

Komponenta zaglavlja

Komponenta zaglavlja, odnosno Header komponenta predstavlja zaglavlje web aplikacije. Ova komponenta obavlja mnoge funkcije ključne za bolje korisničko iskustvo. To uključuje autentifikaciju korisnika, pretraživanje proizvoda i navigiranje kroz kategorije proizvoda. Iz tog razloga potrebno je u konstruktor TypeScript datoteke uvesti odgovarajuće servise prikazane na slici 16. U *ngOnInit* metodi provjerava se autentifikacija korisnika i postavljaju se odgovarajuće stavke izbornika, ovisno o tome je li korisnik prijavljen ili ne. Točnije, ukoliko korisnik nije prijavljen, tada će u izborniku vidjeti *Login* gumb za prijavu, a u suprotnom će vidjeti svoj email na čiji klik će mu se otvoriti padajući izbornik. U padajućem izborniku korisniku su dostupni gumb za vlastitu listu želja te gumb za odjavljivanje.

```
constructor(private authService: AuthService, private router: Router, private productService: ProductsService,
  private elRef: ElementRef) { }

isMenuActive = false;

ngOnInit(): void {
  this.authService.isAuthenticated().subscribe((authenticated) => {
    if (authenticated) {
      this.authService.getCurrentUser().subscribe((user) => {
        this.user = user;
        this.userEmail = user ? user.email : null;
        this.setMenuItems(true);
      });
    } else {
      this.user = null;
      this.userEmail = null;
      this.setMenuItems(false);
    }
  });
}
```

Slika 16 TypeScript kod Header komponente

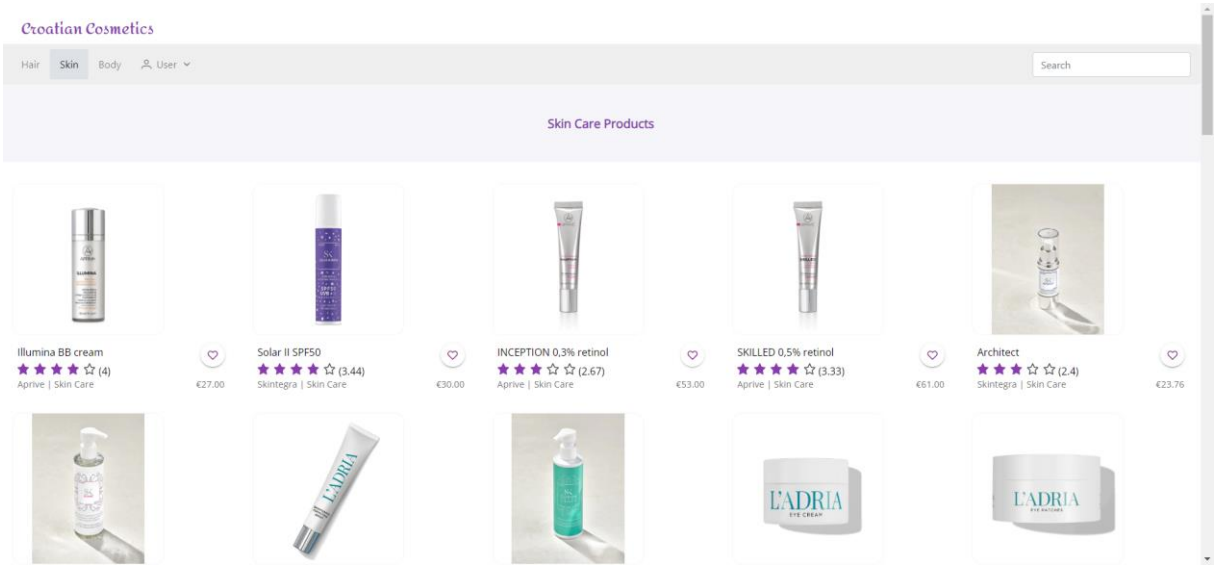
Nadalje, u zaglavlju aplikacije postoji traka za pretraživanje. Metoda *onSearch* prikazana na sljedećoj slici, koja upravlja funkcionalnošću ove trake, prima tekstualni unos korisnika kao argument, nazvan *inputValue*. Provjerava se postojanost unesene vrijednosti te se osigurava da nije riječ o praznom nizu nakon uklanjanja razmaka. Nakon validacije, metoda koristi servis, točnije *searchProductsByName* funkciju unutar *ProductsService*, kako bi dohvatila proizvode koji odgovaraju kriterijima pretraživanja definiranima unutar *searchProductsByName* metode servisa. Kada se podaci uspješno dohvate s poslužitelja, koristi se asinkrona metoda *subscribe* kako bi se obradili i pohranili unutar svojstva *productsForDropdown*. Ovo svojstvo zatim služi za dinamičko prikazivanje rezultata pretraživanja korisniku.

```
onSearch(inputValue: string): void {
  if (inputValue && inputValue.trim() !== '') {
    this.productService.searchProductsByName(inputValue).subscribe(products => {
      this.productsForDropdown = products;
    });
  } else {
    this.productsForDropdown = [];
  }
}
```

Slika 17 Metoda za pretragu proizvoda unutar Header komponente

Komponenta kategorija

U zaglavlju aplikacije se mogu pronaći i tri kategorije na koje se proizvodi mogu podijeliti: proizvodi za kosu, proizvodi za lice i proizvodi za tijelo. Struktura korisničkog sučelja svake kategorije slična je početnoj stranici aplikacije, a primjer *Skin* (proizvodi za lice) stranice prikazan je na slici 18. Iako se ovdje radi o tri stranice, za njih postoji samo jedna komponenta unutar Angular projekta, nazvana *CategoryComponent*. Osnovna ideja iza ovakvog pristupa je da se *CategoryComponent* komponenta konfigurira dinamički na temelju ulaznih parametara, u ovom slučaju kategorije proizvoda. Kada korisnik odabere, primjerice, kategoriju *Skin*, *CategoryComponent* dohvaća i prikazuje samo proizvode koji pripadaju toj kategoriji. Slično tome, odabirom *Hair* (proizvodi za kosu) ili *Body* (proizvodi za tijelo), komponenta će se prilagoditi i prikazati odgovarajuće proizvode.



Slika 18 Korisničko sučelje Category komponente

Ova funkcionalnost postignuta je kroz kombinaciju Angularovih ruta i servisa. U konstruktor (slika 19) je dodan *ActivatedRoute* servis koji omogućuje komponenti prepoznavanje proslijeđenog parametra rute (u ovom slučaju kategorija proizvoda), dok *ProductsService* omogućuje dohvaćanje proizvoda koji odgovaraju toj kategoriji. Funkcija *fetchProductsByCategory* je ključna funkcija koja dohvaća proizvode na temelju kategorije i ažurira prosječne ocjene svakog proizvoda.

```
constructor(private route: ActivatedRoute, private productsService: ProductsService,
  private ratingService: RatingService, private router: Router) { }

ngOnInit() {
  this.route.params.subscribe(params => {
    this.categoryName = params['categoryName'];
    this.fetchProductsByCategory(this.categoryName);
  });
}

fetchProductsByCategory(category: string | null) {
  if (category) {
    this.productsService.getProductsByCategory(category).subscribe(data => {
      this.products = data;
      this.products.forEach(product => {
        if (product.id) {
          this.ratingService.getAverageProductRating(product.id).subscribe(avgRating => {
            this.averageRatings[product.id] = avgRating;
          });
        }
      });
    });
  }
}
```

Slika 19 TypeScript kod Category komponente

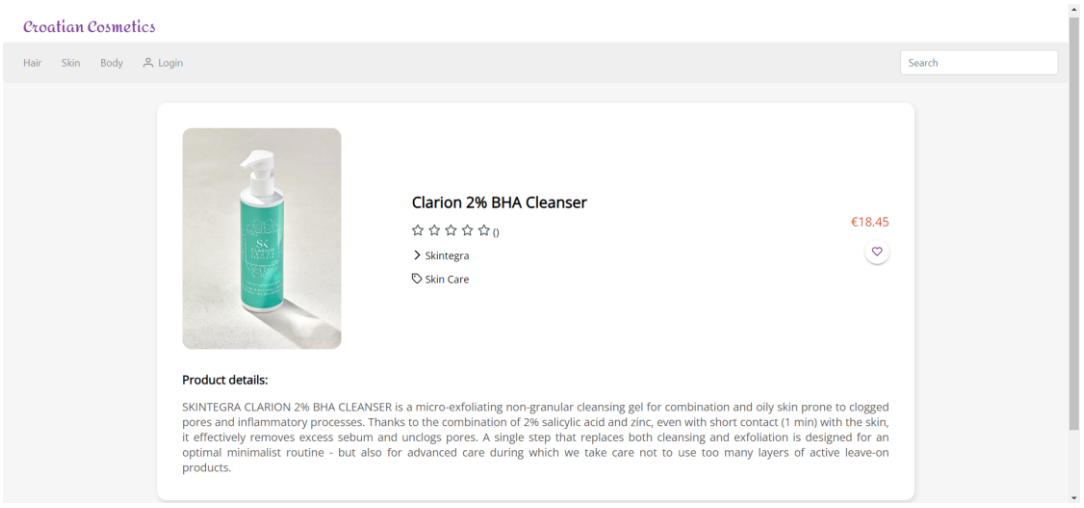
Komponenta detalja proizvoda

Komponenta detalja proizvoda, nazvana *ProductDetailsComponent*, pružajući korisnicima detaljnije informacije o odabranom proizvodu. Ova komponenta integracije različitih servisa i funkcionalnosti koje Angular okvir nudi. Klikom na bilo koji proizvod u aplikaciji prikazan na početnoj stranici ili na stranicama kategorija dovodi na stranicu detalja tog proizvoda. Također, odabirom proizvoda iz trake za pretraživanje također se dolazi na stranicu detalja proizvoda. Za ovu funkcionalnost navigacije i usmjeravanja zaslužna je ranije spomenuta ruta, *{path:'product/:id', component: ProductDetailsComponent},* definirana unutar *AppRoutingModule* modula. Identifikator proizvoda dohvaća se dinamički, slično kao i u *CategoryComponent* komponenti. Na slici 20 prikazana je *ngOnInit* metoda unutar koje se koristi *ActivatedRoute* servis za dohvaćanje parametra identifikatora proizvoda, *productId*, na kojeg je korisnik kliknuo. Kada je dohvaćen, *ProductsService* se koristi za dohvaćanje detaljnih informacija o proizvodu s tim identifikatorom. I na stranicama detalja proizvoda moguće je ocijeniti proizvod pomoću *RatingService* servisa, a u ovoj komponenti definirana je metoda *handleRatingChange* koja ažurira prosječnu ocjenu u stvarnom vremenu.

```
ngOnInit(): void {
  this.activatedRoute.params.subscribe((params) => {
    const productId = params['id'];
    if (productId) {
      this.productService.getProductById(productId).subscribe((product) => {
        this.product = product;
      });
    }
  });
}

handleRatingChange(productId: string, newRatingValue: number): void {
  console.log(`Product with ID: ${productId} received a new rating of ${newRatingValue}`);
  this.ratingService.getAverageProductRating(productId).subscribe(avgRating => {
    this.averageRatings[productId] = avgRating;
  });
}
```

Slika 20 TypeScript kod Product Details komponente

Korisničko sučelje stranice svakog pojedinačnog proizvoda razlikuje se od prikaza proizvoda na početnoj stranici i stranicama kategorija, a prikazano je na slici 21.



Slika 21 Korisničko sučelje Product Details komponente

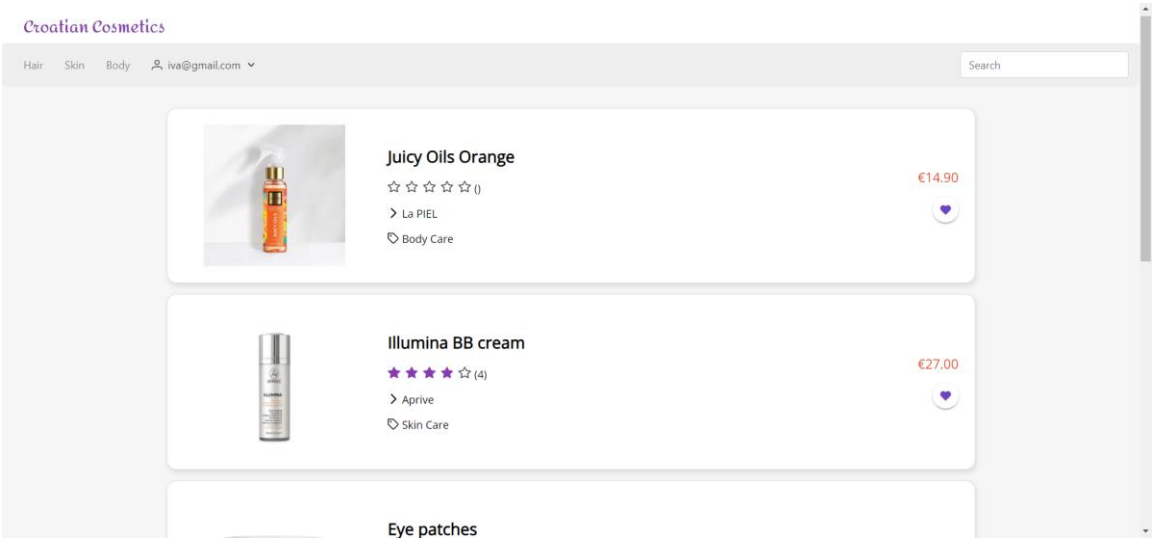
Komponenta liste želja

Svakom registriranom korisniku omogućeno je dodavanje proizvoda u vlastitu listu želja, za čiji prikaz je odgovorna *WishlistPageComponent* komponenta. Slika 22 prikazuje *ngOnInit* metodu koja pomoću *wishlistService* servisa dohvaća sve proizvode liste želja trenutno prijavljenog korisnika. Kao gotovo svaka komponenta do sada, i ova komponenta sadrži funkcije *viewProductDetails*, zaslužne za navigiranje korisnika do stranice detalja proizvoda, te *handleRatingChange* funkcije, zaslužne za ažuriranje prosječne ocjene proizvoda u stvarnom vremenu.

```
export class WishlistPageComponent implements OnInit {
  wishlistProducts: Products[] = [];
  averageRatings: { [key: string]: number | null } = {};

  constructor(private wishlistService: WishlistService, private ratingService: RatingService,
    private router: Router) { }
  ngOnInit() {
    this.wishlistService.getUserWishlistProducts().subscribe(products => {
      this.wishlistProducts = products;
    });
  }

  viewProductDetails(productId: string) {
    this.router.navigate(['/products', productId]);
  }

  handleRatingChange(productId: string, newRatingValue: number): void {
    console.log(`Product with ID: ${productId} received a new rating of ${newRatingValue}`);
    this.ratingService.getAverageProductRating(productId).subscribe(avgRating => {
      this.averageRatings[productId] = avgRating;
    });
  }
```

Slika 22 TypeScript kod Wishlist Page komponente

Primjer korisničkog sučelja stranice liste želja određenog korisnika prikazano je na sljedećoj slici. Ovo korisničko sučelje razlikuje se od dosad spomenutih sučelja, na način da su proizvodi prikazani vertikalno jedan ispod drugoga.



Slika 23 Korisničko sučelje Wishlist Page komponente

Komponente registracije i prijave korisnika

Za ocjenjivanje proizvoda i kreiranje vlastite liste želja korisnici se moraju registrirati te biti ulogirani u aplikaciju. Za te potrebe kreirane su dvije komponente: *RegistrationComponent* i *LoginComponent.* Za registraciju korisnika koristila se Firebase usluga autentifikacije. *RegistrationComponent* predstavlja ključnu točku za nove korisnike koji žele pristupiti dodatnim funkcionalnostima aplikacije. Kroz ovu komponentu, korisnici kreiraju svoj korisnički račun unoseći osnovne informacije kao što su e-mail adresa i lozinka. Za ovu funkcionalnost koriste se Angularove reaktivne forme koje se koriste za obrazac registracije i validaciju korisničkog unosa. One omogućavaju dinamičnu validaciju unosa i bolju interakciju s korisnikom. Na taj način korisnik će biti obaviješten ako unese neispravnu e-mail adresu ili ako lozinka ne zadovoljava određene kriterije.

Reaktivna forma *registerForm*, prikazana na slici 24, instanca je FormGroup. FormGroup je klasa reaktivnih formi koja prati vrijednost i status validnosti skupa FormControl instanci. Svaki FormControl predstavlja pojedinačno polje unutar obrasca. U ovom slučaju postoje dva kontrolna polja: *email* i *password*. *Email* kontrolno polje koristi *Validators.required* validator koji provjerava je li polje popunjeno, a *Validators.email* provjerava je li unesena ispravna e-mail adresa.

Nakon što korisnik unese potrebne informacije i pokrene postupak registracije, *AuthService* komunicira s Firebase autentifikacijskom uslugom kako bi se kreirao novi korisnički račun. U slučaju uspješne registracije, korisnik je automatski ulogiran i preusmjeren na početnu stranicu aplikacije. U suprotnom, korisnik će biti obaviješten o potencijalnim greškama.

```
export class RegistrationComponent {

  registerForm = this.fb.group({
    email: new FormControl('', [Validators.required, Validators.email]),
    password: new FormControl('', [Validators.required])
  })

  constructor(private authService: AuthService, private fb: FormBuilder, private router: Router) { }
  ngOnInit(): void {

  }

  registerWithEmail() {
    console.log(this.registerForm.value);

    const email = this.registerForm.value.email!;
    const password = this.registerForm.value.password!;

    this.authService.register({ email, password })
      .then((res: any) => {
        this.router.navigateByUrl('/');
      })
      .catch((error: any) => {
        console.error(error);
        alert('Something went wrong');
        this.router.navigate(['/login']);
      });
  }
}
```

Slika 24 TypeScript kod Registration komponente

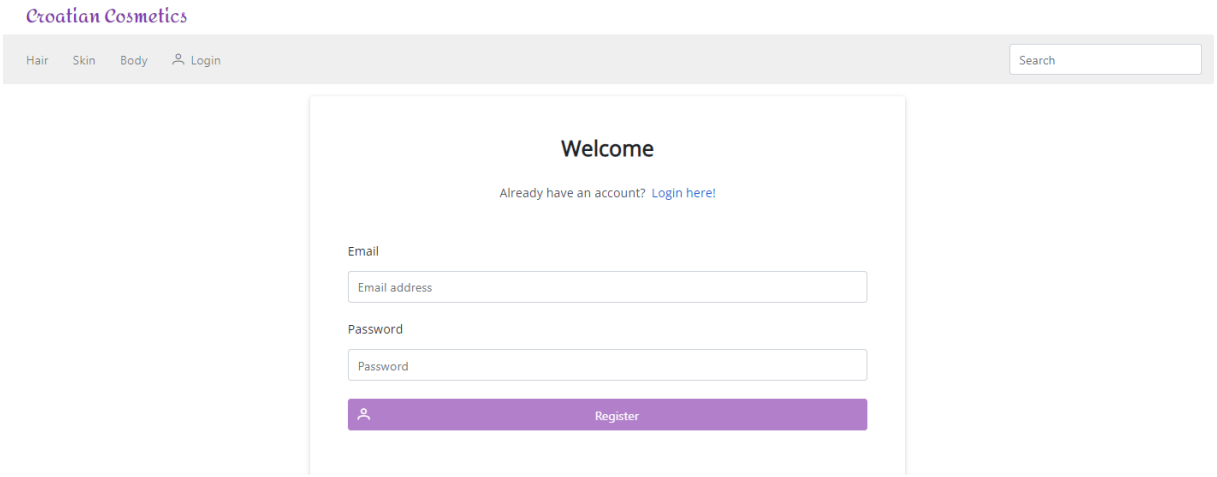
Unutar HTML dijela koda koristi se direktiva [formGroup]="loginForm" kako bi se vezala forma za reaktivni obrazac definiran u komponenti. To omogućava Angularu da prati stanje forme i automatski primijeni promjene korisničkih unosa. Element *<input>* (slika 25) ima atribut *formControlName*, koji se odnosi na ime *FormControl*-a u reaktivnom obrascu. Ovo omogućava Angularu da poveže ovaj input element s odgovarajućim poljem u reaktivnom obrascu. Ovdje se također koristi PrimeNg element *<small>* pomoću Angular direktive **ngIf*. Ovaj element prikazuje poruku o grešci (*Email is required*) ako unos email-a nije ispravan ili nije popunjen. *ngIf je koristan jer omogućava dinamičko upravljanje prikazom i skrivanje elemenata na temelju stanja aplikacije ili korisničkih interakcija. To često olakšava izgradnju korisničkog sučelja koje reagira na promjene u aplikaciji.

```
<div [formGroup]="registerForm" class="mb-5">
    <label for="email1" class="block text-900 font-medium mb-3">Email</label>
    <input id="email1" type="text" placeholder="Email address" pInputText class="w-full mb-4 py-2" formControlName="email">
    <small class="p-error block mb-3" *ngIf="registerForm.get('email')?.invalid && registerForm.get('email')?.dirty">
        Email is required</small>
```

Slika 25 HTML Registration komponente

S druge strane, LoginComponent omogućuje postojećim korisnicima da se prijave u aplikaciju koristeći svoje korisničke podatke. Proces prijave sličan je procesu registracije, ali se ovdje provjeravaju već postojeći korisnički podaci. Ako su uneseni podaci ispravni, korisnik će biti prijavljen i dobit će pristup svim funkcionalnostima aplikacije.

Izgled korisničkih sučelja stranice za prijavu korisnika te stranice za registraciju korisnika gotovo je isti, a razlikuje se jedino po tekstu koji ovisi o tome radi li se o registraciji ili prijavi. Na slici 26 prikazano je sučelje stranice za registraciju.



Slika 26 Korisničko sučelje Registration komponente

3.3.2. *Child* komponente

Dijete komponenta (eng. *child)* u Angularu predstavlja modularne, ponovno upotrebljive dijelove korisničkog sučelja koji su ugniježđeni unutar većih, "roditeljskih" komponenata. Omogućuju izgradnju aplikacije sastavljene od manjih, neovisnih jedinica koje se mogu lako testirati i ponovno koristiti.

Unutar ove aplikacije stvorene su dvije child komponente: *RatingComponent* i *WishlistButtonComponent*.

Komponenta ocjenjivanja

Komponenta *RatingComponent* omogućuje korisnicima ocjenjivanje proizvoda i prikazivanje prosječne ocjene proizvoda. U korisničkom sučelju obje funkcionalnosti prikazane su kroz zvjezdice koje reprezentiraju ocjenu. Za prikaz zvjezdica korišten je *<p-rating>* element vanjske biblioteke PrimeNg, prikazan na slici 27. Dodatno se pokraj zvjezdica u korisničkom sučelju prikazuje i decimalni broj prosječne ocjene pohranjen u varijabli *averageRating,* gdje je korištena cijev (eng. *pipe*) koja služi za formatiranje decimalnog broja na točno dvije decimale.

Nadalje, ovdje se koristi i dvosmjerna veza podataka angularovom direktivom *ngModel,* prikazanom na slici 27. Ovaj atribut omogućuje da se *averageRating* varijabla ažurira kada korisnik promijeni ocjenu putem komponente za ocjenjivanje. Istovremeno, ako se *averageRating* promijeni programski unutar komponente (nakon dohvaćanja nove prosječne ocjene iz baze podataka), komponenta za ocjenjivanje će se automatski ažurirati da prikaže tu novu vrijednost. Metoda *(onRate)="onRatingChanged($event)"* je događaj koji se pokreće kada korisnik ocijeni proizvod. Ova metoda dohvaća novu ocjenu i ažurira je u bazi podataka putem *RatingService* servisa. Atribut *[readonly]="!isAuthenticated"* osigurava da samo autentificirani korisnici mogu mijenjati ocjenu. Ako korisnik nije prijavljen, zvjezdice u korisničkom sučelju će biti dostupne samo za čitanje.

```
<div class="star">
    <p-rating [(ngModel)]="averageRating" [cancel]="false"
              (onRate)="onRatingChanged($event)" [readonly]="!isAuthenticated"></p-rating>
    <span class="rating-decimal" style="padding-left: 5px;">({{averageRating | number:'1.0-2'}})</span>
</div>
```

Slika 27 HTML Rating komponente

Komunikacija između roditeljske i dječje komponente u Angularu postiže se kroz mehanizme @Input() i @Output(). Na slici 28 prikazana je *@Input()* dekoracija koja omoguće RatingComponent da primi vrijednost identifikatora proizvoda. Komponenta koristi *fetchAverageRating* metodu iz *RatingService* servisa za dohvaćanje prosječne ocjene svakog proizvoda temeljem ranije spomenutog identifikatora.

Komponenta također omogućuje autentificiranim korisnicima ocjenjivanje proizvoda. Autentifikacijski status korisnika provjerava se putem *AuthService* servisa i pohranjuje se u varijabli *isAuthenticated*. Ako je korisnik autentificiran i promijeni svoju ocjenu, poziva se *onRatingChanged* metoda. Ova metoda ažurira lokalnu varijablu *userRating* s novom ocjenom te koristi *RatingService* za ažuriranje ocjene u bazi podataka.

```typescript
export class RatingComponent implements OnInit {
  @Input() productId!: string;

  averageRating: number | null = null;
  userRating: number | null = null;
  isAuthenticated: boolean = false;

  constructor(private ratingService: RatingService, private authService: AuthService) { }

  ngOnInit(): void {
    this.fetchAverageRating();

    this.authService.getAuthState().subscribe(user => {
      this.isAuthenticated = !!user;
    });
  }
  fetchAverageRating(): void {
    this.ratingService.getAverageProductRating(this.productId).subscribe(average => {
      this.averageRating = average;
    }, error => {
      console.error("Error fetching average rating:", error);
    });
  }

  onRatingChanged(event: any): void {
    const newRating = event.value;
    this.userRating = newRating;

    this.authService.getAuthState().subscribe(user => {
      if (user) {
        this.ratingService.rateProduct(this.productId, newRating);
        alert('Thank you for rating this!');
      } else {
      }
    });
  }
```

Slika 28 TypeScript kod Rating komponente

Komponenta gumba za dodavanje i brisanje proizvoda iz liste želja

Za funkcionalnost dodavanja i brisanja proizvoda iz korisnikove liste želja kreirana je posebna komponenta nazvana *WishlistButtonComponent*. Kreirana je kao zasebna komponenta kako bi bila lako integrirana u sve ostale komponente koje imaju prikaz proizvoda, kako bi korisnik sa bilo koje stranice mogao dodavati i brisati proizvode iz svoje liste želja.

U ovoj komponenti također je korištena *@Input()* dekoracija koja joj omogućuje dohvaćanje identifikatora proizvoda, kao što je bio slučaj u *RatingComponent.* Pri učitavanju ove komponente, *ngOnInit* (slika 29) poziva *checkIfProductIsInWishlist* metodu koja provjerava proizvode koji su dodani u korisnikovu listu želja, ukoliko oni postoje. Ova metoda koristi *AngularFirestore* za dohvaćanje korisnikovih podataka i provjeru postojanja proizvoda unutar njegove liste želja. Metoda *addToWishlist* predstavlja ključ korisnikove interakcije unutar ove komponente. Kada korisnik klikne na odgovarajući gumb vezan uz proizvod, poziva se ova metoda. Logika unutar *addToWishlist* metode koristi *WishlistService* servis za upravljanje stvarnim dodavanjem ili uklanjanjem proizvoda iz liste želja. U njoj

je također integracija *authService* servisa koji osigurava da se ova akcija može izvršiti samo ako je korisnik prijavljen.

```
ngOnInit(): void {
  this.checkIfProductIsInWishlist();
}

addToWishlist(product: Products) {
  this.authService.getAuthState().subscribe((user) => {
    if (user) {
      this.wishlistService.addToWishlist(product).subscribe(isAdded => {
        this.isFavorite = isAdded;
      });
    } else {
      alert("Please log in to add to wishlist");
    }
  });
}

private checkIfProductIsInWishlist(): void {
  if (this.product) {
    this.authService.getAuthState().subscribe((user) => {
      if (user) {
        const userId = user.uid;
        const userDocRef = this.db.collection('users').doc(userId);
        userDocRef.get().subscribe((doc) => {
          if (doc.exists) {
            const data = doc.data() as { wishlist?: string[] };
            const currentWishlist = data.wishlist || [];
            this.isFavorite = currentWishlist.includes(this.product!.id);
          }
        });
      }
    });
```

Slika 29 TypeScript kod Wishlist Button komponente

U vizualnom predstavljanju, HTML predložak koristi *p-button* (slika 30) oznaku iz PrimeNg biblioteke koja pruža funkcionalan gumb u obliku srca prikladan za ovu komponentu. U ovisnosti o tome nalazi li se proizvod u korisnikovoj listi želja ili ne, ikona gumba mijenja se između ispunjenog srca (*pi pi-heart-fill)* i praznog srca (*pi pi-heart*).

```
<p-button
  [icon]="isFavorite ? 'pi pi-heart-fill' : 'pi pi-heart'"
  styleClass="p-button-rounded p-button-help p-button-text p-button-raised"
  *ngIf="product"
  (click)="addToWishlist(product)"
  [disabled]="isDisabled">
</p-button>
```

Slika 30 HTML Wishlist Button komponente

3.3.3. Servisi

Servis za autentifikaciju

Servis AuthService služi za upravljanje autentifikacijom korisnika koristeći Firebase Authentification uslugu. Ovaj servis koristi nekoliko zavisnosti, uključujući AngularFireAuth za autentifikaciju korisnika putem Firebase-a, Router za upravljanje navigacijom unutar aplikacije nakon uspješne autentifikacije, FirestoreService za interakciju s Firebase bazom podataka te AngularFirestore za direktan pristup Firestore funkcionalnostima.

Na slici 31 prikazane su *login* i *register* metode. Login metoda omogućuje korisnicima da se prijave koristeći svoj e-mail i lozinku. Nakon uspješne prijave, *token* se pohranjuje u lokalno skladište kako bi se korisnikova sesija održavala i nakon osvježavanja stranice. Register metoda omogućuje registraciju novih korisnika. Nakon uspješne registracije, korisnikovi podaci se dodaju u Firestore kolekciju podataka *users* kao novi dokument. Metoda koja sadrži logiku za spremanje korisnikovih podataka u bazu, *addUserToFirestore*, definirana je u firestoreService servisu, a poziva se u ovoj metodi za registraciju. Također, za svakog registriranog korisnika se u tom istom dokumentu koji sadrži korisnikove podatke kreira i prazna lista *wishlist* koja služi za pohranjivanje liste želja proizvoda povezanog sa svakim pojedinim korisnikom. Da bi ovo bilo ostvareno, kreirana je posebna *createEmptyWishlist* metoda unutar ove komponente koja se poziva u *register* metodi kako bi registracija funkcionirala na opisani način. Ova komponenta sadrži još nekoliko metoda kao što su metoda za odjavu korisnika, dohvaćanje trenutno prijavljenog korisnika i provjeru autentifikacije.

```
login(user: { email: string, password: string }): Promise<void> {
  return this.fireauth.signInWithEmailAndPassword(user.email, user.password)
    .then(() => {
      localStorage.setItem('token', 'true');
    });
}

register(user: { email: string; password: string }): Promise<void> {
  return this.fireauth
    .createUserWithEmailAndPassword(user.email, user.password)
    .then((userCredential) => {
      if (userCredential.user) {
        alert('Registration successful');
        const user = userCredential.user;
        const userData = {
          email: user.email
        };

        this.firestoreService.addUserToFirestore(user.uid, userData);
        this.createEmptyWishlist(user.uid);
      } else {
        console.error('User is null after registration');
      }
    })
    .catch((error) => {
      console.error('Error during registration: ', error);
    });
}
```

Slika 31 AuthService

Firestore servis

Firestore servis omogućuje interakciju s Firestore bazom podataka, specifično za upravljanje korisničkim podacima. Sadrži metodu *addUserToFirestore* (slika 32) koja omogućava dodavanje korisnika u Firestore bazu podataka. Ona prima jedinstveni identifikator korisnika (*UID*) i njegove podatke, odnosno *email*, te ih pohranjuje unutar kolekcije *users* u Firestore bazi podataka.

FirestoreService usko se povezuje s *AuthService* servisom. Dok AuthService upravlja procesom autentifikacije korisnika, FirestoreService pohranjuje i upravljanja korisničkim podacima nakon što je autentifikacija uspješno provedena.

```
addUserToFirestore(uid: string, userData: any) {
  const { email } = userData;

  return this.firestore.collection('users').doc(uid).set({
    email
  });
}
```

Slika 32 FirestoreService

Servis proizvoda

Servis *ProductsService* služi za implementaciju raznih logika za dohvaćanje proizvoda iz baze podataka. Sadrži metode koje se koriste za dohvaćanje proizvoda po različitim kriterijima potrebne za prikaz proizvoda unutar početne stranice, stranice detalja proizvoda, stranice kategorija, stranice liste želja te dohvat proizvoda unutar trake za pretraživanje unutar zaglavlja stranice.

Jedna od metoda koje sadrži ovaj servis je *getProducts* koja dohvaća sve proizvode iz baze podataka, a koristi se za prikaz proizvoda na početnoj stranici. U servisu su implementirane metode *getProductById*, *getProductsByCategory* te *searchProductsByName*. Metoda getProductById omogućuje dohvaćanje specifičnog proizvoda temeljem njegovog jedinstvenog identifikatora, a potrebna je za prikaz proizvoda na stranicama detalja proizvoda te za prikaz proizvoda u listama želja.

Metoda getProductsByCategory dohvaća kategoriju svakog proizvoda iz baze te na osnovu toga filtrira i vraća listu proizvoda. Na taj način se odgovarajući proizvodi prikazuju na stranicama proizvoda za lice, tijelo i kosu.

Zadnja metoda u ovom servisu je *searchProductsByName* metoda koja omogućuje korisnicima pretraživanje proizvoda na temelju njihovog naziva. Na osnovu korisnikovog unosa unutar trake za pretraživanje, ova metoda vraća mu popis proizvoda koji odgovaraju njegovom unosu.

Servis ocjena proizvoda (rating service)

Servis RatingService sadrži logiku za spremanje novih recenzija u bazu podataka, dohvaćanje svih ocjena svakog pojedinog proizvoda te računanje prosječne ocjene za svaki proizvod.

Metoda *rateProduct* služi za dodavanje ili ažuriranje korisnikovih ocjena za određeni proizvod. Ova metoda prima identifikator proizvoda i vrijednost ocjene kao parametre. Nakon što korisnik ocijeni proizvod, metoda dohvaća trenutno prijavljenog korisnika pomoću AuthService servisa i stvara objekt ocjene koji sadrži identifikator proizvoda, vrijednost ocjene i vremensku oznaku. Zatim se ovaj objekt dodaje u *ratings* kolekciju Firestore baze podataka.

Metoda *getProductRating*, prikazana na slici 33, omogućuje dohvaćanje svih ocjena za određeni proizvod iz *ratings* kolekcije baze podataka. Metoda *getAverageRating* služi za izračunavanje prosječne ocjene za određeni proizvod koja pomoću prethodne metode dohvaća jedinstveni identifikator proizvoda. Obje metode koriste *Observable* biblioteke RxJS za asinkrono dohvaćanje i obrađivanje podataka o ocjenama, pružajući ažurirane informacije u stvarnom vremenu.

```
getProductRating(productId: string): Observable<any[]> {
  return this.firestore.collection('ratings', ref => ref.where('product', '==', productId))
  .valueChanges()
}

getAverageProductRating(productId: string): Observable<number | null> {
  return this.getProductRating(productId).pipe(
    map(ratings => {
      if (ratings && ratings.length > 0) {
        const totalRatings = ratings.reduce((acc, rating) => acc + rating.rating, 0);
        const average = totalRatings / ratings.length;
        return average;
      }
      return null;
    })
  );
}
```

Slika 33 Metode unutar Rating servisa

Servis za liste želja

Servis WishlistService implementiran je kako bi upravljao funkcionalnostima liste želja unutar aplikacije. Sadrži metodu za dodavanje i brisanje proizvoda iz liste želja svakog pojedinog korisnika te metodu za dohvat proizvoda pohranjenih u korisnikovoj listi želja.

Na slici 34 prikazana je *getUserWishlistProducts* metoda. Ona prvo dohvaća trenutno prijavljenog korisnika preko *getCurrentUser* metode iz authService servisa. Korištenjem *switchMap* operatora, metoda obrađuje dobivene informacije o korisniku: ako je korisnik autentificiran, metoda dohvaća korisnikove podatke iz Firestore kolekcije *users* koristeći korisnikov jedinstveni identifikator (*user.uid*). Ako korisnik nije autentificiran, vraća se *null*. Svaki dokument unutar kolekcije *users* sadrži korisnikov ID i pripadnu listu želja (wishlist). Sljedeći korak je dohvatiti elemente te liste, koji predstavljaju ID-ove proizvoda. Nakon što su ID-jevi proizvoda dohvaćeni, koristi se metoda *getProductById* iz *productService* servisa kako bi se dohvatili detalji svakog proizvoda. Na kraju, operator *combineLatest* kombinira sve dobivene informacije o proizvodima u jedan niz.

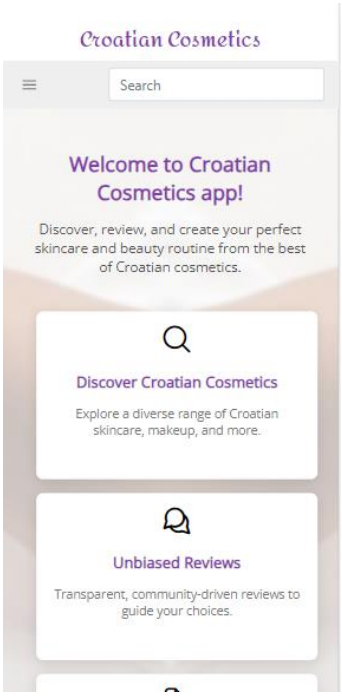
```
getUserWishlistProducts(): Observable<Products[]> {
  return this.authService.getCurrentUser().pipe(
    switchMap(user => {
      if (user) {
        return this.firestore.collection('users').doc(user.uid).valueChanges();
      } else {
        return of(null);
      }
    }),
    switchMap((userData: any) => {
      if (userData?.wishlist) {
        const productIds = userData.wishlist;

        const productObservables = productIds.map((productId: string) => {
          return this.productService.getProductById(productId);
        });

        return combineLatest(productObservables) as Observable<Products[]>;
      } else {
        // Return an empty array if no wishlist or not authenticated
        return of([]);
      }
    })
  );
}
```

Slika 34 Metod unutar Wishlist servisa

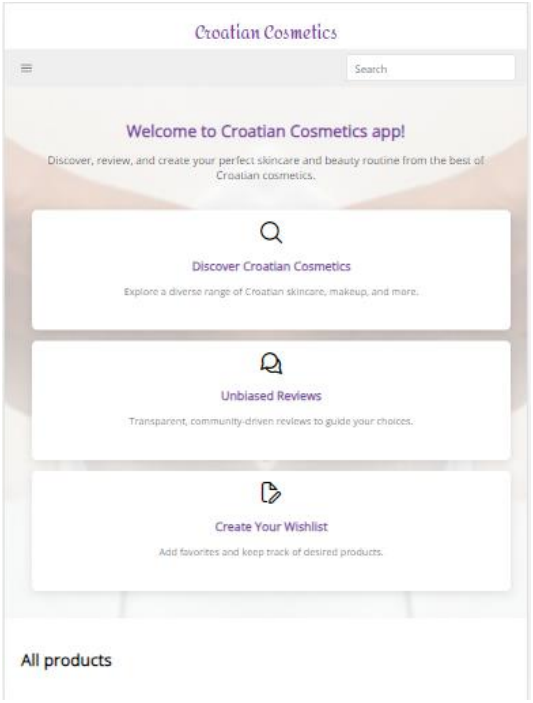
3.4. Responzivnost

Korisnici pristupaju web sadržaju s različitih uređaja, uključujući stolna računala, prijenosna računala, tablete i pametne telefone. Stoga je važno da web aplikacije budu dizajnirane tako da pružaju optimalno korisničko iskustvo na svim tim uređajima. Responzivni dizajn omogućava web aplikacijama da se automatski prilagode veličini zaslona uređaja na kojem se prikazuju, osiguravajući tako konzistentnost i funkcionalnost na svim platformama. „Croatian Cosmetics“ aplikacija je razvijena s naglaskom na responzivnost. Bez obzira na uređaj s kojeg korisnik pristupa, aplikacija će se prilagoditi kako bi pružila što bolje moguće korisničko iskustvo. Responzivnost je postignuta uređivanjem HTML elemenata SCSS stilovima korištenjem CSS medija upite *@media screen and (max-width: 768px)*, gdje max-width označava prijelomnu točku s vrijednošću koja u ovom slučaju odgovara tabletu. Dodatno, za različite elemente aplikacije korištene su PrimeNg komponente koje su same po sebi responzivne, čime je osigurana dodatna prilagodljivost i fluidnost dizajna bez potrebe za dodatnim prilagodbama. Primjerice, korištena je PrimeNg komponenta *p-menubar* za zaglavlje aplikacije, koja nudi gotovo rješenje pretvaranja izbornika u tzv. *hamburger* izgled kada se aplikacija pregledava na manjim uređajima.

U nastavku je slika 35 koja prikazuje početnu stranicu aplikacije na tabletu, te slika 36 koja prikazuje istu tu stranicu na mobilnom uređaju.



Slika 36 Početna stranica aplikacije na mobilnom uređaju



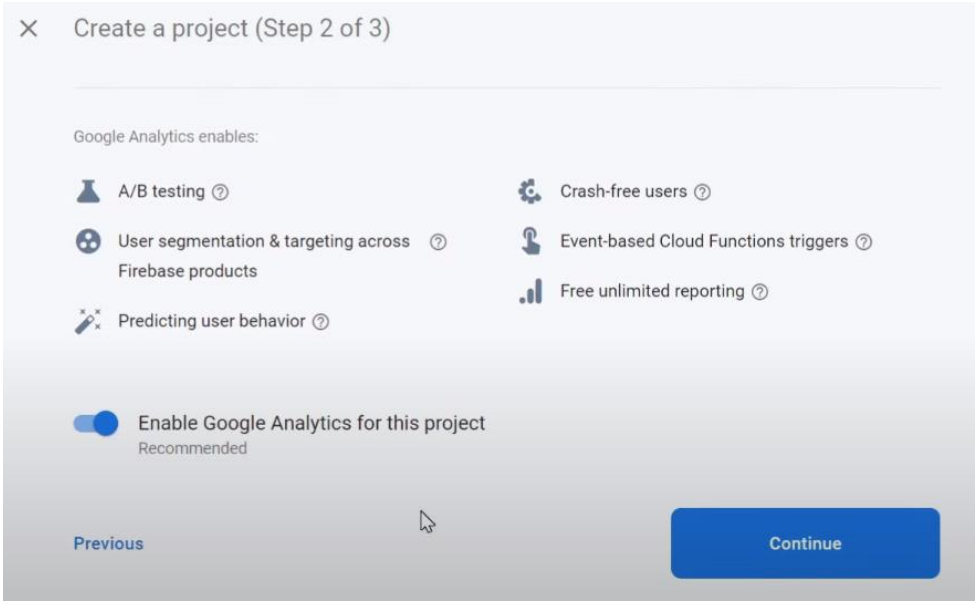
Slika 35 Početna stranica aplikacije na tabletu

3.5. Integracija aplikacije sa Firebaseom

Kako bi se omogućila dinamička interakcija s podacima, potrebno je integrirati aplikaciju s Firebaseovim uslugama.

3.5.1. Izrada Firebase projekta i instalacija unutar Angular projekta

Prvi korak integracije Firebasea s aplikacijom je izraditi Firebase projekt.



Slika 37 Web sučelje izrade Firebase projekta

Nakon što je napravljen firebase projekt unutar Firebase web sučelja, sljedeći korak je instalirati Firebase u aplikaciju pomoću komandne linije unutar projekta. Pokretanjem komandne linije i korištenjem naredbe *ng add @angular/fire*, u projekt je dodana Firebase podrška. Ova naredba automatski vodi korisnika kroz proces, nudeći opcije za odabir specifičnih Firebase servisa koji će biti implementirani. Paket *@angular/fire* pruža set servisa i alata za rad s Firebaseom na Angular platformi.

Nakon uspješne instalacije, konfiguracijski podaci Firebase projekta su uneseni u *environment.ts* datoteku (slika 38). Ova konfiguracija omogućava Angular aplikaciji da komunicira sa Firebase-om i pristupi određenim servisima.

Slika 38 environment.ts datoteka

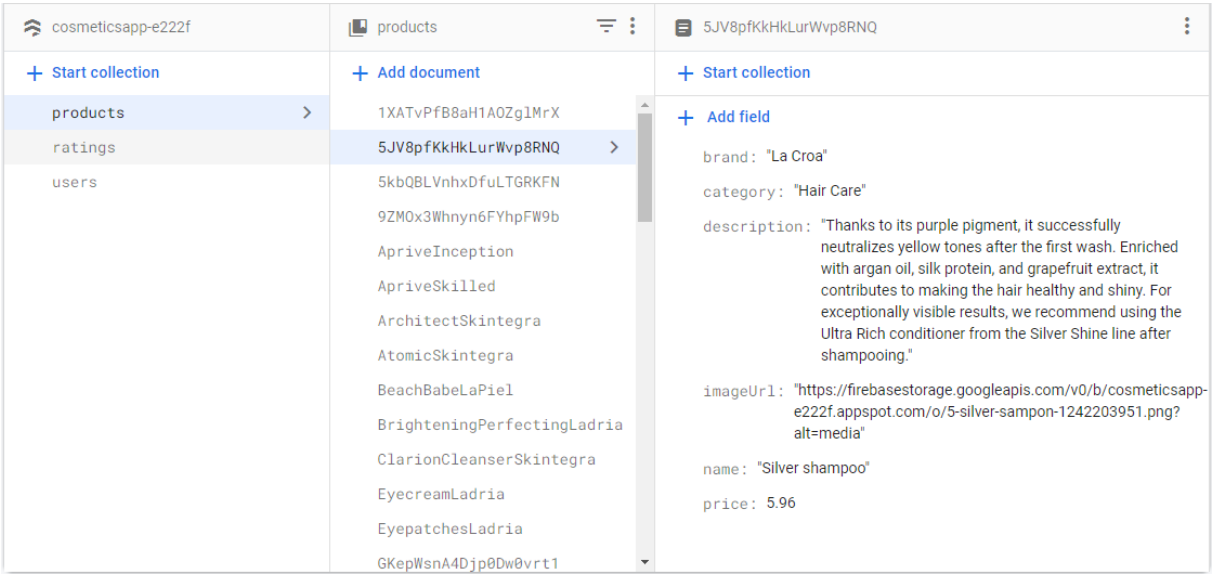
Unutar app.module.ts datoteke prikazane na slici 39, instalacijom su se uvezli Firebase moduli kao što su *AngularFireModule* i *AngularFireAuthModule*. Osim toga, pruženi su servisi kroz specijalne funkcije poput *provideFirebaseApp*, *provideAnalytics*, *provideAuth* i *provideFirestore*, omogućavajući aplikaciji pristup Firebase resursima.

Slika 39 app.module.ts datoteka nakon instalacije Firebasea

3.5.2. Firebase Firestore kolekcije

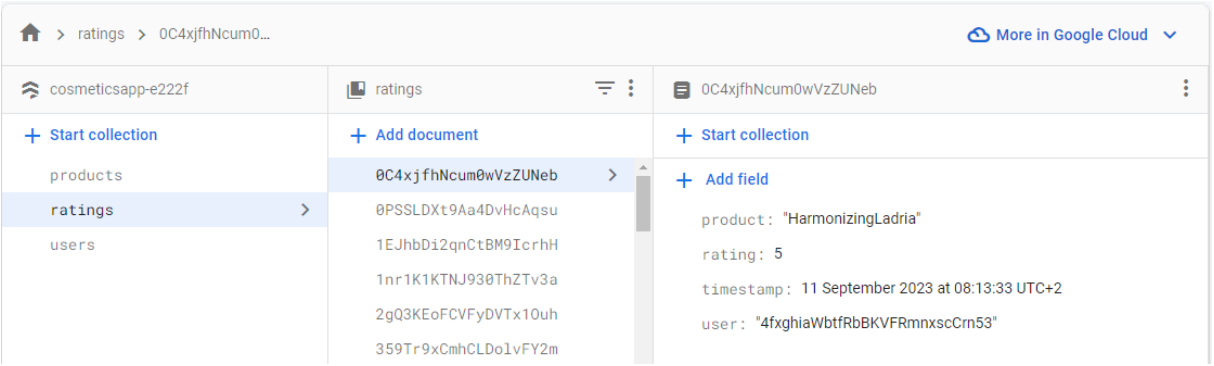
U kontekstu aplikacije „Croatian Cosmetics“ stvorene su tri kolekcije dokumenata unutar Firebase Firestore NoSQL baze podataka.

Prva kolekcija je *products* (slika 40), koja sadrži niz dokumenata. Svaki dokument predstavlja jedan proizvod unutar kojeg su njegovi detalji. Svaki proizvod sadrži informacije o brendu, kategoriji, nazivu, opisu, cijeni te putanji slike koja se dohvaća iz Firebase Storage baze.



Slika 40 products kolekcija unutar Firestore baze

Kolekcija ratings, prikazana na slici 41, služi za pohranjivanje korisničkih ocjena proizvoda u bazu na ranije opisan način. Svaki dokument predstavlja jednu ocjenu, a u njemu se pohranjuje jedinstveni identifikator proizvoda kako bi se znalo na koji proizvod se odnosi, jedinstveni identifikator korisnika koji je ocijenio proizvod, sama ocjena te vrijeme kada je proizvod ocjenjen.

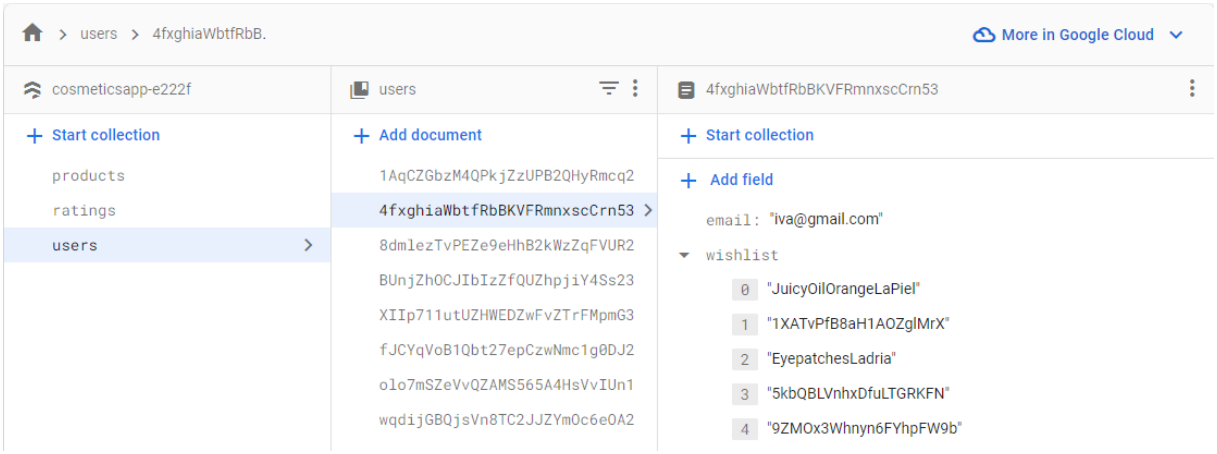


Slika 41 ratings kolekcija unutar Firestore baze

Registracijom korisnika u aplikaciji „Croatian Cosmetics“, njegovi osnovni podaci, poput e-maila i lozinke, automatski se pohranjuju u Firebase Authentication sustav. Ovaj sustav pruža osnovne funkcionalnosti autentifikacije, poput prijave, odjave i provjere autentifikacije, ali ne nudi složene funkcionalnosti za pohranu dodatnih korisničkih podataka. Stoga je kreirana kolekcija users unutar

Firestore baze podataka. Ova kolekcija služi kao most između Firebase Authentication sustava i aplikacije, omogućavajući pohranu dodatnih informacija o korisnicima, u ovom slučaju liste želja.

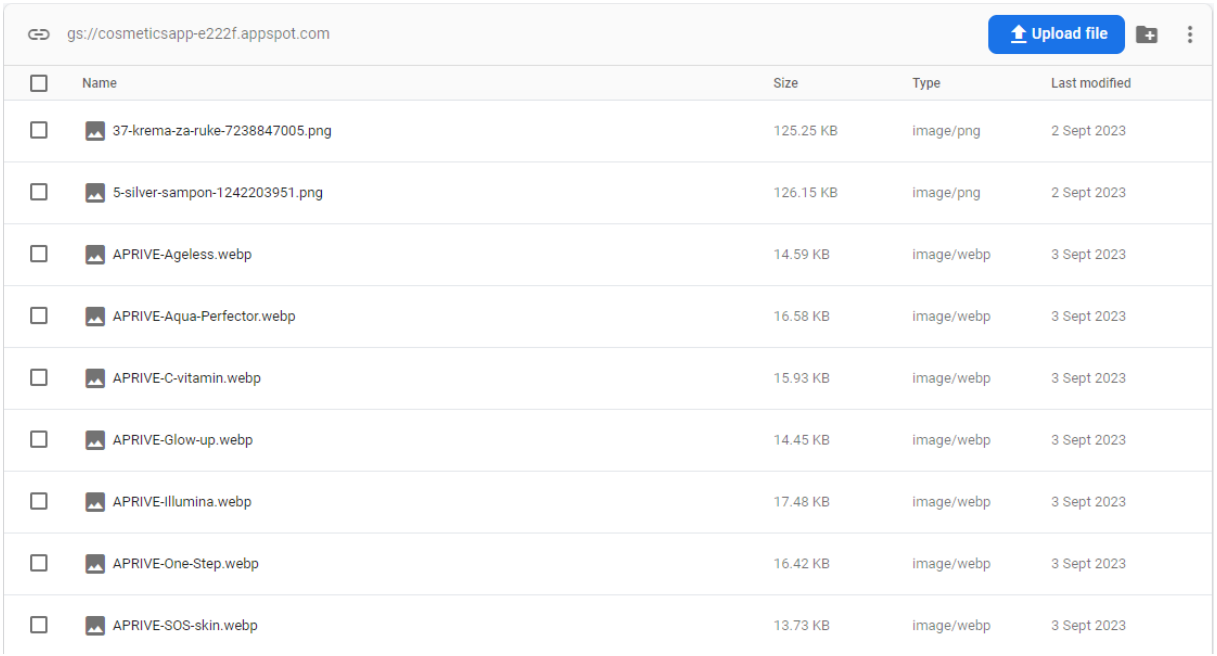
Kada se korisnik registrira, osim što se njegovi osnovni podaci pohranjuju u Firebase Authentication, stvara se i novi dokument unutar users kolekcije. U ovom dokumentu pohranjuje se e-mail adresa korisnika i inicijalizira se prazna lista želja (slika 42). Kako korisnik dodaje proizvode u svoju listu želja, ti proizvodi se identificiraju putem njihovih jedinstvenih identifikatora i dodaju se u korisnikovu listu želja. Važno je napomenuti da je ovaj jedinstveni identifikator proizvoda zapravo referenca na odgovarajući dokument unutar products kolekcije u Firestore bazi podataka.



Slika 42 users kolekcija unutar Firestore baze

3.5.3. Firebase Storage

Firebase Storage koristi se unutar „Croatian Cosmetics“ aplikacije za upravljanje slikama proizvoda. Firebase Storage osigurava brzo i učinkovito posluživanje slikama klijentima, bez obzira na njihovu geografsku lokaciju. Kada se slika proizvoda pohrani u Firebase storage, generira se jedinstveni URL koji se zatim koristi za dohvaćanje i prikaz slike unutar same aplikacije. Ovi URL-ovi implementirani su u dokumente *products* kolekcije unutar Firebase Firestore baze podataka kako bi se svaki proizvod povezao s njegovom pripadajućom slikom. Na slici 43 prikazan je Firebase Storage baza slika proizvoda potrebnih za aplikaciju „Croatian Cosmetics“.



Slika 43 Firebase Storage

3.5.4. Firebase Hosting

Po završetku implementacijske faze razvoja aplikacije, zadnji korak je njeno postavljanje na internet kako bi bila dostupna korisnicima. Za to je korištena Firebase Hosting usluga koja omogućava brzo i jednostavno postavljanje web aplikacija u oblak.

Prvi korak postavljanja aplikacije je izvršiti njeno *buildanje*. To se postiže korištenjem Angular CLI alata naredbom *npm run build* kako je prikazano na sljedećoj slici. Ova naredba generira *dist* datoteku koja sadrži optimiziranu verziju aplikacije spremnu za produkciju. Svi izvorni kodovi, skripte, stilovi i slike su minimizirani, grupirani i kompilirani kako bi se osiguralo brže učitavanje i bolje performanse aplikacije u produkcijskom okruženju.

```
PS C:\Users\iva.magdic\Desktop\Osobno\faks\diplomski\CroatianCosmetics\cosmeticsapp> npm run build

> cosmeticsapp@0.0.0 build
> ng build

✓ Browser application bundle generation complete.
✓ Copying assets complete.
✓ Index html generation complete.

Initial Chunk Files           | Names         |  Raw Size | Estimated Transfer Size
main.332affbd62b514c5.js      | main          |   1.31 MB |               309.39 kB
styles.6dfbc9c20ddbfead.css   | styles        | 504.65 kB |                33.81 kB
polyfills.03c0101da1f85d90.js | polyfills     |  33.02 kB |                10.65 kB
runtime.7f901140d27341b9.js   | runtime       |   1.39 kB |               748 bytes

                              | Initial Total |   1.84 MB |               354.58 kB

Build at: 2023-09-10T16:26:13.630Z - Hash: 4480d2872a811e7e - Time: 42161ms
PS C:\Users\iva.magdic\Desktop\Osobno\faks\diplomski\CroatianCosmetics\cosmeticsapp>
```

Slika 44 Buildanje aplikacije

Nakon što je *dist* datoteka generirana, slijedeći korak je postavljanje ove datoteke na Firebase Hosting. Za to je potrebno koristiti Firebase CLI alat. Firebase CLI alat potrebno je instalirati naredbom *npm install -g firebase-tools* (slika 45).

```
PS C:\Users\iva.magdic\Desktop\Osobno\faks\diplomski\CroatianCosmetics\cosmeticsapp> npm install -g firebase-tools
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain circumstances, which
 is known to be problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142

added 1 package, changed 673 packages, and audited 675 packages in 2m

62 packages are looking for funding
  run `npm fund` for details

2 moderate severity vulnerabilities

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

Slika 45 Instalacija Firebase CLI

Nakon instalacije Firebase CLI-a, potrebno je autentificirati se pomoću naredbe *firebase login*, no u ovom slučaju to je već ranije učinjeno. Također, korištenjem naredbe *firebase init* bilo je potrebno odabrati Hosting kao uslugu i povezati ga s prethodno kreiranim Firebase projektom.

Kada je sve konfigurirano, posljednji korak je postavljanje *dist* datoteke na Firebase Hosting pomoću naredbe *firebase deploy*. Nakon uspješnog postavljanja, Firebase će pružiti URL putem kojeg je aplikacija dostupna na internetu, kao što se može vidjeti na slici 46.

```
PS C:\Users\iva.magdic\Desktop\Osobno\faks\diplomski\CroatianCosmetics\cosmeticsapp> firebase deploy

   Thank you for trying our early preview of Angular support on Firebase Hosting.
   During the preview, support is best-effort and breaking changes can be expected. Proceed with caution.

   Documentation: https://firebase.google.com/docs/hosting/frameworks/angular
   File a bug: https://github.com/firebase/firebase-tools/issues/new?template=bug_report.md
   Submit a feature request: https://github.com/firebase/firebase-tools/issues/new?template=feature_request.md

   We'd love to learn from you. Express your interest in helping us shape the future of Firebase Hosting: https://goo.gle/41enW5X

✓ Browser application bundle generation complete.
✓ Copying assets complete.
✓ Index html generation complete.

Initial Chunk Files           | Names         |  Raw Size | Estimated Transfer Size
main.6f54b0f6df932eca.js      | main          |   1.35 MB |               314.21 kB
styles.6dfbc9c20ddbfead.css   | styles        | 504.65 kB |                33.81 kB
polyfills.03c0101da1f85d90.js | polyfills     |  33.02 kB |                10.65 kB
runtime.7f901140d27341b9.js   | runtime       |   1.39 kB |               748 bytes

                              | Initial Total |   1.87 MB |               359.40 kB

Build at: 2023-09-10T18:12:49.734Z - Hash: 1d7ed52e25249c54 - Time: 13898ms

=== Deploying to 'cosmeticsapp-e222f'...

i  deploying hosting
i  hosting[cosmeticsapp-e222f]: beginning deploy...
i  hosting[cosmeticsapp-e222f]: found 71 files in .firebase\cosmeticsapp-e222f\hosting
+  hosting[cosmeticsapp-e222f]: file upload complete
i  hosting[cosmeticsapp-e222f]: finalizing version...
+  hosting[cosmeticsapp-e222f]: version finalized
i  hosting[cosmeticsapp-e222f]: releasing new version...
+  hosting[cosmeticsapp-e222f]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/cosmeticsapp-e222f/overview
Hosting URL: https://cosmeticsapp-e222f.web.app
```

Slika 46 firebase deploy

Zaključak

U digitalnom dobu, web aplikacije postale su ključna sredstva za pružanje informacija i usluga širokoj publici. Ovaj rad pruža sveobuhvatan uvid u proces razvoja web aplikacije na primjeru „Croatian Cosmetics“ aplikacije, platforme posvećene hrvatskim kozmetičkim brendovima. Zahvaljujući rastu softvera otvorenog koda, mnoge od tehnologija potrebnih za razvoj web aplikacija postale su dostupne svima.

Ovaj rad pružio je duboki uvid u sve faze razvoja web aplikacije, od planiranja i dizajna, preko frontend i backend implementacije, sve do postavljanja aplikacije na web. Čitatelj može steći jasno razumijevanje kompleksnosti i detalja koji ulaze u izradu suvremene web aplikacije.

Kroz rad je istaknuta i važnost web dizajna i korisničkog iskustva u kreiranju aplikacija koje su ne samo funkcionalne, već i intuitivne i estetski privlačne. Korištenjem suvremenih razvojnih alata i tehnologija, poput Angulara i Firebasea, demonstrirano je kako se može postići visoka razina funkcionalnosti i korisničke interakcije. Osim korisničkog iskustva, važan je i izgled korisničkog sučelja koji ostavlja prvi dojam na korisnika. Prvi dojam često određuje hoće li korisnik nastaviti s interakcijom s aplikacijom ili će je napustiti. Stoga je na praktičnom primjeru ovog rada posvećena pažnja kreiranju vizualno privlačnog dizajna koji je prilagođen svim uređajima, od desktop računala do mobilnih uređaja.

Izrada web aplikacije složen je proces koji zahtjeva vrijeme i mnogo rada. Uz to, uvijek ima mjesta za napredak pa je tako plan budućeg rada na ovoj aplikaciji proširiti funkcionalnosti davanjem korisnicima mogućnost ostavljanja komentara na proizvode.

Razvoj web aplikacije predstavlja složen proces koji zahtjeva predanost, trud i vrijeme. Unatoč postignutom, svaka aplikacija nudi prostor za inovacije i poboljšanja. To se odnosi i na ovu aplikaciju koja nudi prostor za nadogradnju funkcionalnosti u budućnosti, poput omogućavanja korisnicima ostavljanje komentara na proizvode.

Literatura

[1] N. Hoić-Božić, »2. poglavlje: World Wide Web, Skripta za predmet "Multimedijski sustavi",« [Mrežno]. Dostupno na: https://moodle.srce.hr/2019-2020/pluginfile.php/3374535/mod_resource/content/1/2_WWW_2019_2020.pdf. [Pokušaj pristupa 29. 8. 2023.].

[2] Wikipedija, »World Wide Web Consortium,« [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/World_Wide_Web_Consortium. [Pokušaj pristupa 1. 9. 2023.].

[3] Medium, »Single-page application vs. multiple-page application,« [Mrežno]. Dostupno na: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. [Pokušaj pristupa 1. 9. 2023.].

[4] codingninjas, »Single Page Apps vs. Multi-Page Apps,« [Mrežno]. Dostupno na: https://www.codingninjas.com/studio/library/single-page-apps-vs-multi-page-apps. [Pokušaj pristupa 2. 9. 2023.].

[5] Wikipedia, »Ajax (programming),« [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Ajax_(programming). [Pokušaj pristupa 2. 9. 2023.].

[6] K. Nygard, »Single page architecture as basis for web applications,« [Mrežno]. Dostupno na: https://aaltodoc.aalto.fi/bitstream/handle/123456789/17773/master_Nyg%c3%a5rd_Klaus_2015.pdf?sequence=1&isAllowed=y. [Pokušaj pristupa 3. 9. 2023.].

[7] Ember, »What is Ember.js?,« [Mrežno]. Dostupno na: https://medium.com/aeturnuminc/what-is-ember-js-ff94403fec96. [Pokušaj pristupa 3. 9. 2023.].

[8] E. Molin, »Comparison of Single-Page Application,« [Mrežno]. Dostupno na: https://smallake.kr/wp-content/uploads/2016/09/eric_molin.pdf. [Pokušaj pristupa 3. 9. 2023.].

[9] Stackdiary, »The Most Popular Front-end Frameworks in 2023,« [Mrežno]. Dostupno na: https://stackdiary.com/front-end-frameworks/. [Pokušaj pristupa 3. 9. 2023.].

[10] Wikipedia, »Vue.js,« [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Vue.js. [Pokušaj pristupa 3. 9. 2023.].

[11] Google, »Google trends,« [Mrežno]. Dostupno na: https://trends.google.com/. [Pokušaj pristupa 3. 9. 2023.].

[12] Figma, »Figma,« [Mrežno]. Dostupno na: https://www.figma.com/. [Pokušaj pristupa 4. 9. 2023.].

[13] N. Hoić-Božić, »Osnove Web dizajna, Skripta za predmete "Multimedijski sustavi",« [Mrežno]. Dostupno na: https://moodle.srce.hr/2019-2020/pluginfile.php/3374582/mod_resource/content/1/web_dizajn_sve_2019.pdf. [Pokušaj pristupa 4. 9. 2023.].

[14] Wikipedia, »Angular (web framework),« [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Angular_(web_framework). [Pokušaj pristupa 5. 9. 2023.].

[15] Angular, »Angular,« [Mrežno]. Dostupno na: https://angular.io/. [Pokušaj pristupa 5. 9. 2023.].

[16] TypeScript, »TypeScript,« [Mrežno]. Dostupno na: https://www.typescriptlang.org/. [Pokušaj pristupa 5. 9. 2023.].

[17] Wikipedia, »HTML,« [Mrežno]. Dostupno na: https://hr.wikipedia.org/wiki/HTML. [Pokušaj pristupa 5. 9. 2023.].

[18] GeeksForGeeks, »What is the difference between CSS and SCSS?,« [Mrežno]. Dostupno na: https://www.geeksforgeeks.org/what-is-the-difference-between-css-and-scss/. [Pokušaj pristupa 5. 9. 2023.].

[19] Firebase, »Firebase documentation,« [Mrežno]. Dostupno na: https://firebase.google.com/docs. [Pokušaj pristupa 6. 9. 2023.].

[20] Wikipedia, »Node.js,« [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Node.js. [Pokušaj pristupa 6. 9. 2023.].

[21] npm, »About npm,« [Mrežno]. Dostupno na: https://docs.npmjs.com/about-npm. [Pokušaj pristupa 6. 9. 2023.].

[22] K. Stančin, »Git i GitHub,« [Mrežno]. Dostupno na: https://moodle.srce.hr/2022-2023/pluginfile.php/6907328/mod_resource/content/0/Git%20i%20Github.pdf. [Pokušaj pristupa 6. 9. 2023.].

[23] Microsoft, »Visual Studio,« [Mrežno]. Dostupno na: https://visualstudio.microsoft.com/. [Pokušaj pristupa 7. 9. 2023.].

[24] J. Dunlavy, »Angular Modules,« [Mrežno]. Dostupno na: https://fleury14.github.io/angular-class/09-modules.html. [Pokušaj pristupa 7. 9. 2023.].

Popis slika

Prilozi

Aplikacija „Croatian Cosmetics“ dostupna je na sljedećoj web adresi: https://cosmeticsapp-e222f.web.app/

Izvorni kod aplikacije dostupan je na online GitHub repozitoriju na web adresi: https://github.com/imagdic/cosmeticsapp