Aplikacija za prikaz i uređivanje digitalnog kataloga salona namještaja

Abramović, Luka

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

LUKA ABRAMOVIĆ

APLIKACIJA ZA PRIKAZ I UREĐIVANJE DIGITALNOG KATALOGA SALONA NAMJEŠTAJA

Diplomski rad

Pula, veljača 2023.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

LUKA ABRAMOVIĆ

APLIKACIJA ZA PRIKAZ I UREĐIVANJE DIGITALNOG KATALOGA SALONA NAMJEŠTAJA

Diplomski rad

JMBAG: 0303069232, redovni student
Studijski smjer: Informatika

Kolegij: Mobilne aplikacije
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijsko-komunikacijske znanosti
Znanstvena grana: Informacijski sustavi
Mentor: izv. prof. dr. sc. Siniša Sovilj

Pula, veljača 2023.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani LUKA ABRAMOVIĆ, kandidat za magistra INFORMATIKE ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Luka Abramović

U Puli, 22.2.2023.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, LUKA ABRAMOVIĆ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom APLIKACIJA ZA PRIKAZ I UREĐIVANJE DIGITALNOG KATALOGA SALONA NAMJEŠTAJA koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 22.2.2023.

Potpis

Luka Abramović

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Pula, 28. ožujka 2021.

DIPLOMSKI ZADATAK

Pristupnik: **Abramović Luka (0303069232)**
Studij: Sveučilišni diplomski studij Informatike

Naslov (hrv.): **Aplikacija za prikaz i uređivanje digitalnog kataloga salona namještaja.**
Naslov (eng.): Application for presenting and editing digital catalog of furniture salon.

Opis zadatka: Zadatak je izraditi aplikaciju za salon namještaja. Moblina aplikacija bi se sastojala od digitalnog kataloga salona namještaja, informacija o proizvodima (cijenama, dimenzijama i gdje su dostupni u fizičkim salonima), kontaktu sa salonom, postavljanje pitanja i traženje informacija o proizvodima na daljinu, te mogućnost online naručivanja proizvoda po posebnim mjerama. Administratorska web aplikacija bi se koristila za izmjenu podataka u bazi koje se prikazuju u mobilnoj te prihvaćanje upita od strane korisnika s mobilne aplikacije.
Aplikacija se treba temeljiti na tehnologijama Android sustava za izradu mobilne aplikacije te dodatno Vue.js za izradu administratorske web aplikacije. Mobilna i web aplikacija će zajedno komunicirati preko SQL baze podataka.
Opisati sustav: korisničke scenarije, funkcionalnosti, izraditi potrebne UML dijagrame - klasne, *use case*, *use sequence*, opisati implementaciju te na kraju izraditi kratke korisničke upute.

Zadatak uručen pristupniku: 28. ožujka 2021.
Rok za predaju rada: 28. veljače 2022.

Mentor:
Siniša Sovilj
_____
doc.dr.sc. Siniša Sovilj

SADRŽAJ

1. UVOD

Termin mobilnih aplikacija je u današnje vrijeme sveprisutan. Današnja tehnologija sve više teži tome da gotovo sve ono za što smo inače koristili računala (i za što ih koristimo i danas) se danas može staviti u svačiji džep i da bude pristupačno bilo kome, bilo kada, i bilo gdje. To je otvorilo i nove i pristupačnije načine brojnim kompanijama koje se bave prodajom da brže i lakše mogu doći do potencijalnih kupaca kojima će ponuditi svoje proizvode i usluge. To podjednako vrijedi i za trgovine koje se bave prodajom manjih artikala s kojima je jednostavno rukovati (knjige, dijelovi odjeće, razni sitni predmeti, namirnice, itd.), ali i za trgovine koje prodaju glomazne predmete poput, recimo, namještaja.

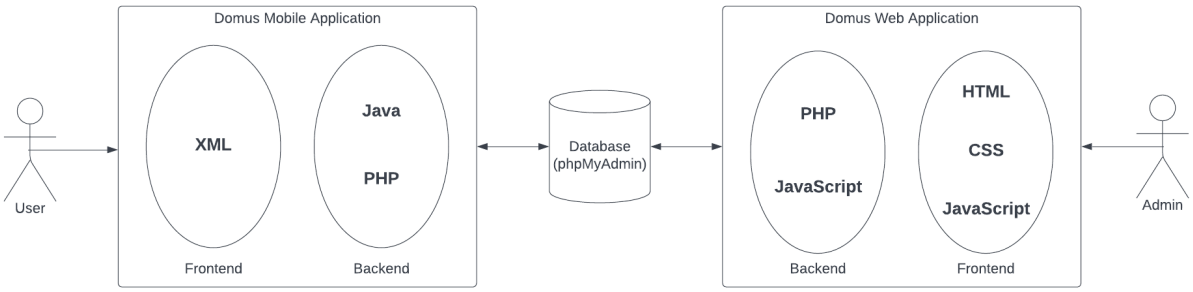
U svrhu ovog diplomskog rada razvijene su mobilna i web aplikacija za fiktivni salon namještaja Domus (od latinske riječi koja u prijevodu znači 'dom'). Spomenute aplikacije, mobilna i web, međusobno komuniciraju preko zajedničke baze podataka i tako ostvaruju neku vrstu indirektne komunikacije između administratora i korisnika. Pod administrator u ovom slučaju govorimo o zaposleniku poduzeća koji je zadužen za primanje narudžbi od korisnika i ažuriranje kataloga u koji prima podatke iz baze podataka i prikazuje ponudu potencijalnim kupcima, bilo da su oni korisnici mobilne ili web aplikacije.

Slika 1. *Domus* logotip

Moblina aplikacija razvijana je isključivo za korisnike, te za nju nije potrebno imati korisničko ime i lozinku. Na nju administrator ima utjecaj samo kada se preko web aplikacije vrši dodavanje i/ili ažuriranje ponude u bazi podataka koja se kasnije prikazuje korisniku u mobilnoj aplikaciji. Mobilna aplikacija je dostupna svim potencijalnim korisnicima, te se u njoj jednostavno može pregledati katalog proizvoda i potom i vršiti narudžba koja se šalje u bazu podataka i shodno tome administrator dobiva potrebne informacije o narudžbi pojedinog korisnika.

Web aplikacija stvorena je kao aplikacija koja je primarno razvijena za korištenje od strane administratora, ali opcionalno je mogu koristiti i korisnici. Za korištenje web aplikacije potrebno je korisničko ime i lozinka te je u bazi podataka precizno navedeno ima li određeni korisnik administratorska prava prilikom korištenja web aplikacije. Ako ima, tada se taj korisnik smatra administratorom i može jednostavno mijenjati ponudu u katalogu i pritom primati i pregledavati dospjele narudžbe drugih korisnika (bilo da one dolaze s mobilne ili s web aplikacije). Ako je korisnik prijavljen u sustav samo kao standardni korisnik, onda tada ne može pristupiti administratorskom dijelu aplikacije, ali i preko web aplikacije može jednostavno pretraživati katalog, vršiti narudžbe, i ktome pregledavati koje je narudžbe prethodno ostvario.

Za ovaj diplomski rad razvijen je projekt za čiju realizaciju se koristilo nekoliko različitih programskih jezika i tehnologija koje su danas uobičajene u izradi mobilnih i web aplikacija, kao i u razvoju baza podataka. Primarni programski jezik koji se koristio za izradu mobilne aplikacije je Java, dok je za razvoj web aplikacije korištena kombinacija programskih jezika, primarno i većinski PHP-a, ali i programskog jezika JavaScript. Označni (markup) jezici koji su se koristili za dizajniranje prednjeg sučelja (front-end) su bili XML u slučaju mobilne, te HTML u slučaju web aplikacije, te su uz HTML korišteni CSS i dodatni JavaScript koji su se pobrinuli za responzivnost web aplikacije i da prednje sučelje izgleda atraktivnije. Programski jezik PHP korišten je i za razvoj programerskog sučelja mobilne aplikacije (API) preko kojeg mobilna aplikacija komunicira s bazom podataka, koja je razvijena korištenjem administracijskog alata phpMyAdmin preko kojeg smo upravljali sustavom za upravljanje bazom podataka MySQL.



Slika 2. Veza između mobilne i web aplikacije i baze podataka uz prikazane korištene jezike

2. DIJAGRAMI JEZIKA ZA UNIFICIRANO MODELIRANJE (UML)

Kada se ulazi u razvoj novog projekta, jedna od bitnijih stvari na samom početku je dobro ga osmisliti i napraviti određenu skicu, a zatim tu skicu i ažurirati ili izmjenjivati ovisno o tome koliko se ideja projekta ili način izvođenja projekta mijenja tijekom razvoja. U branši koja se bavi programiranjem i razvojem aplikacija to je preporučeno raditi koristeći dijagrame jezika za unificirano modeliranje (UML). Važno je istaknuti da ovdje ne pričamo o programskom nego vizualnom jeziku, koji je »namijenjen pružanju standardnog načina vizualizacije dizajna sustava«[1], i pritom ga učini što jednostavnijim za shvatiti ljudima koji rade na projektu i/ili potencijalnim poslodavcima za koje se određeni projekt razvija.

Kompletni jezik za unificirano modeliranje sadrži nekoliko vrsta dijagrama koji omogućuju da svaki na sebi sopstveni način prikaže rad cijelog sustava ili jednog njegovog dijela. Za potrebe ovog projekta, dizajnirane su tri vrste dijagrama; dijagram slučaja (use-case diagram), dijagram slijeda (sequence diagram), i klasni dijagram. Važno je za napomenuti da su u ovom slučaju dijagram slijeda i klasni dijagram razlomljeni u više dijagrama zbog kompleksnosti cijelog projekta i praktičnijeg čitanja svakog dijagrama.

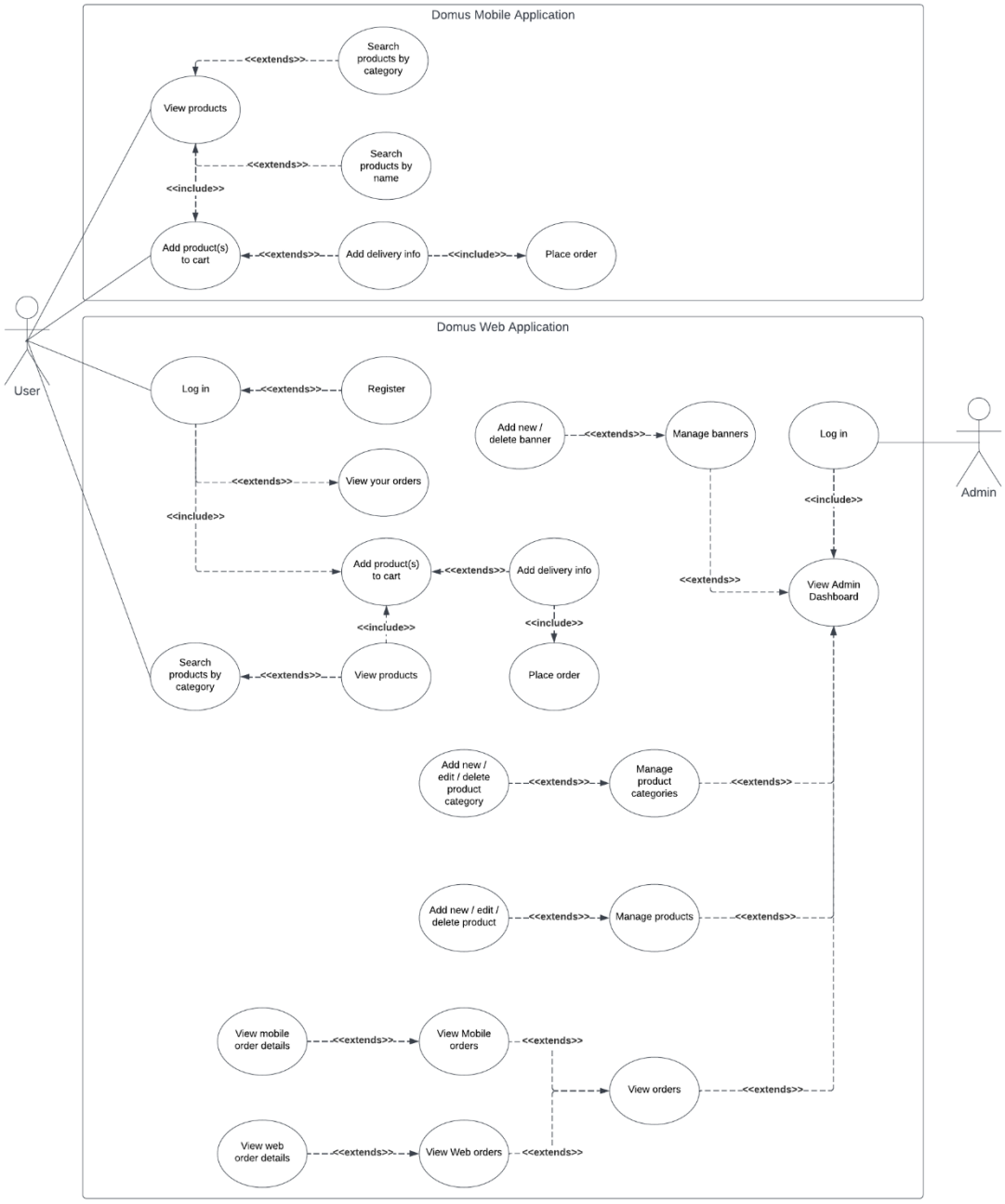
2.1. Dijagram slučaja

Dijagram slučaja je vrsta dijagrama čija je svrha »pokazati različite načine na koje korisnik može komunicirati sa sustavom.«[2] Svaki korisnik sustava, u kojem god on bio svojstvu, prikazan je u dijagramu kao nacrtani čovječuljak te se prikazuje svaki slučaj koji s kojim se bilo koji korisnik može susresti prilikom korištenja sustava. U ovom slučaju, standardni korisnik i administrator su dvije različite vrste korisnika, te dijagram slučaja svojim prikazom zapravo daje ideju, odnosno vizualni prikaz na sve mogućnosti koje osoba koja koristi aplikaciju ima, ovisno od toga je li ta osoba standardni korisnik ili administrator.

S obzirom na to da ovaj cjelokupni projekt se sastoji od mobilne i web aplikacije, radi lakšeg razumijevanja dijagrama su ta dva sustava raspoređena u dva različita kontejnera. Dijagram prikazuje kako standardni korisnik ima mogućnost korištenja mobilne, ali i web aplikacije u svojstvu standardnog korisnika. Kako je prikazano u dijagramu, korisnik ima sve mogućnosti koje mu mobilna aplikacija nudi, a to je pregledavanje ponuđenih proizvoda, što se može raditi preko odabira kategorije proizvoda ili preko trake za pretraživanje u koju se upisuje ime proizvoda te sustav potom prikazuje dostupne proizvode s upisanim imenom. Uz to, dijagram slučaja prikazuje i mogućnost korisnika da željene proizvode dodaje u košaricu te potom upisivanjem potrebnih osobnih podataka ostvaruje narudžbu koja se sprema u bazu podataka i shodno tome će biti vidljiva administratoru preko web aplikacije.

[1] Booch, Grady; Rumbauch, James; Jacobson, Ivar, *The Unified Modeling Language User Guide*, Addison Wesley Professional, 2005
[2] Lucid Software Inc., "UML Use Case Diagram Tutorial", lucidchart.com/pages/uml-use-case-diagram

Dijagram slučaja prikazuje i kakvu interakciju korisnik može ostvariti sa sustavom web aplikacije. Iako korisnik može nesmetano pretraživati katalog proizvoda u web aplikaciji, za dodavanje proizvoda u košaricu i kreiranje narudžbe je potrebno prijaviti se korisničkim imenom i lozinkom. Time prijavljeni korisnik ima mogućnost ne samo da kreira novu narudžbu nego i pregleda sve prijašnje narudžbe koje je ostvario.



Slika 3. Dijagram slučaja

Kao osoba, administrator nije ništa više od standardnog korisnika dok se ne prijavi u sustav. Kako je prikazano u dijagramu slučaja, administrator se mora prijaviti svojim vlastitim korisničkim imenom i lozinkom (u bazi podataka su administratori posebno

označeni kao korisnici koji imaju pravo pristupiti administratorskom dijelu web aplikacije). Svojom prijavom, administratoru se otvara nadzorna ploča iz koje može vršiti sve administratorske uvide i promjene. To uključuje pregledavanje narudžbi koje su korisnici kreirali (kreirane narudžbe su podijeljene u dvije skupine; one koje su kreirane preko mobilne, i one koje su kreirane preko web aplikacije). Administrator uz to ima i mogućnost dodavanja, izmjene, ili brisanja ponuđenih kategorija proizvoda, kao i samih proizvoda. Uz to, korisnik preko web aplikacije može dodavati i brisati slike koje se korisniku u mobilnoj aplikaciji prikazuju kao naslovne slike.

2.2. Dijagram slijeda

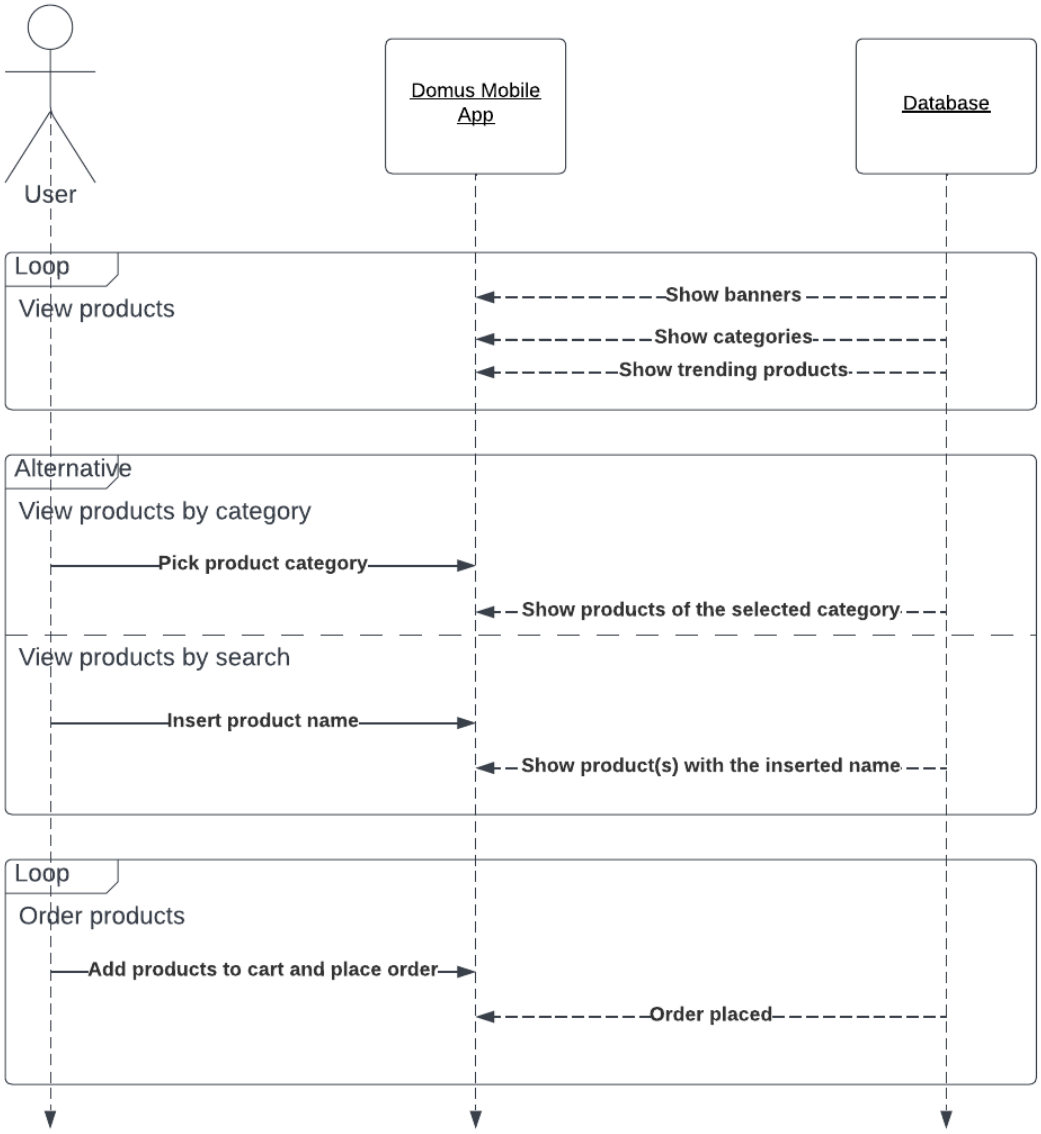
Dijagrami slijeda ili sekvenci koriste se »prikaz interakcija između objekata redoslijedom kojim se te interakcije događaju.«[3] Pritom se prikazuju svi procesi koji se razmjenjuju između aplikacije i baze podataka kako bi se izvršila određena funkcija te kako bi aplikacija reagirala na način na koji korisnik od nje očekuje.

Za razliku od dijagrama slučaja, dijagram slijeda je u slučaju ovog projekta razlomljen na tri različita dijagrama od kojih se jedan prikazuje procese između mobilne aplikacije i baze podataka, dok druga dva prikazuju isto za web aplikaciju, ovisno o tome radi li se o standardnom korisniku ili administratoru.

Dijagram slučaja za mobilnu aplikaciju (slika 4.) prikazuje samo procese koji se razmjenjuju između mobilne aplikacije i baze podataka. Čim korisnik pristupi aplikaciji, aplikacija mora automatski povući iz baze podataka naslovne fotografije, kategorije proizvoda te proizvode koji su trenutno najpopularniji (što administrator zadaje prilikom kreiranja ili izmjene određenog proizvoda u nadzornoj ploči). Te tri stavke se odmah prikazuju po ulasku korisnika u aplikaciju. Potom, korisnik ima mogućnost pretražiti proizvode koje želi bilo odabirom kategorije proizvoda ili pretraživanjem preko tražilice. U oba slučaja, pozadinski program filtrira proizvode koje te ih prikazuje korisniku ovisno o odabranoj kategoriji ili ovisno o upisanom pojmu u tražilicu. Time se korisniku olakšava navigiranje kroz katalog pregledavanje ponuđenih proizvoda.

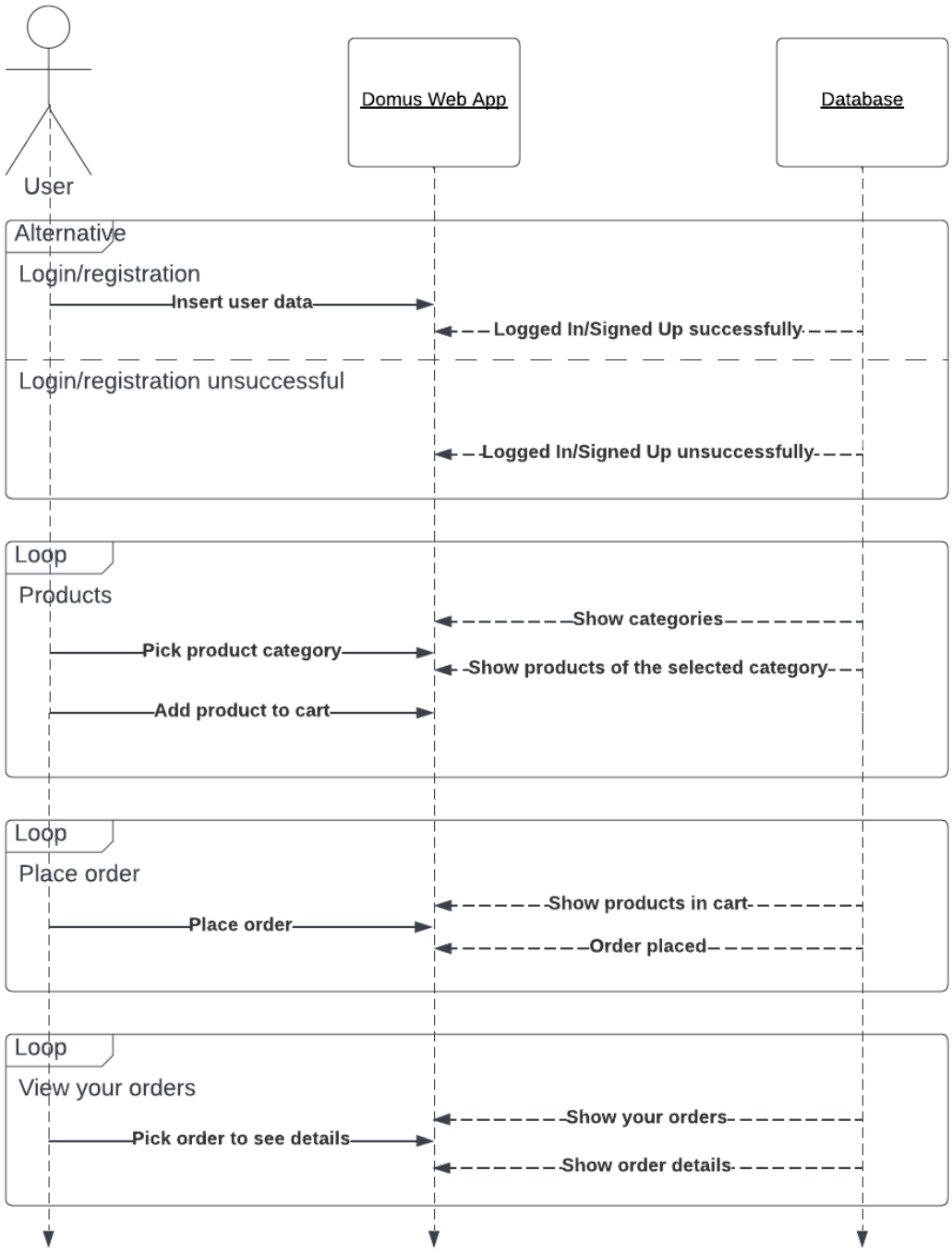
Jednom kada korisnik pronađe željeni proizvod ili više njih, u mogućnosti je dodati u standardnu košaricu, čiji sustav inače koriste aplikacije za internet prodaju, i u konačnici kreirati narudžbu koja se šalje u bazu podataka čime će ju administrator preko web aplikacije dobiti na uvid.

[3] Bell, Donald, "Explore the UML sequence diagram", developer.ibm.com/articles/the-sequence-diagram



Slika 4. Dijagram slijeda za mobilnu aplikaciju

Za web aplikaciju kreirana su dva dijagrama slijeda kojima je ključna razlika u tome radi li se o standardnom korisniku ili administratoru. U slučaju standardnog korisnika (slika 5.), najosnovnija stvar za pristup svim funkcijama aplikacije je odraditi prijavu korisničkim imenom i lozinkom. Jednom kada korisnik upiše podatke, pozadinska PHP skripta provjerava odgovaraju li upisani podaci onima koji se nalaze u bazi podataka. U slučaju da prijava uspije, korisniku se omogućuje pristup svim funkcijama aplikacije, u suprotnom se vraća na početak prijave. Isto vrijedi i za registraciju korisnika koji se prijavljuje po prvi put, uz razliku da se novoupisani podaci spremaju u bazu podataka (osim ako su u koliziji s nekim već postojećim podacima u bazi), kako bi se kasnije koristili za naknadne prijave korisnika u sustav.



Slika 5. Dijagram slijeda za web aplikaciju u slučaju standardnog korisnika

Iako web aplikacija funkcionira na drugačiji način, u dijagramu slijeda su prikazane na sličan način zbog suštinski iste ideje. Iako u web aplikaciji nema tražilice za pretraživanje, i dalje korisnik započinje traženje željenog proizvoda birajući neku od ponuđenih kategorija proizvoda koji se prikazuju zahvaljujući PHP skripti koja povlači podatke o kategorijama iz baze podataka. Po odabiru željene kategorije, korisniku se prikazuju proizvodi koji su u bazi podataka označeni da pripadaju kategoriji koju je korisnik

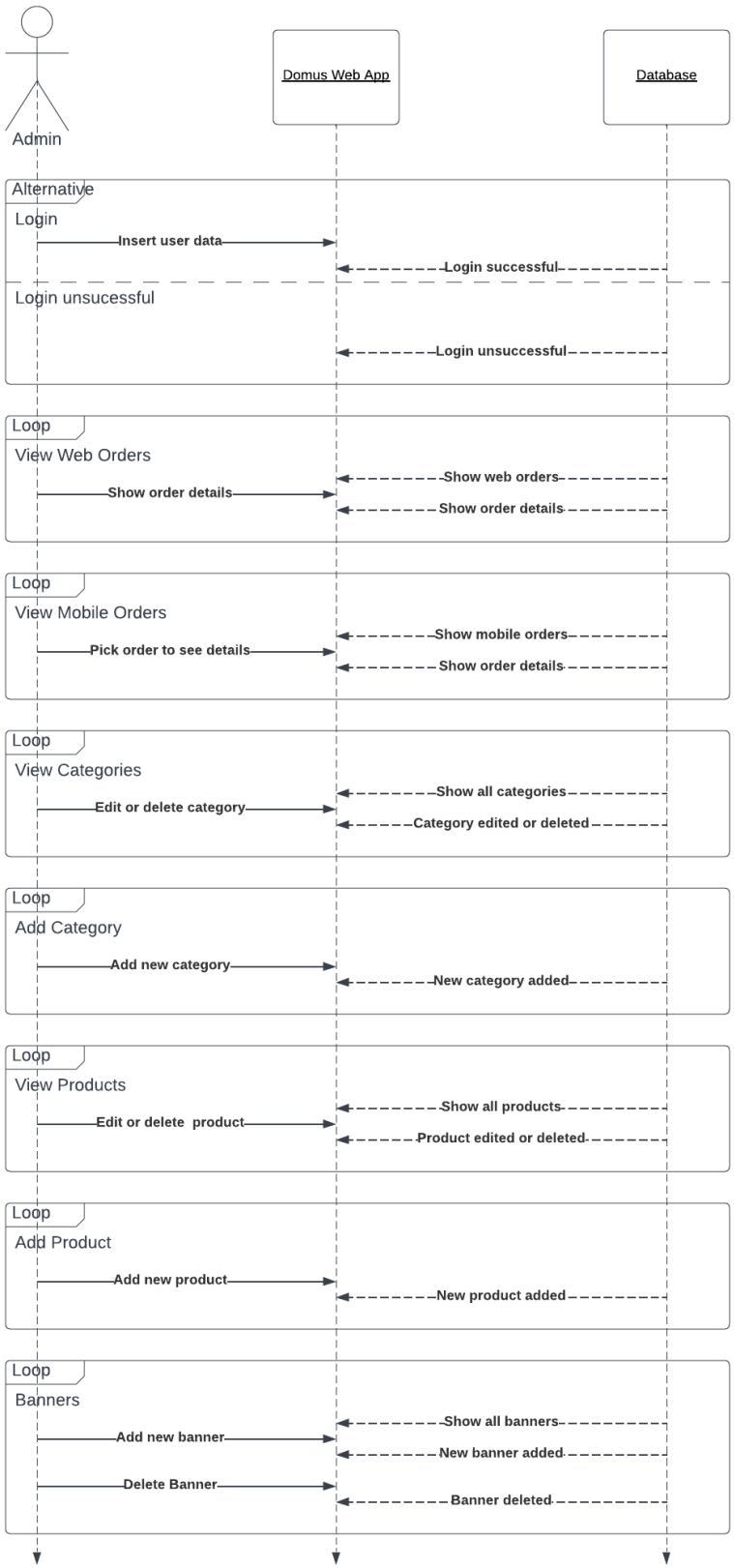
prethodno odabrao. Korisnik potom ima mogućnost dodati željeni proizvod ili više njih u košaricu.

Odabirom na pregled košarice, korisnik može vidjeti sve proizvode koje je spremio u košaricu, čiji su proizvodi, odnosno podaci o proizvodima spremljeni u posebnu tablicu koja će se isprazniti onog momenta kada korisnik potvrdi narudžbu ili ukloni proizvode iz košarice, odnosno prestane s kreiranjem narudžbe. Podaci o trenutnom stanju košarice se također preko pozadinske PHP skripte povlače iz spomenute tablice i prikazuju korisniku u aplikaciji, nakon čega korisnik može potvrditi narudžbu čiji se cjelokupni podaci spremaju u dvije tablice, jedna koja sadrži općenite podatke o kreiranoj narudžbi te druga koja sadrži podatke o proizvodima unutar te narudžbe.

Posljednji, treći dijagram slijeda prikazuje odnos između web aplikacije i baze podataka prilikom rada administratora. Kako je i ranije spomenuto, administrator nije ništa više od standardnog korisnika prije obavljanja obavezne prijave u sustav. Administrator vrši prijavu u sustav prilikom koje se opet provjerava ispravnost unesenih podatka u bazi. Ako uneseni podaci nisu ispravni, administrator je vraćen na početak prijave. Ako je prijava uspješna, web aplikacija se automatski preusmjerava prema nadzornoj ploči preko koje administrator može vršiti uvid u zaprimljene narudžbe te vršiti izmjene u prikazu naslovnih slika, te u ponudi proizvoda i njihovim kategorijama.

Kako se već i kod dijagrama slučaja okvirno objasnili, administrator ima mogućnost pregledavati narudžbe koje su korisnici kreirali i spremili u bazu podataka. Shodno tome, svi podaci o narudžbama se automatski povlače iz baze podataka onog trenutka kada administrator odabere želi li pregledati mobilne ili web narudžbe. Važno je za naglasiti da se u tom trenutku samo povuku općeniti podaci o narudžbama, dok se podaci o proizvodima u određenoj narudžbi povlače onog trenutka kada administrator odluči pregledati detalje neke određene narudžbe. Ti detalji uključuju popis proizvoda u toj narudžbi, kao i podatke o vitalne podatke o korisniku koji je izvršio narudžbu.

Osim pregledavanja narudžbi, administrator ima mogućnost dodavanja novih kategorija proizvoda, kao i izmjenu ili brisanje već postojećih. Kada korisnik želi dodati novu kategoriju, nikakvi podatci se ne povlače iz baze podataka, ali zato se po upisanim podacima u kategoriji svi ti podaci spremaju u bazu podataka kao nova kategorija. Ukoliko administrator želi izmijeniti ili obrisati neku od postojećih kategorija proizvoda, tada je potrebno povući sve postojeće kategorije iz baze kako bi se moglo odabrati nad kojom kategorijom proizvoda je potrebno izvesti izmjenu i/ili brisanje podataka. Ista stvar vrijedi i za same proizvode. U slučaju naslovnih fotografija, tada je situacija malo drugačija jer nema mogućnosti izmjene, samo mogućnost dodavanja nove fotografije, ili brisanje već postojeće. Ali princip kojim se to dodavanje, odnosno brisanje iz baze podataka izvodi je potpuno isti.

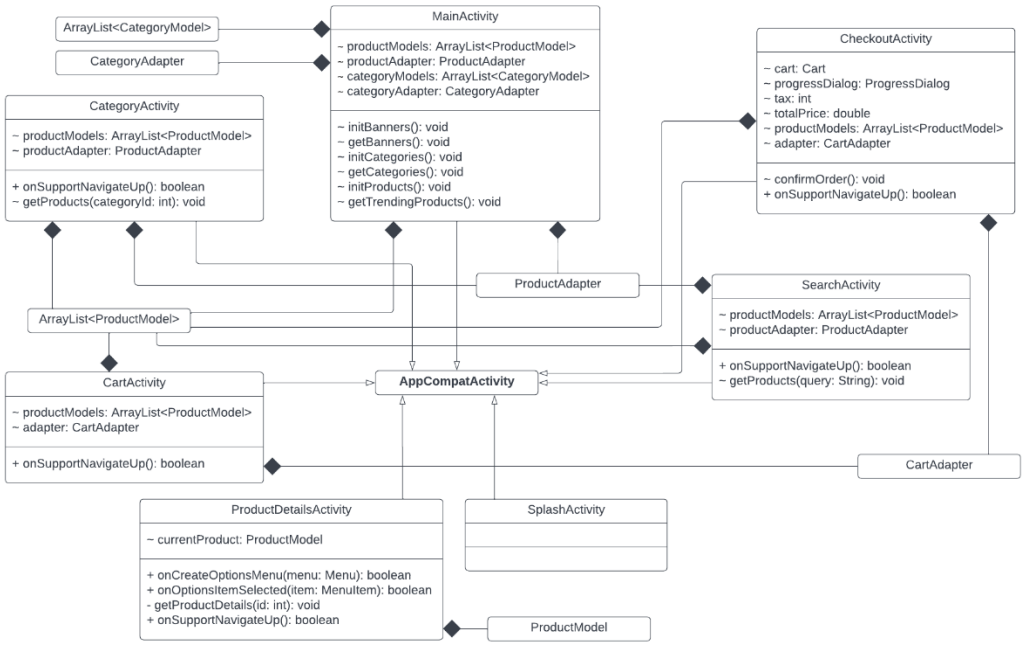


Slika 6. Dijagram slijeda za web aplikaciju u slučaju administratora

2.3. Klasni dijagram

»Klasni dijagram jedan je od najkorisnijih vrsta dijagrama u UML-u budući da jasno prikazuju strukturu određenog sustava modeliranjem njegovih klasa, atributa, operacija i odnosa između objekata.«[4] Shodno tome, klasni dijagrami su specifični za objektno-orijentirano programiranje. Prilikom razvoja mobilne aplikacije koristio se Java, programski jezik koji se bazira na klasama i objektno-orijentiranom programiranju. S obzirom na kompleksnost cijelog sustava, korištene klase u projektu su podijeljene na aktivnosti, adaptere, i modele, i shodno tome je i kompletni klasni dijagram razlomljen na tri dijagrama.

Prvi od tri klasna dijagrama je klasni dijagram koji predstavlja klase aktivnosti (activity). »Aktivnost je komponenta aplikacije koja pruža zaslon s kojim korisnici mogu ostvariti interakciju kako bi nešto učinili, poput biranja broja telefona, snimanja fotografije, slanja e-pošte ili pregledavanja karte. Svaka aktivnost dobiva prozor u kojem se kreira korisničko sučelje.«[5] Ovaj projekt ukupno broji sedam klasa aktivnosti, od kojih jedna (SplashActivity), postoji samo radi vizualnog uvoda u aplikaciju, te ne sadrži nikakve atribute ili operacije. Ostale klase se sastoje od atributa i operacija koji omogućavaju da aplikacija uspješno izvodi sve svoje temeljne funkcije. Za razumijevanje ovog klasnog dijagrama je važno znati što je AppCompatActivity, »osnovna klasa za aktivnosti koje žele koristiti neke od značajki novije platforme na starijim Android uređajima.«[6] Jedna od tih značajki je i naslovna traka koja je također dio većeg dijela aktivnosti mobilne aplikacije (izuzetak je SplashActivity).



Slika 7. Klasni dijagram, aktivnosti

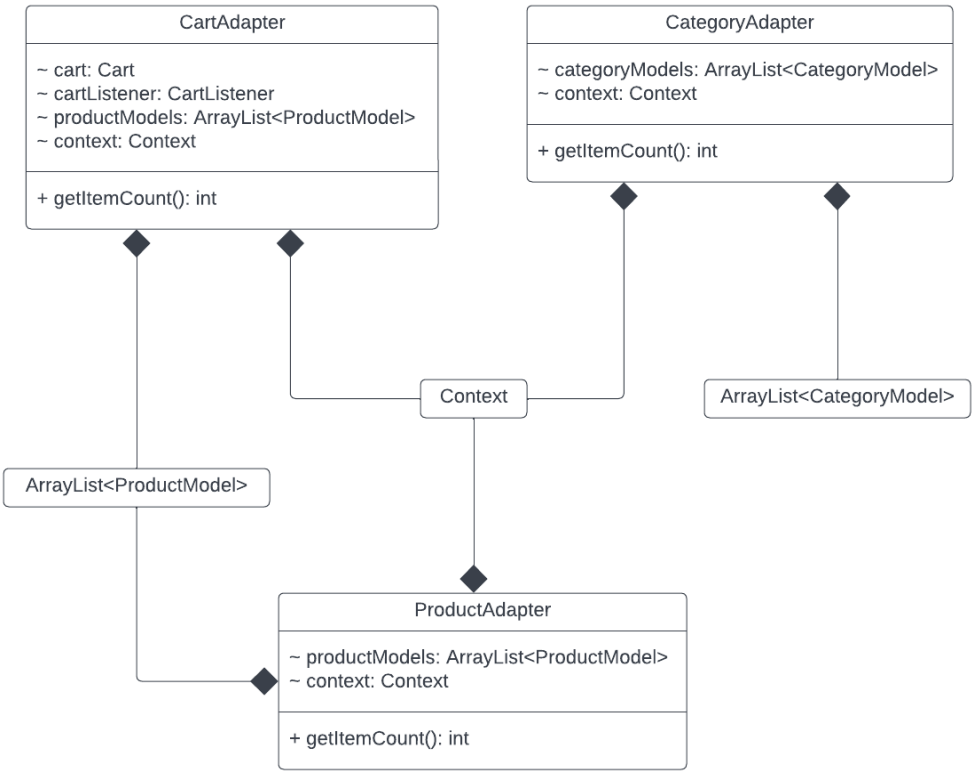
[4] Lucid Software Inc., "UML Class Diagram Tutorial", lucidchart.com/pages/uml-class-diagram
[5] "Activities", stuff.mit.edu/afs/sipb/project/android/docs/guide/components/activities.html
[6] Android Developers, "AppCompatActivity", developer.android.com/reference/androidx/appcompat/app/AppCompatActivity

Kako se i vidi u klasnom dijagramu aktivnosti (slika 7.), sve klase aktivnosti su zapravo potklase koje su generalizacijom povezane sa superklasom AppCompatActivity. Drugim riječima, klase aktivnosti su ovom slučaju klase djeteta koje nasljeđuju ponašanje superklase.

Glavna klasa aktivnosti (MainActivity) sadrži četiri atributa koji su zapravo paketi koji su s klasom povezani kompozicijom. Pojednostavljeno, ukoliko bi se, teoretski, ta glavna klasa obrisala, i klase koje su kompozicijom spojene s njom bi također bile obrisane. U glavnoj klasi vidimo da koristi modele i adaptere proizvoda i kategorija, s tim da su modeli deklarirani kao nizovi podataka s koji će biti potrebni toj glavnoj klasi. Operacije koje glavna klasa koristi su sve privatne i koriste se za dohvat, odnosno slanje naslovnih fotografija, kategorija, i proizvoda, respektivno.

Klasa aktivnosti kategorija (CategoryActivity) sadrži i model i adapter proizvoda kao dva paketna atributa potrebna za rad. Zadaća ove klase, odnosno aktivnosti je da dohvati proizvode koji spadaju u određenu kategoriju i time se zapravo proizvodi filtriraju po kategorijama prilikom korisnikovog pretraživanja kataloga. Na vrlo sličan način funkcionira i klasa aktivnosti pretraživanja (SearchActivity), koja je programirana za rad tražilice, odnosno korištenje pojma kojeg je korisnik upisao u tražilicu kako bi dohvatio i prikazao proizvode čije ime sadrži taj traženi pojam. S druge strane, klasa aktivnosti detalja proizvoda (ProductDetailsActivity) zadužena je dohvat i prikazivanje dodatnih informacija o proizvodu kada korisnik odabere da neki proizvod kako bi ga detaljnije proučio. Klase aktivnosti košarice (CartActivity) i aktivnosti ovjere narudžbe (CheckoutActivity) služe za okupljanje proizvoda koje je korisnik spremio u košaricu i koje je potom spreman naručiti. Te dvije aktivnosti koriste implementiranu ovisnost (dependency) TinyCart[7] koja je omogućila programiranje rada košarice i potom slanje narudžbe u bazu podataka.

[7] TinyCart, github.com/hishd/TinyCart



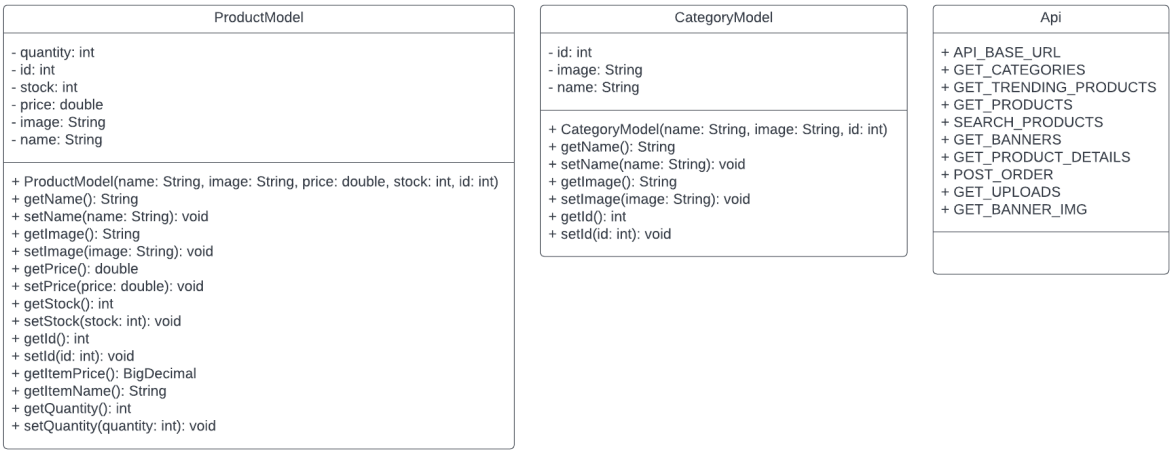
Slika 8. Klasni dijagram, adapteri

»U Androidu, adapter je most između komponente korisničkog sučelja i izvora podataka koji nam pomaže ispuniti podatke u komponenti korisničkog sučelja.«[8] Funkcionira tako da je u adapteru zapravo zadano što će se prikazati u nekom od načina pogleda kao što su RecyclerView, ListView, ili GridView. Klasnom dijagramu koji prikazuje adaptere vidimo da se u mobilnoj aplikaciji koriste tri adaptera, za kategorije proizvoda, proizvode, i košaricu. Sve tri klase su kompozicijom povezane s kontekstom, »apstraktnom klasom čiju implementaciju osigurava Android sustav. Omogućuje pristup resursima i klasama specifičnima za aplikaciju, kao i pozive za operacije na razini aplikacije kao što je pokretanje aktivnosti.«[9]

Klasa adapter kategorija (CategoryAdapter) povezana je s listom modela kategorija kako bi se omogućilo dohvat liste proizvoda za tu specifičnu kategoriju, odnosno kako bi se ta lista preko adaptera prikazala u korisničkom sučelju aplikacije. Na sličan način funkcionira i adapter proizvoda (ProductAdapter) koji sadrži atribut paketa te je povezana s listom modela proizvoda. Konačno, klasa adaptera košarice (CartAdapter) također kao atribut ima pakete iz već spomenute implementirane ovisnosti TinyCart. Sve tri klase u ovom dijagramu imaju operaciju getItemCount koja služi za prebrojavanje subjekata (kategorija ili proizvoda) u tim aktivnostima.

[8] Abhishek Saini, "Adapter Tutorial With Example In Android Studio", abhiandroid.com/ui/adapter
[9] Android Developers, "Context", developer.android.com/reference/android/content/Context

Završni, treći klasni dijagram sastoji se od modela kategorija (CategoryModel) i proizvoda (ProductModel). Ideja programiranja u programskom jeziku Java je uvijek bila "napiši jednom, pokreni bilo gdje", i to je u biti ono što ovi modeli služe, jer su njihovi atributi zapravo samo atributi određenog entiteta u bazi podataka, dok se operacije koriste za dohvat (getter) ili postavljanje vrijednosti (setter) atributa.



Slika 9. Klasni dijagram, modeli i programsko sučelje aplikacije (API)

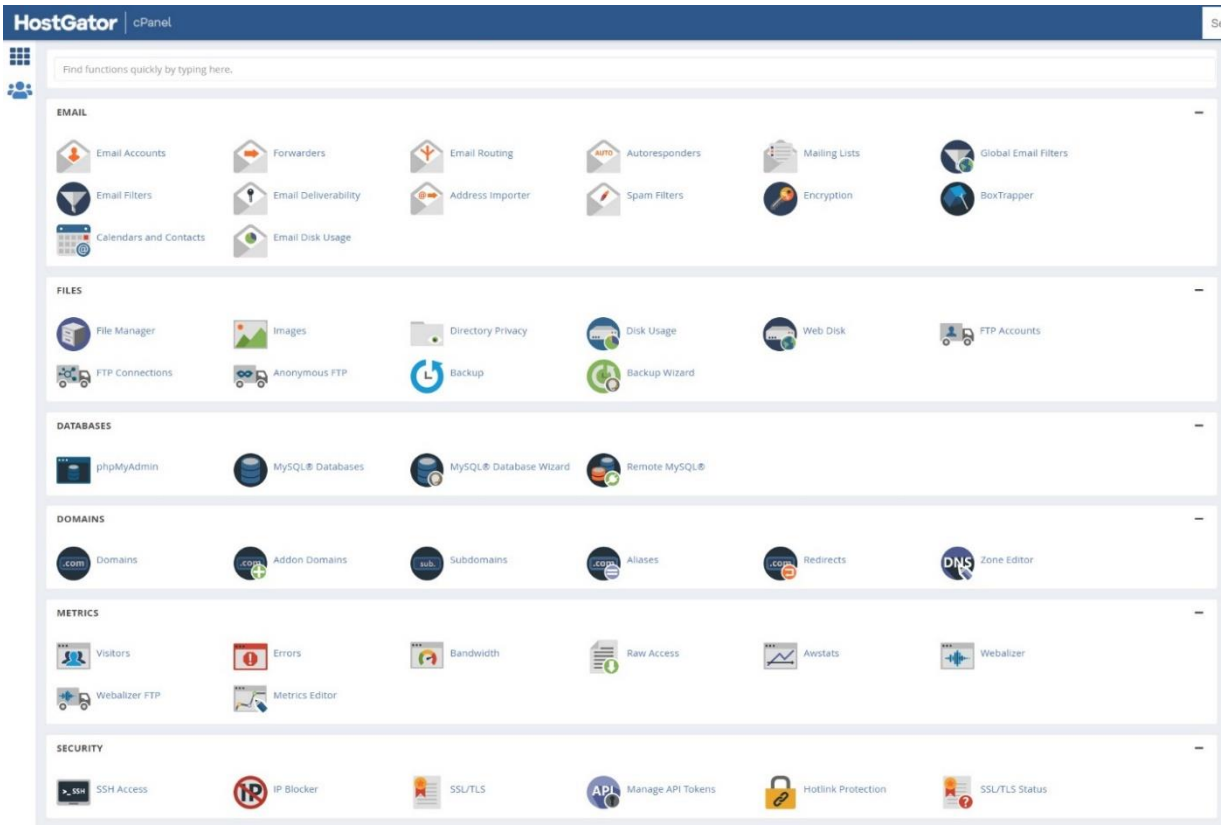
Programsko sučelje aplikacije pisano je u zasebnoj klasi Api u kojoj su definirane URL poveznice na PHP skripte na serveru koje izvode zadane zadatke u bazi podataka. Atribut API_BASE_URL sastoji se od osnovne URL adrese dok se ostali atributi sastoje od pozivanja te osnovne URL adrese i puta (path) koji vodi do te PHP skripte koja izvodi potrebna dohvaćanja (GET) ili postavljanja (POST) podataka u bazu podataka.

3. POČETNO POSTAVLJANJE PROJEKTA

Mobilna i web aplikacija ovog projekta su zbog svojih različitih vrsta se trebale razvijati u različitim programskim okruženjima. Mobilna aplikacija razvijena je korištenjem programa Android Studio koji čija je i glavna namjena razvijanje mobilnih aplikacija za android uređaje, dok je web aplikacija razvijena na online serveru i pisana primarno korištenjem programskog jezika PHP. Uz korištenje HTMLa i CSSa za dizajn korisničkog sučelja. Dodatno je i za potrebe i prednjeg i stražnjeg sučelja web aplikacije korišten i programski jezik JavaScript. Programski jezik PHP korišten je i za pisanje skripti koje služe kao most između mobilne aplikacije i baze podataka.

3.1. HostGator

Web aplikacija i kompletna baza podataka, kao i PHP skripte koje mobilna aplikacija poziva kada je potrebno pisani su i pohranjeni na mrežnoj usluzi poslužitelja (hosting) HostGator. Prema recenziji PCMag-a, HostGator je »izvrsna usluga web hostinga koja nudi niz moćnih alata, uključujući izvrstan alat za izradu web stranica za blogere i mlada poduzeća.«[10] Iako HostGator ima mogućnost pružanja usluge hostinga i za WordPress kako bi se olakšalo stvaranje web stranica, nama za ovaj projekt to nije bilo potrebno.



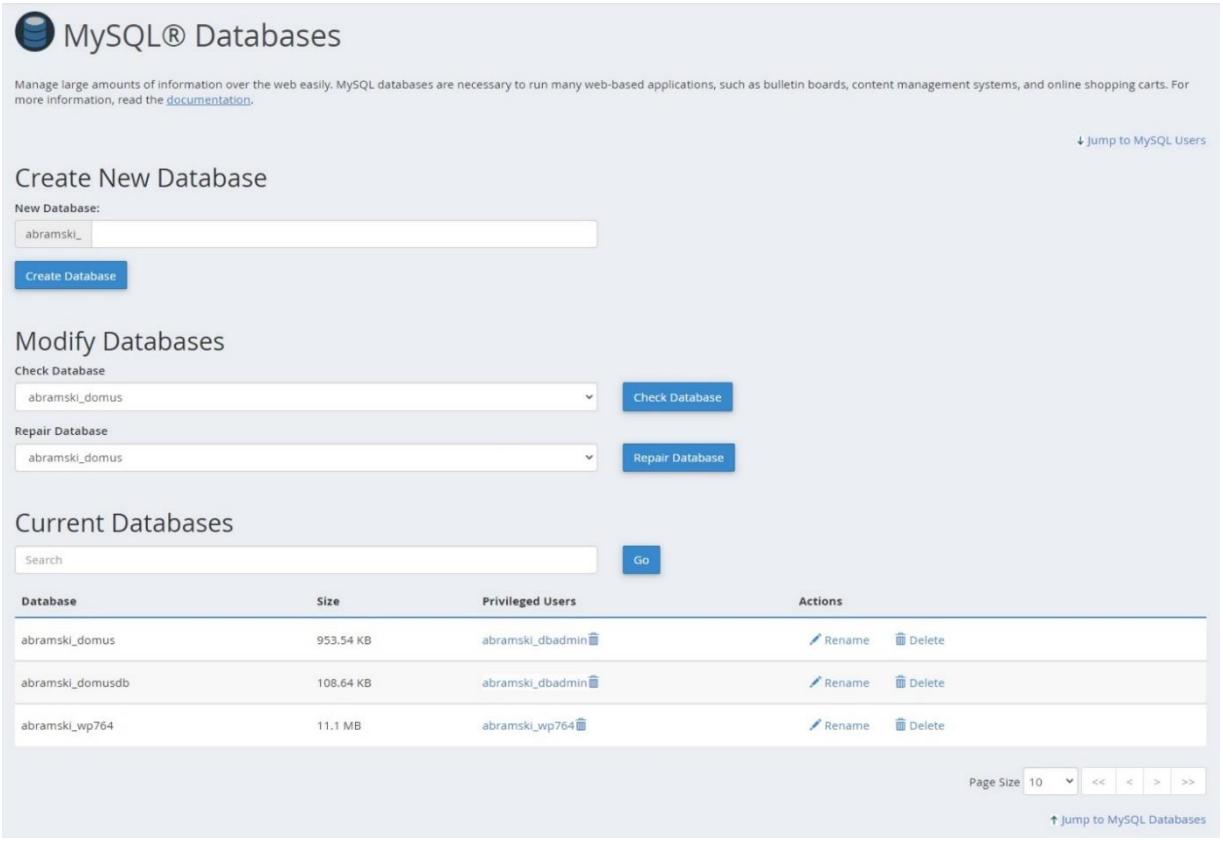
Slika 10. HostGator, kontrolna ploča

[10] L. Wilson, Jeffrey; Zamora, Gabriel, "HostGator Web Hosting Review", pcmag.com/reviews/hostgator-web-hosting

Za korištenje HostGatora bilo je potrebno registrirati domenu[11], kako bi se temelji za ovaj projekt uopće mogli početi postavljati. Jednom kada se taj lakši dio posla odradio, bilo je potrebno postaviti sve što će biti potrebno za daljnji razvoj ovog projekta. To je prije svega baza podataka. HostGator na svojoj kontrolnoj ploči (cPanel, slika 10.) ima mogućnost upravljanja svime što nam je od njega potrebno. U našem slučaju, potrebni su bili alati za postavljanje i upravljanje bazama podataka i njihovim korisnicima (MySQL Databases i phpMyAdmn), a kasnije će biti potreban i upravitelj datoteka (File Manager).

3.2. Postavljanje baze podataka

Prije nego što se uopće upustimo u programiranje, odnosno implementaciju cijelog ovog projekta, potrebno je bilo postaviti bazu podataka koja je neophodna za pravilo funkcioniranje i mobilne i web aplikacije. Shodno tome, nije niti potrebno naglašavati da su prilikom razvoja projekta i njegovog testiranja svi uređaji (osobno računalo i pametni telefon za testiranje mobilne aplikacije) koji su se koristili morali biti spojeni na internet mrežu. Drugim riječima, morali su biti "online".

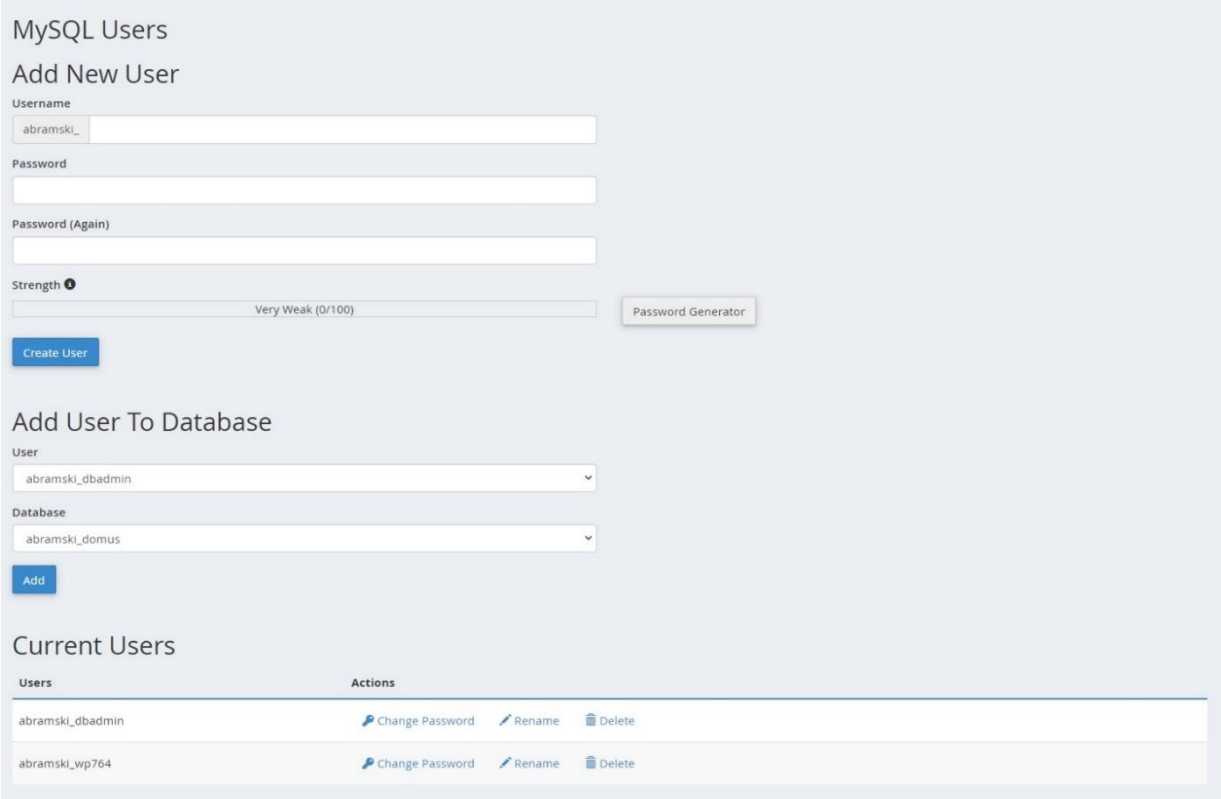


Slika 11. HostGator, MySQLDatabases

Izvorni proces postavljanja baze podataka započinje klikom na MySQL Databases (slika 11.) na HostGator kontrolnoj ploči. Otvaranjem tog prozora dobivamo mogućnost stvaranja nove, ali i modificiranja, preimenovanja, i brisanja već postojećih baza podataka. Svakoj novostvorenoj bazi podataka daje se ime koje sadrži i prefiks, te je

[11] abramovicluka-diplomski.com

shodno tome puno ime baze podataka *abramski_domus*. Po kreiranju nove baze podataka, nužno je deklarirati i najmanje jednog MySQL korisnika (slika 12.) kojem se dodjeljuje privilegija upravljanja netom stvorenom bazom podataka.



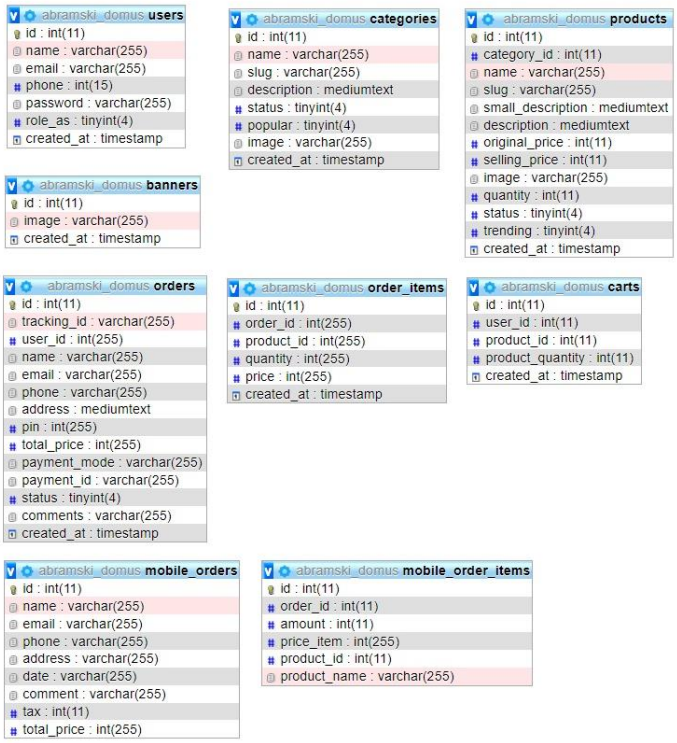
Slika 12. HostGator, MySQL korisnici

Korisničko ime korisnika kojem se dodjeljuje privilegija upravljanja bazom podataka također ima jednaki prefiks kao i ime baze. Po odabiru željenog korisničkog imena potrebno je odabrati i pripadajuću lozinku. Iako se može odabrati bilo kakva lozinka, preporučljivo je koristiti generator lozinke koji automatski upisuje nasumično generiranu lozinku koju je potrebno zapisati negdje sa strane jer će nam biti neophodna za daljnji razvoj cijelog projekta. Nakon potvrde upisanih podataka i kreiranja novog korisnika, potrebno je tog korisnika dodati bazi podataka nad kojom će on imati privilegiju upravljati njome. Na slici 11. se može vidjeti koji korisnik je dodan određenoj bazi podataka kao privilegirani korisnik. Podaci o bazi podataka i korisniku bit će neophodni kada bude potrebno deklariranje povezanosti s bazom podataka u PHP-u.

Iako se u PHP skripti može napisati SQL naredba za kreiranje tablice u bazi podataka, SQL naredbe u PHP skriptama ticat će se samo dodavanja, mijenjanja, i brisanja podataka iz tablica, dok same tablice su "ručno" kreirane u softverskom alatu phpMyAdmin kojem se također može pristupiti iz HostGator kontrolne ploče. Kompletna baza podataka korištena za ovaj projekt sadrži ukupno devet tablica. Prva tablica koju je bilo potrebno kreirati je tablica korisnika (users) u kojoj su spremljeni podaci korisnika koji su potrebni za prijavu u sustav web aplikacije. Potrebno je za naglasiti da su u ovoj tablici smješteni podaci i standardnih korisnika i administratora, a stupac *role_as* obilježava o

kojoj vrsti korisnika je riječ i na osnovu toga sprječava standardnog korisnika da se prijavi kao administrator, iako administrator može koristiti aplikaciju kao standardni korisnik.

Ostale potrebne tablice su bile kategorija (categories) i proizvoda (products) u koje administrator preko web aplikacije dodaju, izmjenjuje, ili briše podatke, dok i mobilna i web aplikacije koriste ove tablice za dohvat informacija o kategorijama, odnosno proizvodima i tako ih prikazuje standardnim korisnicima u svom korisničkom sučelju. Na isti način funkcionira i tablica naslovnih fotografija (banners). Preostale tablice bave se spremanjem narudžbi korisnika te se njihovi podaci prikazuju administratoru u njegovom dijelu web aplikacije. Važno je za napomenuti da su se podaci o narudžbama i proizvodima koje one sadrže spremaju u različite tablice ovisno o tome jesu li narudžbe kreirane preko mobilne i web aplikacije. U tom kontekstu, glavne tablice su tablice narudžbi (*orders* za web aplikaciju i *mobile_orders* za mobilnu aplikaciju), dok se informacije o svakom proizvodu u pojedinoj narudžbi nalaze u tablicama o stavkama narudžbe (*order_items* za web aplikaciju i *mobile_order_items* za mobilnu aplikaciju). U tim tablicama je spremljen i identifikacijski broj narudžbe iz glavne tablice kako bi se znalo koja stavka pripada kojoj narudžbi.



Slika 13. phpMyAdmin, prikaz tablica u dizajnerskom načinu rada

Dodatno, postoji i tablica košarica (*carts*) na kojoj izmjene se vrše isključivo preko web aplikacije. Ta tablica dinamički upravlja proizvodima koje je korisnik preko web aplikacije stavio ili uklonio iz košarice. Po kreiranju nove narudžbe u tablici narudžbi (*orders*), sav sadržaj košarice se prebacuje u tablicu stavke narudžbe (*order_items*), zajedno sa pripadajućim identifikacijskim brojem iz tablice narudžbi. Tablica košarice ne postoji za narudžbe koje se kreiraju preko mobilne aplikacije zato što je u mobilnoj aplikaciji

implementirana ovisnost TinyCart koja sama skuplja proizvode i podatke o košarici prije nego ih narudžba bude završena i spremljena u tablicu mobilnih narudžbi (*mobile_orders*) zajedno sa stavkama te narudžbe (*mobile_order_items*). Podatke iz tablice mobilnih narudžbi i podatke o stavkama tih narudžbi web aplikacije i dalje dohvaća kako bi administrator i dalje imao uvid.

3.3. Organizacija u upravitelju datoteka

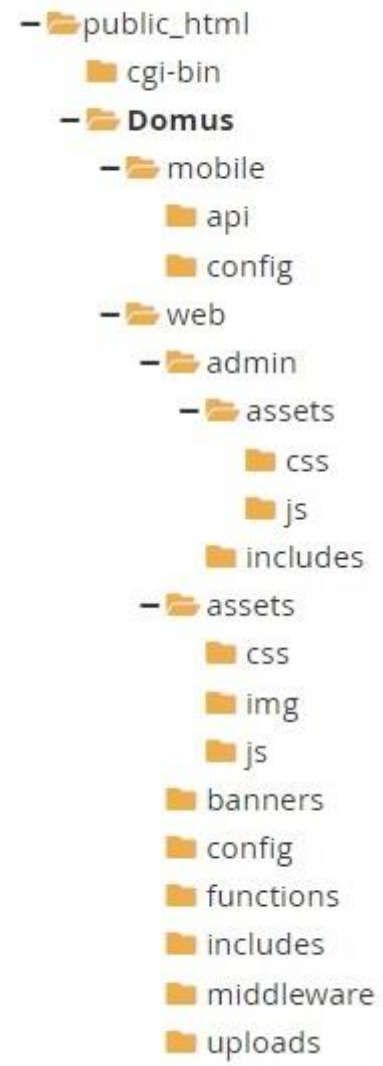
Prema definiciji, upravitelj datoteka (File Manager) je »softver koji se koristi za organiziranje datoteka na uređaju za pohranu. Upravitelj datoteka prikazuje hijerarhiju datoteka/mapa i pruža funkcije za stvaranje, kopiranje, premještanje, preimenovanje, i brisanje mapa kao i kopiranje, premještanje, preimenovanje i brisanje datoteka.«[12] Kada se pogleda na razini upravitelja datoteka, i ovaj cjelokupni projekt je skupina mapa i datoteka od kojih one koje se tiču web aplikacije i pozadinskog sučelja mobilne napisanog u programskom jeziku PHP smješteni u upravitelju datoteka na HostGator serveru, te se njima može pristupiti preko osnovne domene, te puta koji vodi do određene datoteke.

URL (Uniform Resource Locator) adrese se tako koriste ne samo za pozivanje osnovne domene web stranice/aplikacije, već i određene mape i datoteke unutar upravitelja datoteka koji je pohranjen na mrežnom serveru.

Slika 14. Dijelovi URL adrese naslovne stranice web aplikacije

Kako bi mape i njihove stavke bile javne, odnosno kako bi im se moglo javno pristupiti preko URL adrese, potrebno je da budu spremljene u izvornu mapu zvanu *public_html*, gdje je moguće slobodno organizirati mape i datoteke. U toj izvornoj mapi je stvorena mapa *Domus*, koju možemo nazvati središnjom mapom ovog projekta, odnosno dijela projekta čije su stavke pohranjene na online serveru. Unutar te mape su stvorene dvije mape *web* i *mobile*, kako bismo mogli razdijeliti daljnje mape i datoteke, ovisno o tome za rad koje aplikacije su one potrebne. Važno je naglasiti da je web aplikacije u potpunosti, odnosno sve njezine stavke su spremljene unutar mape *web*, dok mapa *mobile* sadrži samo potrebne skripte napisane u PHP-u koje mobilna aplikacija koristi kako bi uspostavila kontakt s bazom podataka. To će biti veoma važno imati na umu prilikom implementacije aplikacija.

[12] PCMag, "PCMag Encyclopedia, File Manager", pcmag.com/encyclopedia/term/file-manager



Slika 15. Organizacija mapa u upravitelju datoteka (File Manager)

3.4. Uspostavljanje veze s bazom podataka

S obzirom na to da je web aplikacija u potpunosti pohranjena unutar mape *web*, samim time ta mapa sadrži mnogo više pod-mapa i datoteka koje su potrebne za rad web aplikacije. S druge strane, mapa *mobile* sadrži samo potrebne skripte koje vrše potrebne akcije nad bazom podataka kada to bude potrebno. Međutim, ono što je zajedničko mapama *web* i *mobile* je da obje sadrže posebnu mapu *config* (skraćeno od riječi configuration, odnosno konfiguracija) u kojoj se nalazi samo jedna datoteka; *connection.php.* To je najosnovnija datoteka u kojoj je u PHP-u zadana konfiguracija bez koje nije moguće uspostaviti vezu s bazom podataka.

Iako je teoretski moguće pisati kompletni potrebni kod za uspostavljanje veze s bazom u svaku datoteku zasebno, mnogo je jednostavnije napisati taj kod samo jednom u zasebnu skriptu koju ćemo kasnije pozivati u ostalim datotekama korištenjem naredbi *include* ili *require*, koje »preuzimaju sav tekst, kod, i/ili oznake koje postoje u navedenoj datoteci i kopiraju ih u datoteku koja koristi tu naredbu.«[13]

U toj osnovnoj datoteci *connection.php* piše se skripta u kojoj se deklariraju četiri varijable; *servername, username, password,* i *database.* Kao ime servera se deklarira tzv. *localhost,* koji se odnosi na »lokalno računalo na kojem se izvodi program.«[14] Krucijalno je da baza podataka i PHP skripta koja uspostavlja vezu s tom bazom budu na istoj toj lokalnoj mašini. U ovom slučaju govorimo o HostGator serveru.

Potom se pod korisničko ime (varijabla *username*) zadaje korisničko ime MySQL korisnika kojeg smo dodali bazi podataka te njemu pripadajuću lozinku zadajemo kao vrijednost varijabli *password.* Potom još varijabli *database* dodajemo ime baze podataka s kojom će se ostvariti povezanost. Kada su te četiri glavne varijable pravilno deklarirane, deklariramo varijablu *con* (skraćeno od connection) kojoj zadajemo funkciju *mysqli_connect* koja će na osnovu prethodno deklariranih varijabli ostvariti vezu s MySQL serverom. Tu funkciju je bilo važno spremiti u zasebnu varijablu koju će biti potrebno pozivati u ostalim skriptama koje će naredbom *include* preuzimati varijable i naredbe deklarirane u ovoj skripti.

[13] W3Schools, "PHP Include Files", w3schools.com/php/php_includes.asp
[14] TechTerms, "Localhost", techterms.com/definition/localhost

```php
<?php
    $servername = "localhost";
    $username = "abramski_dbadmin";
    $password = "nQjMeqdE9Au";
    $database = "abramski_domus";

    $con = mysqli_connect($servername,$username,$password,$database);
?>
```

Slika 16. PHP skripta koja otvara vezu s MySQL serverom

Prilikom prvog pisanja ove PHP skripte, koristila se i *if...else* naredba koja bi vratila odgovarajuću poruku ovisno o tome je li veza s MySQL serverom uspješno ili neuspješno ostvarena. Time se vršila provjera. Jednom kada smo bili sigurni da se veza s bazom podataka uspostavlja besprijekorno, naredbu smo mogli obrisati jer nema potrebe da u samostalnom radu ova skripta vraća ikakvu poruku nakon uspješne uspostave veze. Iako se mogla jedna skripta koristiti i pozivati za skripte unutar i *mobile* i *web* mape, zbog praktičnih razloga su stvorene dvije iste skripte koje su spremljene dvije mape *config*, od kojih se jedna nalazi unutar mape *web,* a druga unutar mape *mobile.*

Informacije radi, ova, kao i za sve PHP skripte koje su pisane kao dio ovog projekta, su pisane na proceduralni, a ne objektno-orijentirani način u programskom jeziku PHP.

4. IMPLEMENTACIJA MOBILNE APLIKACIJE

»Android Studio službeno je integrirano razvojno okruženje (IDE) za razvoj Android aplikacija. Temelji se na IntelliJ IDEA, Java-integriranom razvojnom okruženju za softver koji uključuje alate za uređivanje koda i razvoj programa.«[15] Kompletno korisničko sučelje i programerski dio mobilne aplikacije u programskom jeziku Java je razvijen u Android Studiju. Početak rada započinje otvaranjem novog projekta koji odmah u startu otvara zadanu XML datoteku u kojoj se razvija korisničko sučelje glavne aktivnosti (activity_main.xml), kao i klasa te glavne aktivnosti (MainActivity).

4.1. Implementirane ovisnosti (dependencies)

Za uspješno funkcioniranje mobilne aplikacije je prilikom razvoja projekta bilo potrebno implementirati potrebne ovisnosti (dependencies) u datoteci build.gradle. Prva dodana ovisnost zvana je Glide[16], koja omogućuje učitavanje i prikazivanje fotografija u korisničkom sučelju aplikacije.

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    //Loading images from internet
    implementation 'com.github.bumptech.glide:glide:4.13.2'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.13.2'

    //Search Bar
    implementation 'com.github.mancj:MaterialSearchBar:0.8.5'

    //Slider
    // Material Components for Android. Replace the version with the latest version of Material Components library.
    implementation 'com.google.android.material:material:1.5.0'
    // Circle Indicator (To fix the xml preview "Missing classes" error)
    implementation 'me.relex:circleindicator:2.1.6'
    implementation 'org.imaginativeworld.whynotimagecarousel:whynotimagecarousel:2.1.0'

    //Volley
    implementation 'com.android.volley:volley:1.2.1'

    //TinyCart
    implementation 'com.github.hishd:TinyCart:1.0.1'
}
```

Slika 17. Ovisnosti (dependencies) implementirane u build.gradle (:app)

[15] TechTarget Contributor, "Android Studio", techtarget.com/searchmobilecomputing/definition/Android-Studio
[16] Glide, github.com/bumptech/glide

Druga implementirana ovisnost je MaterialSearchBar[17] koji omogućuje implementiranje tražilice, dok se iduće tri ovisnosti zajednički koriste za omogućavanje implementiranja Why Not! Image Carousel[18], koji omogućuje korištenje klizača (slider) koji će prikazivati naslovne fotografije u korisničkom sučelju glavne aktivnosti. Konačne dvije, možda i ključne ovisnosti su Google-ov Volley[19], kao i već spomenuti TinyCart[20]. »Volley je HTTP biblioteka koja olakšava i, što je još važnije, ubrzava umrežavanje Android aplikacije«[21], dok TinyCart, kako je i spomenuto ranije, omogućava baratanje s košaricom unutar mobilne aplikacije.

4.2. Splash aktivnost (SplashActivity)

Prilikom prvog postavljanja projekta mobilne aplikacije u programu Android Studio, zadane postavke u datoteci *AndroidManifest.xml* nalažu da prva aktivnost koja se prikazuje bude glavna aktivnost.

```
<activity
    android:name=".activities.MainActivity"
    android:exported="true" />
<activity
    android:name=".activities.SplashActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Slika 18. Postavljanje Splash aktivnosti kao početne aktivnosti

Međutim, iako je glavna aktivnost po hijerarhiji najvažnija od svih aktivnosti u aplikaciji (jer bez nje sve ostale nemaju gotovo nikakvog značaja), radi veće vizualne atraktivnosti aplikacije je stvorena tzv. Splash aktivnost koja prilikom ulaska u aplikaciju na kratko vrijeme prikazuje logotip aplikacije. Može se reći da je Splash svojevrsni uvod u sam početak rada s aplikacijom.

Implementacija aktivnosti Splash najjednostavnija je od svih aktivnosti u ovom projektu. XML datoteka *activity_main* sastoji se od samo jedne zadane slike (u ovom slučaju govorimo o logotipu).

[17] MaterialSearchBar, github.com/mancj/MaterialSearchBar
[18] Why Not! Image Carousel, github.com/ImaginativeShohag/Why-Not-Image-Carousel
[19] Volley, github.com/google/volley
[20] TinyCart, github.com/hishd/TinyCart
[21] Google, "Volley", google.github.io/volley

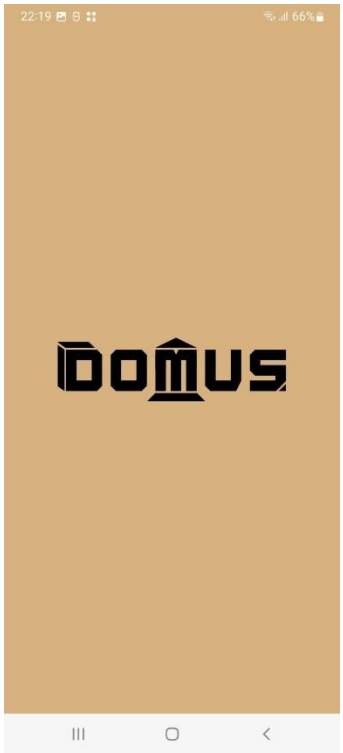
```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/domus"
    tools:context=".activities.SplashActivity">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/logo" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Slika 19. Datoteka *activity_splash.xml*

```java
package com.example.domusmobile2.activities;

import ...

public class SplashActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                startActivity(new Intent( packageContext: SplashActivity.this, MainActivity.class));
                finish();
            }
        }, delayMillis: 2000);
    }
}
```

Slika 20. Implementacija klase SplashActivity.java

Važno je znati da prilikom deklariranja nove aktivnosti je nužno omogućiti i stvaranje njoj pripadajuće klase. U slučaju aktivnosti Splash govorimo o klasi SplashActivity.java, u kojoj se deklarira jednostavna naredba koja će omogućiti prikaz aktivnosti na kratki trenutak prilikom ulaska u aplikaciju i potom automatski preko namjere (Intent) preusmjeriti korisnika na glavnu aktivnost aplikacije. »Intent (namjera) je apstraktni opis operacije koju treba izvesti. Može se koristiti za pokretanje aktivnosti s neke početne aktivnosti na drugu željenu aktivnost.«[22]



Slika 21. Prikaz aktivnosti Splash u mobilnoj aplikaciji

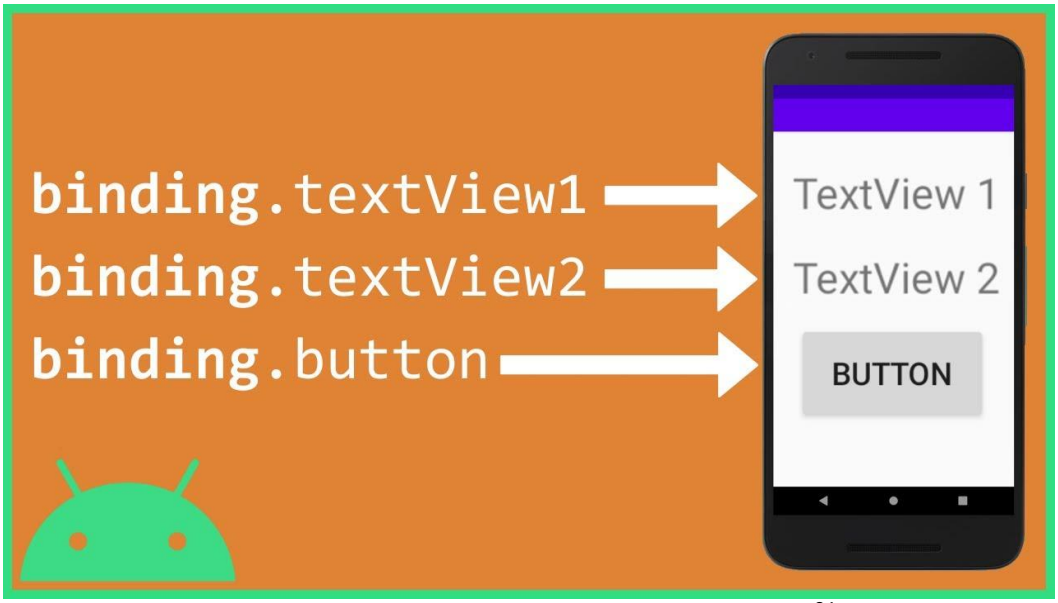
[22] Android Developers, "Intent", developer.android.com/reference/android/content/Intent

Aktivnost Splash ne zadržava se dugo prilikom ulaska u aplikaciju, i odmah nas preusmjerava prema glavnog aktivnosti, koja je najkompleksnija od svih.

4.3. Glavna aktivnost (MainActivity)

Uz izuzetak već spomenute Splash aktivnosti, implementacija svih aktivnosti u mobilnoj aplikaciji započinje deklariranjem varijabli potrebnih za uspješnu implementaciju aktivnosti.

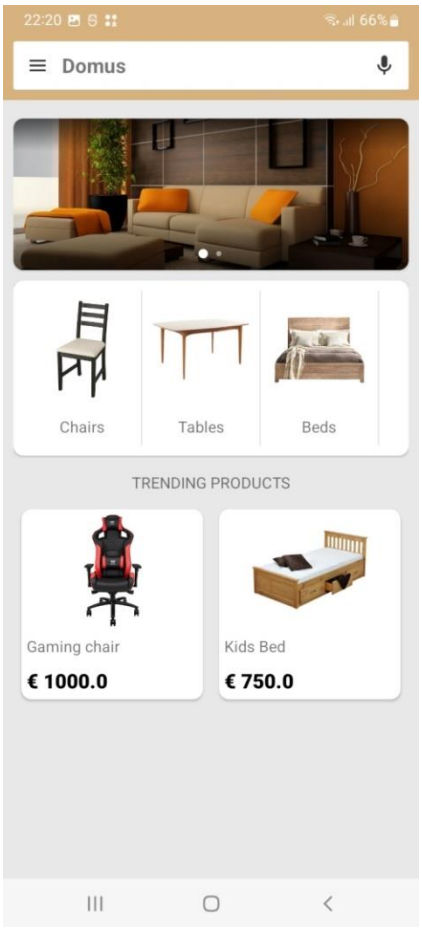
Najosnovnija varijabla je varijabla *binding* koja uveliko olakšava pisanje koda jer zamjenjuje još uvijek standardni *findViewById*. »Povezivanje prikaza (view binding) je značajka koja omogućuje lakše pisanje koda koji je u interakciji s pogledima (view). Jednom kada je view binding omogućen u modulu, on generira klasu vezivanja za svaku XML datoteku rasporeda (layout) prisutnu u tom modulu. Instanca te klase sadrži izravne reference na sve poglede koji imaju odgovarajući identifikacijski broj u XML datoteci.«[23]



Slika 22. Ilustrirano objašnjenje view bindinga[24]

Drugim riječima, kada koristimo view binding nije potrebno "ručno" povezivati određene dijelove korisničkog sučelja (poglede na tekst, slike, i sl.) u aktivnosti korištenjem *findViewById*, bitno je samo određenom pogledu u korisničkom sučelju dati jedinstvenu identifikaciju kojom će ga aktivnost prepoznati kako bi znala što s njime treba učiniti, odnosno što treba prikazati u tom pogledu.

[23] Android Developers, "View Binding", developer.android.com/topic/libraries/view-binding
[24] Florian, Walther, Coding in Flow, "View Binding – Getting Started + Differences – Android Studio Tutorial", youtube.com/c/CodinginFlow



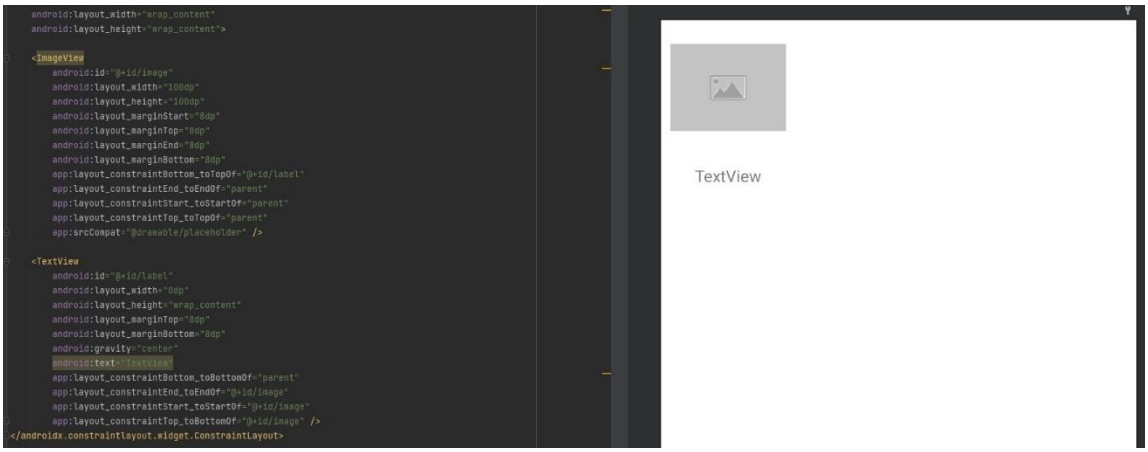
Slika 23. Prikaz glavne aktivnosti u mobilnoj aplikaciji

Uz varijablu *binding* koja će i kroz ostale aktivnosti postati neizostavni dio kompletne mobilne aplikacije, u glavnoj aktivnosti su još deklarirane potrebne varijable koje služe kao poveznica s modelima i adapterima kategorija i proizvoda, budući da su obje te stavke potrebne za prikazati u glavnoj aktivnosti. Iako je, kada se pogleda, najgornja stavka glavne aktivnosti tražilica, njezina metode su programirane naknadno.

Umjesto toga, prilikom implementacije glavne aktivnosti prvo su napisane metode koje služe za inicijalizaciju (*initBanners*) i dohvat (*getBanners*) naslovnih fotografija koje će se prikazati u karuselu (ili klizaču), za čije je korištenje prethodno implementirana ovisnost Why Not! Image Carousel. S obzirom na to da, za razliku od kategorija i proizvoda, za naslovne fotografije nije potrebno imati modele i adaptere, onda je i samim time programiranje njihove zadaće u glavnoj aktivnosti najjednostavnije. Inicijalizacija se provodi u metodi *initBanners*, koja se poziva unutar glavne metode *onCreate* (zaštićena zadana metoda koja se automatski pokreće prilikom svakog pokretanja aktivnosti). Unutar metode *initBanners* se poziva metoda *getBanners* u kojoj se koristi Volley, s kojim se piše zahtjev za dohvat naslovnih fotografija iz baze podataka. Zahtjev dohvaća naslovne fotografije koje su spremljene u obliku JSON niza (array), i potom se preko *for* petlje svaki objekt (odnosno naslovna fotografija) u tom nizu smješta u taj karusel koji će ju prikazati u korisničkom sučelju.

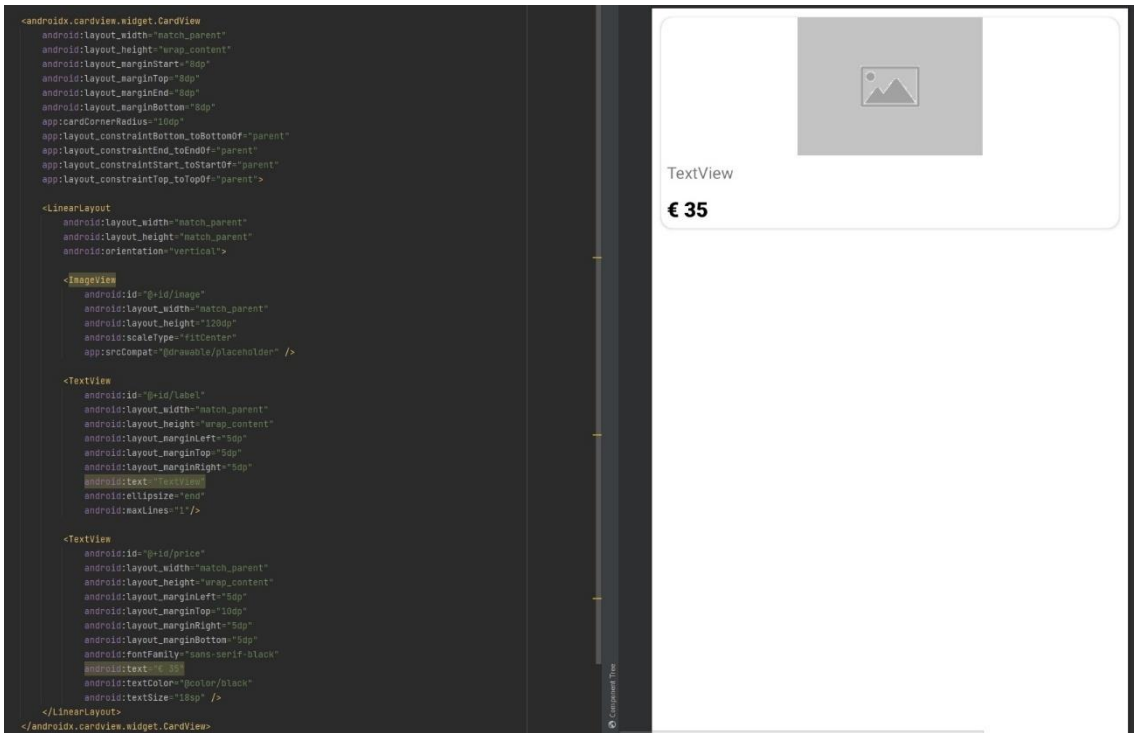
```
void initBanners() {
    getBanners();
}

void getBanners(){
    RequestQueue queue = Volley.newRequestQueue( context: this);

    StringRequest request = new StringRequest(Request.Method.GET, Api.GET_BANNERS, response -> {
        try {
            JSONObject object = new JSONObject(response);

            JSONArray bannersArray = object.getJSONArray( name: "banners");
            for(int i=0; i<bannersArray.length(); i++){
                JSONObject childObj = bannersArray.getJSONObject(i);
                binding.carousel.addData(
                        new CarouselItem(
                                imageUrl: Api.GET_BANNER_IMG + childObj.getString( name: "image")
                        )
                );
            }

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }, error -> {});

    queue.add(request);
}
```
Slika 24. Metode za inicijalizaciju i dohvat naslovnih fotografija

Sljedeća stavka u glavnoj aktivnosti su kategorije i proizvodi, koje se kao i naslovne fotografije inicijaliziraju i dohvaćaju iz baze podataka, ali uz neke ključne razlike. Prva je razlika da je za prikaz kategorija i proizvoda bilo potrebno stvoriti dvije dodatne XML datoteke (*item_categories.xml* i *item_products.xml*) u kojima se realizira kako će izgledati prvi red prikaza kategorija ili proizvoda u korisničkom sučelju.



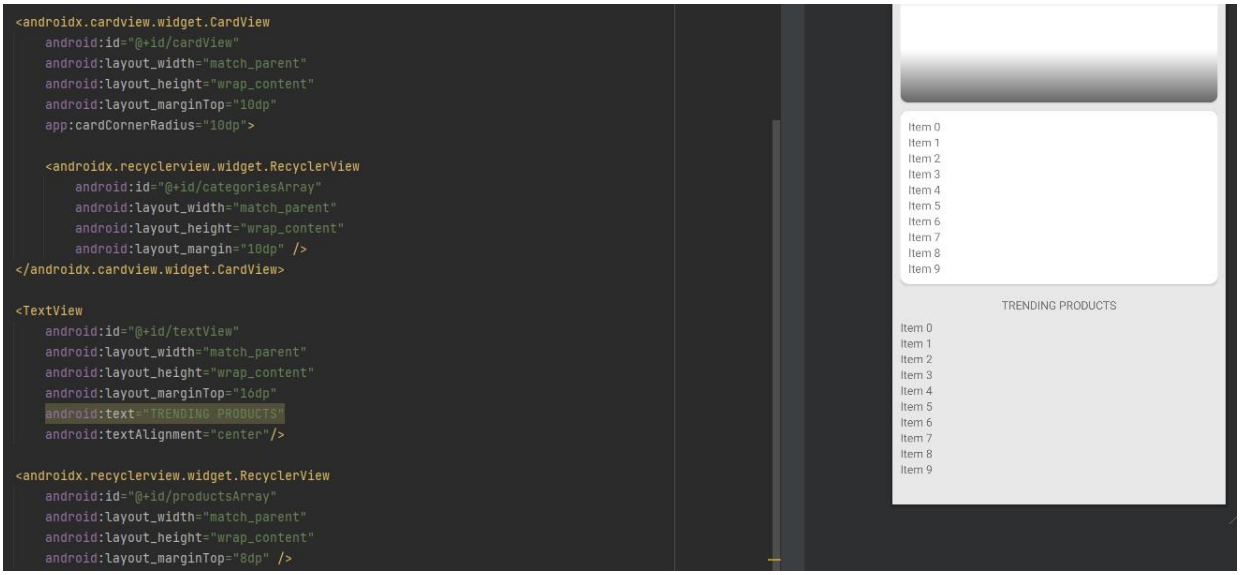
Slika 25. Prvi red prikaza kategorije u datoteci *item_categories.xml*

Dok se prvi red prikaza kategorije sastoji samo od pogleda za fotografiju kategorije i imena, prvi red prikaza proizvoda dizajniran je u tzv. kartičnom pogledu (CardView) koji se sastoji od fotografije, opisa, i cijene proizvoda. Na mjestu fotografija su postavljeni tzv. *placeholderi* koji će prilikom pokretanja aplikacije biti zamijenjeni odgovarajućim fotografijama kategorija i proizvoda.



Slika 26. Prvi red prikaza proizvoda u datoteci *item_products.xml*

Potom se stavke tih dviju datoteka postavljaju u RecyclerView koji se nalazi u XML datoteci glavne aktivnosti.



Slika 27. RecylerView-ovi za kategorije i proizvode u datoteci *activity_main.xml*

Sljedeća razlika u odnosu na naslovne fotografije je ta da metoda za inicijalizaciju koristi modele i adaptere. Deklarira se novi niz koji će sadržavati stavke ovisne o *get* i *set* metodama koje su napisane u modelu kategorija te se deklarira novi adapter koji će biti zadužen da prikaže sve promjene u korisničkom sučelju nakon što se inicijalizacija obavi. Inicijalizacija poziva metodu dohvaćanja kategorija (*getCategories*) koja ponovno koristi Volley preko kojeg se piše zahtjev za dohvat kategorija i njihovih stavki. Kategorije se dohvaćaju ponovno u obliku JSON niza te se preko *for* petlje svaki od objekata u tom nizu sprema zasebno dokle god je taj niz dug. Potom se preko adaptera poziva javna metoda *notifyDataSetChanged* koja »obavještava priložene promatrače da su temeljni podaci promijenjeni i svaki pogled koji održava skup podataka trebao bi se sam osvježiti.«[25] Jednom kada se obavi dohvat fotografija, u metodi za inicijalizaciju je također zadan LinearLayoutManager, javni konstruktor koji daje funkcionalnost određenom RecyclerView-u u korisničkom sučelju. U ovom slučaju, govorimo o RecylerView-u koji nosi ime "categoriesArray" (u slučaju proizvoda "productsArray", i shodno tome se i koristi view binding kako bi aktivnost znala u kojoj stavci korisničkog sučelja mora prikazati dobiveni rezultat.

[25] Android Developers, "BaseAdapter", developer.android.com/reference/android/widget/BaseAdapter

```
void initCategories(){
    categoryModels = new ArrayList<>();
    categoryAdapter = new CategoryAdapter( context: this, categoryModels);

    getCategories();

    LinearLayoutManager layoutManager = new LinearLayoutManager( context: this);
    layoutManager.setOrientation(RecyclerView.HORIZONTAL);
    DividerItemDecoration itemDecoration = new DividerItemDecoration( context: this, layoutManager.getOrientation());
    binding.categoriesArray.setLayoutManager(layoutManager);
    binding.categoriesArray.addItemDecoration(itemDecoration);
    binding.categoriesArray.setAdapter(categoryAdapter);
}

void getCategories(){
    RequestQueue queue = Volley.newRequestQueue( context: this);

    StringRequest request = new StringRequest(Request.Method.GET, Api.GET_CATEGORIES, response -> {
        try {
            JSONObject mainObj = new JSONObject(response);

            JSONArray categoriesArray = mainObj.getJSONArray( name: "categories");
            for(int i=0; i<categoriesArray.length(); i++){
                JSONObject object = categoriesArray.getJSONObject(i);
                CategoryModel categoryModel = new CategoryModel(
                        object.getString( name: "name"),
                        image: Api.GET_UPLOADS + object.getString( name: "image"),
                        object.getInt( name: "id")
                );
                categoryModels.add(categoryModel);
            }
            categoryAdapter.notifyDataSetChanged();
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }, error -> {

    });

    queue.add(request);
}
```

Slika 28. Metode za inicijalizaciju i dohvat kategorija

Posljednje dvije metode koje se nalaze u glavnoj aktivnosti su metode za inicijalizaciju i dohvat proizvoda koji se prikazuju u glavnoj aktivnosti. Za razliku od kategorija, koje se sve prikazuju na glavnoj aktivnosti, proizvodi se u pravilu svi filtriraju po tim kategorijama te se odabirom određene kategorije se zapravo prikazuju svi proizvodi te kategorije. No, koji onda proizvodi se prikazuju odmah u glavnoj aktivnosti? Ideja je da se u glavnoj aktivnosti korisniku odmah prikažu proizvodi koji su trenutno najpopularniji, odnosno "u trendu". Odluku hoće li neki proizvod biti u trendu je na administratoru prilikom unošenja ili izmjene podataka o proizvodu preko web aplikacije.

Metoda za inicijalizaciju proizvoda u glavnoj aktivnosti radi na gotovo identičan način kao i metoda u slučaju kategorija, uz razliku da se ovdje ne koristi javni konstruktor LinearLayoutManager, koji je zadužen za linearno prikazivanje stavki u korisničkom sučelju, već se koristi GridLayoutManager koji omogućuje prikaz stavki u obliku rešetke.

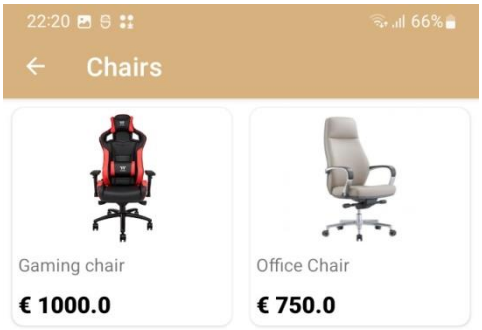
Važno je zadati broj koliko se stavki može prikazati u istom redu. Postavljeno je da idu po dvije stavke. Metoda za dohvat popularnih proizvoda u glavnoj aktivnosti funkcionira na identičan način kao i metoda za dohvat kategorija, s tim da se tu radi o više podataka vezanima za proizvod. Dok se u slučaju kategorije dohvaćaju identifikacijski broj, ime i fotografije, u slučaju se pored te tri važne stavke uzimaju i cijena te količina, koje su jednako važne. Kao i u metodi za dohvat kategorija, i u metodi za dohvat proizvoda se koristi adapter odrađuje sve potrebne promjene u korisničkom sučelju preko javne metode *notifyDataSetChanged*.

```
void initProducts(){
    productModels = new ArrayList<>();
    productAdapter = new ProductAdapter( context: this, productModels);

    getTrendingProducts();

    GridLayoutManager layoutManager = new GridLayoutManager( context: this, spanCount: 2);
    binding.productsArray.setLayoutManager(layoutManager);
    binding.productsArray.setAdapter(productAdapter);
}

void getTrendingProducts() {
    RequestQueue queue = Volley.newRequestQueue( context: this);

    StringRequest request = new StringRequest(Request.Method.GET, Api.GET_TRENDING_PRODUCTS, response -> {
        try {
            JSONObject mainObj = new JSONObject(response);

            JSONArray productsArray = mainObj.getJSONArray( name: "products");
            for(int i=0; i<productsArray.length(); i++){
                JSONObject childObj = productsArray.getJSONObject(i);
                ProductModel productModel = new ProductModel(
                        childObj.getString( name: "name"),
                         image: Api.GET_UPLOADS + childObj.getString( name: "image"),
                        childObj.getDouble( name: "selling_price"),
                        childObj.getInt( name: "quantity"),
                        childObj.getInt( name: "id")
                );
                productModels.add(productModel);
            }
            productAdapter.notifyDataSetChanged();
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }, error -> {

    });

    queue.add(request);
}
```

Slika 29. Metode za inicijalizaciju i dohvat proizvoda u glavnoj aktivnosti

4.4. Aktivnost kategorije (CategoryActivity)

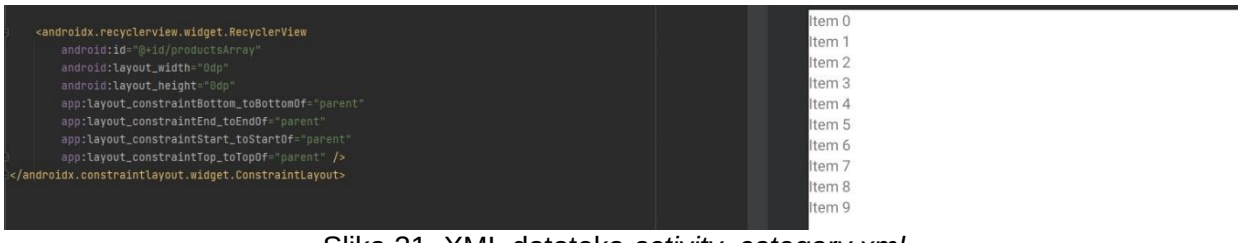
Ako korisnik mobilne aplikacije želi pretražiti proizvode po kategorijama, tada ima mogućnost odabrati neku od ponuđenih kategorija koje se prikazuju u glavnoj aktivnosti.



Slika 30. Aktivnost kategorije prikazuje proizvode u kategoriji stolice (chairs)

Po odabranoj kategoriji, pokreće se aktivnost kategorije i korisnik dolazi u korisničko sučelje koje je dizajnirano u XML datoteci *activity_category.xml* koja u sebi ima samo stavku RecyclerView u kojoj će se prikazati svi proizvodi koji pripadaju odabranoj kategoriji. Shodno tome, prilikom odabira kategorije dohvaćaju se ime i identifikacijski broj kategorije, nakon kojeg se ime kategorije prikazuje u naslovnoj traci, dok prema identifikacijskom broju aktivnost razaznaje o kojoj se kategoriji radi i koje onda proizvode treba dohvatiti i prikazati korisniku.

Aktivnost kategorije ne sadrži metodu za inicijalizaciju s obzirom na to da značajke koje su se u glavnoj aktivnosti tamo nalazile sada smještene u zaštićenoj metodi *onCreate.* No, aktivnost kategorije i dalje sadrži metodu za dohvat proizvoda koja funkcionira na istoimena metoda u glavnoj aktivnosti, uz ključnu razliku da je ovdje zadan parametar identifikacijskog broja odabrane kategorije kako bi metoda znala o kojoj se kategoriji radi i kako bi Volley stvorio zahtjev za dohvat proizvoda odgovarajuće kategorije.



Slika 31. XML datoteka *activity_category.xml*

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityCategoryBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());


    productModels = new ArrayList<>();
    productAdapter = new ProductAdapter( context: this, productModels);

    int categoryId = getIntent().getIntExtra( name: "categoryId",  defaultValue: 0);
    String categoryName = getIntent().getStringExtra( name: "categoryName");

    getSupportActionBar().setTitle(categoryName);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    getProducts(categoryId);

    GridLayoutManager layoutManager = new GridLayoutManager( context: this, spanCount: 2);
    binding.productsArray.setLayoutManager(layoutManager);
    binding.productsArray.setAdapter(productAdapter);
}

@Override
public boolean onSupportNavigateUp() {
    finish();
    return super.onSupportNavigateUp();
}

void getProducts(int categoryId) {
    RequestQueue queue = Volley.newRequestQueue( context: this);

    String url = Api.GET_PRODUCTS + categoryId;

    StringRequest request = new StringRequest(Request.Method.GET, url, response -> {
        try {
            JSONObject mainObj = new JSONObject(response);

            JSONArray productsArray = mainObj.getJSONArray( name: "products");
            for(int i=0; i<productsArray.length(); i++){
                JSONObject childObj = productsArray.getJSONObject(i);
                ProductModel productModel = new ProductModel(
                        childObj.getString( name: "name"),
                         image: Api.GET_UPLOADS + childObj.getString( name: "image"),
                        childObj.getDouble( name: "selling_price"),
                        childObj.getInt( name: "quantity"),
                        childObj.getInt( name: "id")
                );
                productModels.add(productModel);
            }
            productAdapter.notifyDataSetChanged();
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }, error -> {

    });
```

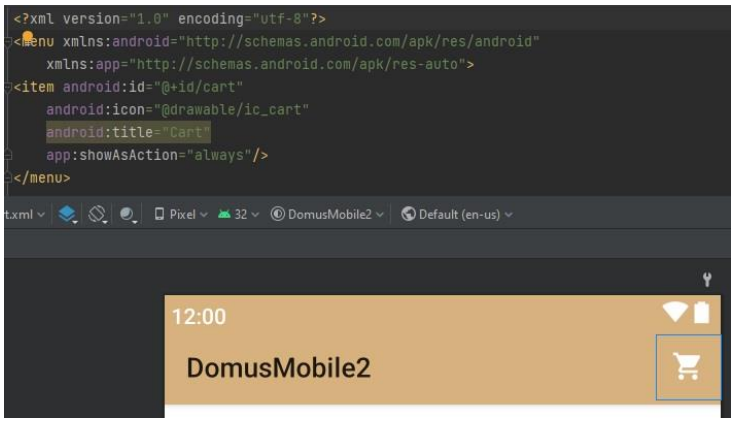
Slika 32. Metode u aktivnosti kategorije (CategoryActivity)

4.5. Aktivnost proizvoda (ProductDetailsActivity)

Metode za dohvat proizvoda koje su se koristile u glavnoj aktivnosti i aktivnosti kategorije služile su za dohvat više proizvoda i njihovih najosnovnijih značajki. No, u slučaju aktivnosti proizvoda (ProductDetailsActivity), sada je cilj skupiti detalje, odnosno informacije o samo jednom određenom proizvodu i prikazati ih u toj aktivnosti.

Slika 32. Aktivnost proizvoda prikazuje informacije o jednom određenom proizvodu

Prilikom odabira određenog proizvoda, nezavisno je li odabran među popularnim proizvodima u glavnoj aktivnosti ili s aktivnosti kategorije, otvara se posebna aktivnost u kojoj će se prikazati potrebne informacije o tom proizvodu. Ime se prikazuje u naslovnoj traci. Ispod fotografije se nalazi opis proizvoda i istaknuta cijena. U XML datoteci *activity_product_details.xml* je dizajnirano gdje će svaka od tih stavki biti smještena, uz gumb "Add to Cart" (dodaj u košaricu) koji će prikazani proizvod po korisnikovoj želji smjestiti u košaricu. Potrebno je još istaknuti i simbol košarice koji korisnika odvodi aktivnost košarice (CartActivity, stranica XY.) preko koje korisnik ostvaruje uvid u sadržaj košarice.



Slika 33. Datoteka *cart.xml*, spremljena u paketu *menu*

Kada korisnik odabere određeni proizvod u glavnoj aktivnosti ili aktivnosti kategorije, u adapteru proizvoda (ProductAdapter, stranica XY.) aktivira se *onClick* metoda koja sadrži namjeru (Intent, stranica 23.) koja će u ciljanu aktivnost (u ovom slučaju aktivnost proizvoda) poslati potrebne informacije koje ciljana aktivnost mora imati kako bi znala koji određeni proizvod mora prikazati. Tu nastupa zaštićena metoda *onCreate* koja će u aktivnosti proizvoda preuzeti te značajke i na osnovu identifikacijskog broja (ID) (koji će služiti kao parametar metodi *getProductDetails*) prikazati potrebne informacije o odabranom proizvodu.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityProductDetailsBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    String name = getIntent().getStringExtra( name: "name");
    String image = getIntent().getStringExtra( name: "image");
    int id = getIntent().getIntExtra( name: "id", defaultValue: 0);
    double price = getIntent().getDoubleExtra( name: "price", defaultValue: 0);

    Glide.with( activity: this).load(image).into(binding.productImage);

    getProductDetails(id);

    getSupportActionBar().setTitle(name);

    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    Cart cart = TinyCartHelper.getCart();

    binding.addToCartBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            cart.addItem(currentProduct,  qty: 1);
            binding.addToCartBtn.setEnabled(false);
            binding.addToCartBtn.setText("Added to cart");
        }
    });
}
```

Slika 34. Zaštićena metoda *onCreate* i javna metoda *onClick* u aktivnosti proizvoda

Metoda *onCreate* u aktivnosti proizvoda također sadrži i javnu *onClick* metodu koja daje funkcionalnost gumbu na čiji pritisak se odabrani proizvod pohranjuje u košaricu. Prilikom pohranjivanja proizvoda u košaricu, gumb će promijeniti boju i promijeniti svoj tekst u "Added to Cart" (dodano u košaricu) kako bi se korisniku dalo do znanja da je pohranjivanje u košaricu prošlu uspješno.

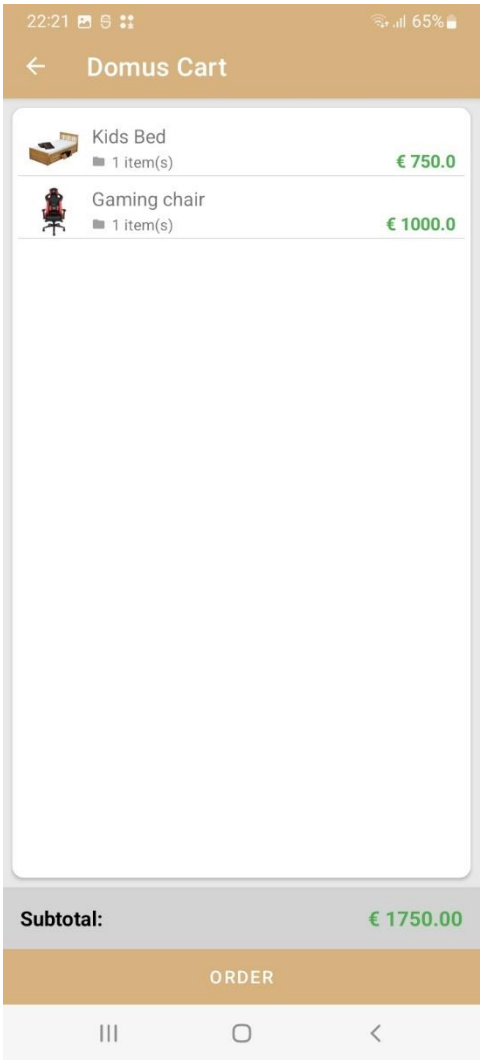
```
void getProductDetails(int id){
    RequestQueue queue = Volley.newRequestQueue( context: this);

    String url = Api.GET_PRODUCT_DETAILS + id;

    StringRequest request = new StringRequest(Request.Method.GET, url, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            try {
                JSONObject object = new JSONObject(response);

                JSONArray productDetailsArray = object.getJSONArray( name: "products");
                for(int i=0; i<productDetailsArray.length(); i++){
                    JSONObject product = productDetailsArray.getJSONObject(i);
                    String description = product.getString( name: "small_description");
                    String price = product.getString( name: "selling_price");
                    binding.productDes.setText(description);
                    binding.productPrice.setText("€ " + price);

                    currentProduct = new ProductModel(
                            product.getString( name: "name"),
                             image: Api.GET_UPLOADS + product.getString( name: "image"),
                            product.getDouble( name: "selling_price"),
                            product.getInt( name: "quantity"),
                            product.getInt( name: "id")
                    );
                }

            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    });

    queue.add(request);
}
```

Slika 35. Metoda za dohvat detalja o proizvodu

4.6. Aktivnost košarice (CartActivity)

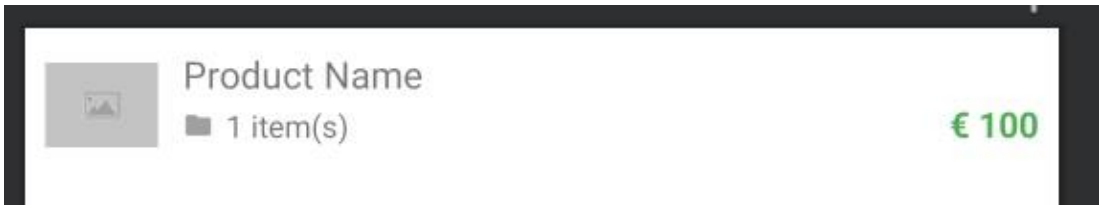
Aktivnost košarice pokreće se kada korisnik odlazi provjeriti sadržaj košarice pritiskom na sopstveni simbol u naslovnoj traci u aktivnosti proizvoda. U ovoj aktivnosti najveću ulogu

ima implementirana ovisnost TinyCart koja je zapravo biblioteka koja sadrži razne metode za upravljanje košaricom. U ovom slučaju su nam bitne metode *getAllItemsWithQty,* koja služi za dohvat svih proizvoda pohranjenih u košaricu s pripadajućom količinom, te metoda *getTotalPrice* koja izračunava ukupnu cijenu svih proizvoda u košarici.



Slika 36. Prikaz košarice u mobilnoj aplikaciji

Datoteka *activity_cart.xml* u kojoj je dizajnirano korisničko sučelje aktivnosti košarice većim dijelom se sastoji od Recycler View-a s identifikacijom "cartList", koju View Binding ponovno koristi kako bi znao u koje poglede treba smjestiti potrebne podatke, dok je prvi red prikaza određenog proizvoda u košarici dizajniran u datoteci *item_cart.xml,* u kojoj je određeno gdje se prikazuje slika, ime, količina, i cijena proizvoda u košarici.



Slika 37. Dizajn prvog reda proizvoda u košarici

Potom, u datoteci *activity_cart.xml,* ispod Recycler View-a je na dnu aktivnosti dizajnirano mjesto u kojem stoji ukupna cijena svih proizvoda u košarici, te gumb koji korisnika vodi prema sljedećemu korak u kompletiranju narudžbe.

Važno je za naglasiti da TinyCart metoda *getAllItemsWithQty* dohvaća sve proizvode u košarici preko Mape. »Mapa je objekt koji preslikava ključeve u vrijednosti, ne može sadržavati duple ključeve, i svaki ključ može mapirati najviše jednu vrijednost. Sučelje mape uključuje metode za osnovne i skupne operacije.«[26] Shodno tome, mapa se deklarira u for petlji i ključ se zadaje preko vrijednosti modela proizvoda.

```
for(Map.Entry<Item,Integer> item : cart.getAllItemsWithQty().entrySet()){
    ProductModel productModel = (ProductModel) item.getKey();
    int quantity = item.getValue();
    productModel.setQuantity(quantity);

    productModels.add(productModel);
}
```

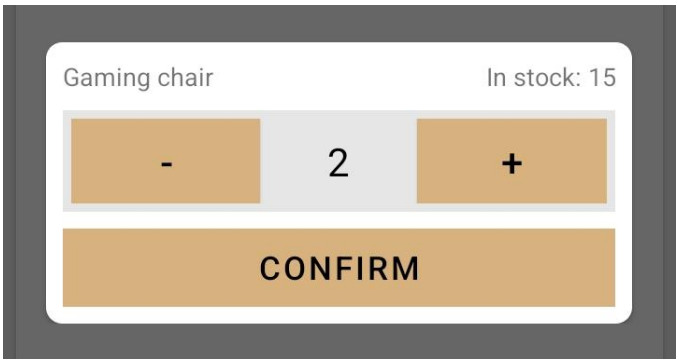
Slika 38. Korištenje for petlje koja koristi mapu i metodu *getAllItemsWithQty* u aktivnosti košarice

[26] Oracle, "The Map Interface", docs.oracle.com/javase/tutorial/collections/interfaces/map.html

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityCartBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    productModels = new ArrayList<>();

    Cart cart = TinyCartHelper.getCart();

    for(Map.Entry<Item,Integer> item : cart.getAllItemsWithQty().entrySet()){
        ProductModel productModel = (ProductModel) item.getKey();
        int quantity = item.getValue();
        productModel.setQuantity(quantity);

        productModels.add(productModel);
    }

    adapter = new CartAdapter( context: this, productModels, new CartAdapter.CartListener() {
        @Override
        public void onQuantityChanged() {
            binding.subtotal.setText(String.format("€ %.2f", cart.getTotalPrice()));
        }
    });

    LinearLayoutManager layoutManager = new LinearLayoutManager( context: this);
    DividerItemDecoration itemDecoration = new DividerItemDecoration( context: this, layoutManager.getOrientation());
    binding.cartList.setLayoutManager(layoutManager);
    binding.cartList.addItemDecoration(itemDecoration);
    binding.cartList.setAdapter(adapter);

    binding.subtotal.setText(String.format("€ %.2f", cart.getTotalPrice()));

    binding.orderBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            startActivity(new Intent( packageContext: CartActivity.this, CheckoutActivity.class));
        }
    });

    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
}
```

Slika 39. Metode u aktivnosti košarice

Važno je za naglasiti da je prilikom dodavanja proizvoda u košaricu svaki proizvod se automatski dodaje u košaricu sa zadanom količinom 1, no količina određenog proizvoda može se promijeniti pritiskom na željeni proizvod u košarici, čime se otvara posebni mali prozor koji je dizajniran u datoteci *quantity_window.xml* i koji omogućuje korisniku da promjeni količinu određenog proizvoda. Maksimalna količina koju korisnik može odabrati je koliko tog proizvoda ima na zalihi (administrator unosi podatke o zalihama preko web aplikacije). Prozor za promjenu količine funkcionira preko adaptera košarice (CartAdapter, stranica XY.). Prozor prikazuje ime proizvoda na kojem se vrši promjena količine, kao i koliko je trenutno proizvoda na zalihi i preko tog broja promjena količine ne može ići.

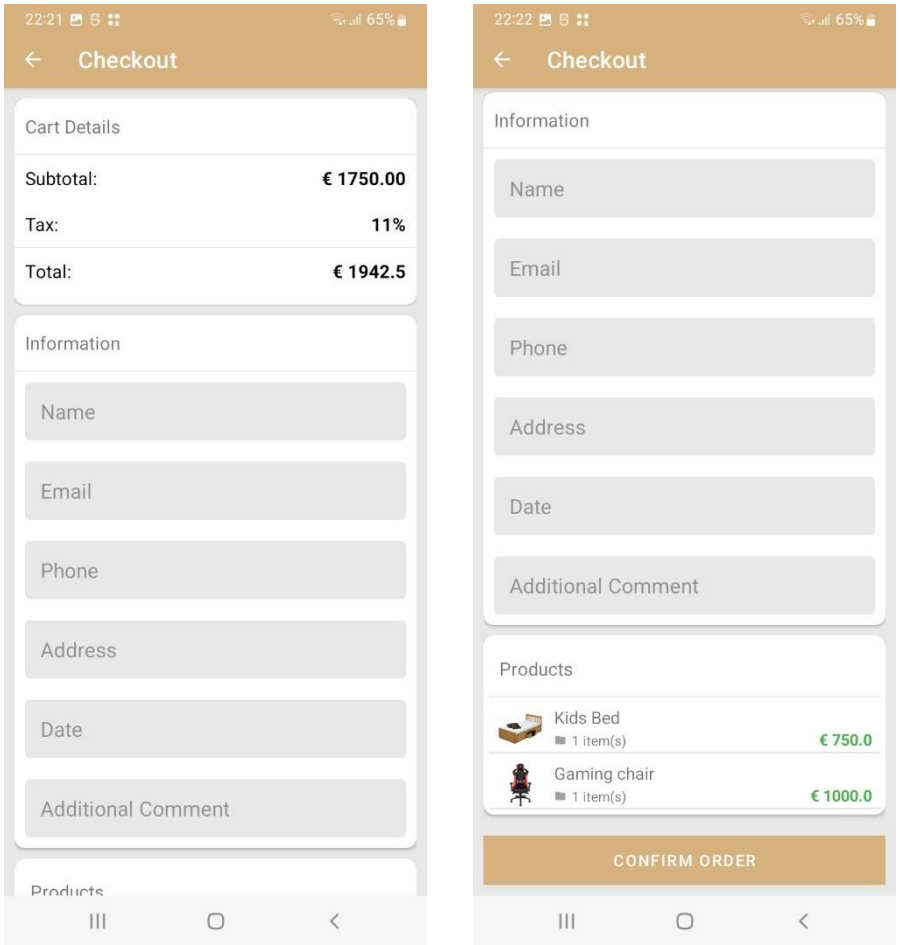


Slika 40. Prikaz prozora za promjenu količine proizvoda

Jednom kada se odabere željena količina, korisnik može potvrditi količinu i potom se u adapteru aktivira metoda *updateItem* koja je dio TinyCart biblioteke i automatski se ažurira i *getTotalPrice* jer se shodno promjenom količine ažurira i ukupna cijena. Na dnu aktivnosti košarica nalazi se gumb Order (naruči) koji vodi korisnika prema aktivnosti potvrde narudžbe.

4.7. Aktivnost završavanja narudžbe (CheckoutActivity)

Jednom kada korisnik odabere odgovarajuće proizvode i količinu svakog proizvoda, gumbom Order (naruči) prelazi u sljedeći i posljednji korak u stvaranju svoje narudžbe.



Slika 41 i 42. Aktivnost Završavanja Narudžbe (CheckoutActivity)

XML datoteka *activity_checkout.xml* sastoji se od tri dijela koji su raspoređeni u tri kartična prikaza (CardView). Prvi kartični prikaz prikazuje općenite detalje o narudžbi tj. ukupnu cijenu košarice, informativni dodatak o dodatnih 11% koji se stavlja na osnovnu cijenu i

na kraju konačna cijena narudžbe. U drugom kartičnom prikazu nalazi se forma u kojoj korisnik upisuje potrebne podatke (ime, email, telefonski broj, adresa, itd.) za završavanje narudžbe. U trećem kartičnom prikazu ponovno se nalazi popis proizvoda koje je korisnik odabrao kako bi korisnik prije konačne potvrde narudžbe mogao još jednom provjeriti proizvode koje namjerava naručiti. Treći kartični prikaz ponovno se sastoji samo od Recycler View-a koji funkcionira na identičan način kao i u aktivnosti košarice. Ukoliko je korisnik siguran u proizvode koje je odabrao i upisane podatke, pritiskom na gumb "Confirm Order" (potvrdi narudžbu) aktivira se metoda za potvrdu narudžbe (*confirmOrder*) u kojoj je programirano da svi podaci o korisniku i naručenim proizvodima spremaju u JSON niz koji će se preko PHP skripte na serveru kasnije dekodirati i spremiti sve podatke u odgovarajuće tablice u bazi podataka.

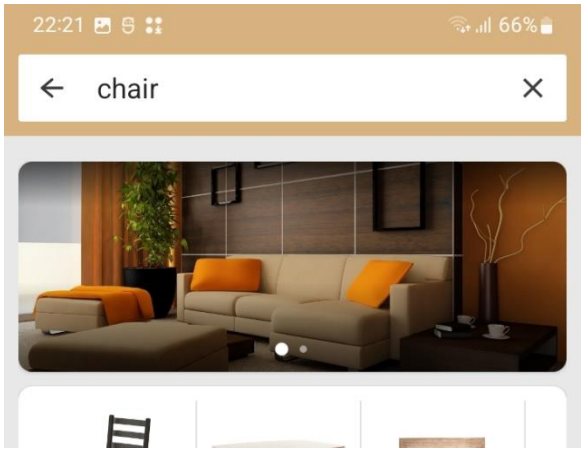
```
void confirmOrder() {
    progressDialog.show();
    RequestQueue queue = Volley.newRequestQueue( context: this);

    JSONObject productOrder = new JSONObject();
    JSONObject dataObject = new JSONObject();
    try {
        productOrder.put( name: "name", binding.nameBox.getText().toString());
        productOrder.put( name: "email", binding.emailBox.getText().toString());
        productOrder.put( name: "phone", binding.phoneBox.getText().toString());
        productOrder.put( name: "address", binding.addressBox.getText().toString());
        productOrder.put( name: "date", binding.dateBox.getText().toString());
        productOrder.put( name: "comment", binding.commentBox.getText().toString());
        productOrder.put( name: "tax", tax);
        productOrder.put( name: "total_price", totalPrice);

        JSONArray mobile_order_items = new JSONArray();
        for (Map.Entry<Item, Integer> item : cart.getAllItemsWithQty().entrySet()) {
            ProductModel productModel = (ProductModel) item.getKey();
            int quantity = item.getValue();
            productModel.setQuantity(quantity);

            JSONObject productObj = new JSONObject();
            productObj.put( name: "amount", quantity);
            productObj.put( name: "price_item", productModel.getPrice());
            productObj.put( name: "product_id", productModel.getId());
            productObj.put( name: "product_name", productModel.getName());
            mobile_order_items.put(productObj);
        }

        dataObject.put( name: "mobile_order", productOrder);
        dataObject.put( name: "mobile_order_items", mobile_order_items);

        //Log.e("err", dataObject.toString());

    } catch (JSONException e) {
    }

    JsonObjectRequest request = new JsonObjectRequest(Request.Method.POST, Api.POST_ORDER, dataObject, new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
            Toast.makeText( context: CheckoutActivity.this, text: "Order placed", Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        }
```

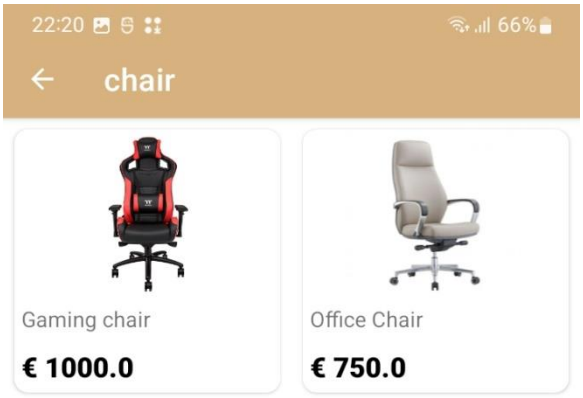
Slika 43. Metoda za potvrdu narudžbe

4.8. Aktivnost tražilice (SearchActivity)

Po završetku programiranja svih ranije spomenutih aktivnosti, dodatno je dodana i aktivnost pretraživanja ili tražilice, koja zapravo daje funkciju tražilici koja se nalazi u naslovnoj traci na vrhu glavne aktivnosti. Već je spomenuto da je za uspješnu implementaciju tražilice bila potrebna ovisnost (dependency) zvana MaterialSearchBar, a aktivnost tražilice (SearchActivity) se pobrinuo da tražilica funkcionira onako kako bi

trebala. XML datoteka activity_search.xml u sebi sadrži samo jedan Recycler View koji je zapravo istovjetan onom koji se koristio u aktivnosti kategorije. Drugim riječima, radi se o jednostavnom prikazu čija je zadaća da pokaže proizvode.





Slika 44. i 45. Korištenje tražilice u mobilnoj aplikaciji

Dok smo u aktivnosti kategorije proizvode filtrirali prema identifikacijom broju kategorije određenog proizvoda, sada se radi filtriranje po imenu. Ipak, kako se i dalje radi po vrlo sličnim načinima filtriranja proizvoda, koristila se načelno ista metoda za dohvat proizvoda kao i u aktivnosti kategorije, ali uz jednu malu, ali ključnu razliku

Kao i u aktivnosti kategorije, i u ovoj metodi dohvata proizvoda se piše zahtjev korištenjem Volleya koji će zatražiti JSON niz (koji stvara PHP skripta na HostGator serveru) koji sadrži proizvode koji su izabrani iz tablice u bazi podataka prema identifikacijskom broju te kategorije. U slučaju tražilice je vrlo slična situacija, samo umjesto identifikacijskog broja kategorija se ovaj put koristi upit (query) koji će zatražiti sve JSON niz sa svim proizvodima koji u svom imenu imaju riječ koja cjelokupno ili djelomično odgovara riječi u upitu.

```
void getProducts(String query) {
    RequestQueue queue = Volley.newRequestQueue( context: this);

    String url = Api.SEARCH_PRODUCTS + query;

    StringRequest request = new StringRequest(Request.Method.GET, url, response -> {
        try {
            JSONObject mainObj = new JSONObject(response);

            JSONArray productsArray = mainObj.getJSONArray( name: "products");
            for(int i=0; i<productsArray.length(); i++){
                JSONObject childObj = productsArray.getJSONObject(i);
                ProductModel productModel = new ProductModel(
                        childObj.getString( name: "name"),
                         image: Api.GET_UPLOADS + childObj.getString( name: "image"),
                        childObj.getDouble( name: "selling_price"),
                        childObj.getInt( name: "quantity"),
                        childObj.getInt( name: "id")
                );
                productModels.add(productModel);
            }
            productAdapter.notifyDataSetChanged();
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }, error -> {

    });

    queue.add(request);
}
```

Slika 46. Metoda za dohvat proizvoda u aktivnosti tražilice

4.9. Modeli

Kako je i prethodno navedeno, modeli adapteri su neizostavni dio ove mobilne aplikacije. Namjenu modela je relativno jednostavno za objasniti jer se u biti samo radi o javnim klasama koje sadrže javne metode koje postavljaju ili vraćaju određene vrijednosti neke varijable. Modeli i kategorija i proizvoda funkcioniraju na identičan način, uz iznimku da proizvodi imaju puno više entiteta pa je shodno tome program nešto duži. Oba modela se koriste kroz cijelu aplikaciju i pokreću se odmah prilikom ulaska u glavnu aktivnost, no model proizvoda sadrži i neke metode koje se koriste i korištenja prilikom aktivnosti košarice.

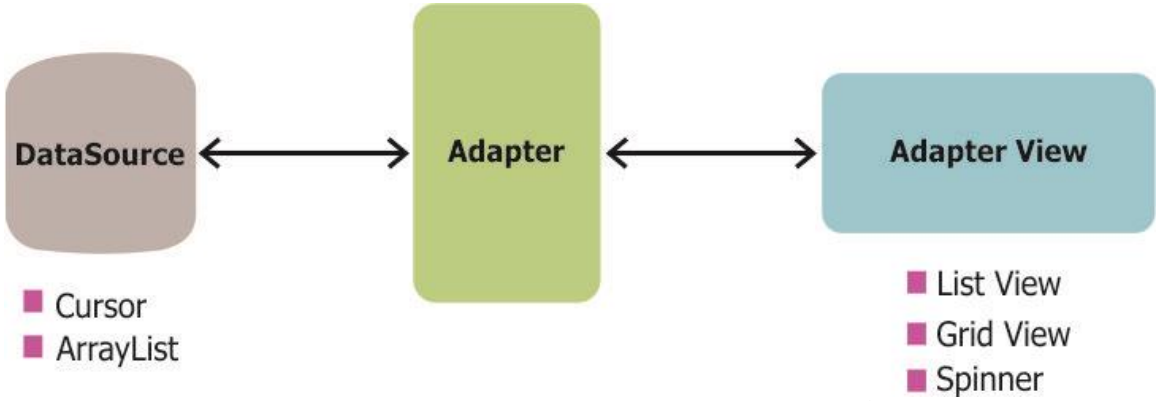
```java
public class CategoryModel {
    private String name, image;
    private int id;

    public CategoryModel(String name, String image, int id) {
        this.name = name;
        this.image = image;
        this.id = id;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getImage() { return image; }

    public void setImage(String image) { this.image = image; }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }
}
```

```java
public class ProductModel implements Item, Serializable {
    private String name, image;
    private double price;
    private int stock, id;
    private int quantity;

    public ProductModel(String name, String image, double price, int stock, int id) {
        this.name = name;
        this.image = image;
        this.price = price;
        this.stock = stock;
        this.id = id;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getImage() { return image; }

    public void setImage(String image) { this.image = image; }

    public double getPrice() { return price; }

    public void setPrice(double price) { this.price = price; }

    public int getStock() { return stock; }

    public void setStock(int stock) { this.stock = stock; }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    @Override
    public BigDecimal getItemPrice() { return new BigDecimal(price); }

    @Override
    public String getItemName() { return name; }

    public int getQuantity() { return quantity; }

    public void setQuantity(int quantity) { this.quantity = quantity; }
}
```

Slika 47. i 48. Implementacija modela kategorija i proizvoda

4.10. Adapteri

Već smo prethodno spomenuli da adapteri služe kao poveznica između izvora podataka i korisničkog sučelja koji će prikazati te podatke. Moblina aplikacija *Domus* broji ukupno tri adaptera; adapter kategorije (CategoryAdapter), adapter proizvoda (ProductAdapter), i adapter košarice (CartAdapter).



Slika 49. Grafičko objašnjenje uloge adaptera[27]

Ono što adapteri u biti praktično čine je koriste se odgovarajućim modelom (kategorija ili proizvoda) čime vrše dohvat potrebnih podataka i korištenjem javne metode *onBindViewHolder* (koju poziva Recycler View) smještaju sve podatke na njima odgovarajuća mjesta.

```
@Override
public void onBindViewHolder(@NonNull CategoryViewHolder holder, int position) {
    CategoryModel categoryModel = categoryModels.get(position);
    holder.binding.label.setText(categoryModel.getName());
    Glide.with(context).load(categoryModel.getImage()).into(holder.binding.image);

    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(context, CategoryActivity.class);
            intent.putExtra( name: "categoryId", categoryModel.getId());
            intent.putExtra( name: "categoryName", categoryModel.getName());
            context.startActivity(intent);
        }
    });
}
```

```
@Override
public void onBindViewHolder(@NonNull ProductViewHolder holder, int position) {
    ProductModel productModel = productModels.get(position);
    Glide.with(context).load(productModel.getImage()).into(holder.binding.image);
    holder.binding.label.setText(productModel.getName());
    holder.binding.price.setText("€ " + productModel.getPrice());

    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(context, ProductDetailsActivity.class);
            intent.putExtra( name: "name", productModel.getName());
            intent.putExtra( name: "image", productModel.getImage());
            intent.putExtra( name: "id", productModel.getId());
            intent.putExtra( name: "price", productModel.getPrice());
            context.startActivity(intent);
        }
    });
}
```

Slika 50. i 51. Korištenje javne metode *onBindViewHolder* u adapterima kategorija i proizvoda

Za razliku od adaptera kategorija i proizvoda, koji u biti imaju istu funkciju samo s različitim podatcima, adapter košarice ima specifičnu zadaću, a tiče se ranije spomenutog prozora za promjenu količine u košarici. Zadan je tzv. *QuantityWindowBinding* koji prvo vrši dohvat imena, broja zalihe i trenutne količine proizvoda u košarici. Također prepoznaje gumbove za povećanje i smanjivanje količine, te je svakom gumbu zadana njegova funkcija. Dok gumb za povećanje količine (*increaseBtn*) koristi dohvat broja zaliha kako bi se spriječilo da korisnik odabere količinu koja je veća od broja proizvoda na zalihi.

[27] Codepath, "Using an ArrayAdapter with ListView", guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView

Gumb za smanjenje količine (*decreaseBtn*) takvu funkciju nema, ali napisan je s *if* petljom kojom se sprječava da se broj količine ne može smanjiti na manje od 1.

```
quantityWindowBinding.productName.setText(productModel.getName());
quantityWindowBinding.productQuantity.setText("In stock: " + productModel.getStock());
quantityWindowBinding.quantity.setText(String.valueOf(productModel.getQuantity()));
int stock = productModel.getStock();

quantityWindowBinding.increaseBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        int quantity = productModel.getQuantity();
        quantity++;

        if(quantity>productModel.getStock()){
            Toast.makeText(context, text: "No more available", Toast.LENGTH_SHORT).show();
        }else{
            productModel.setQuantity(quantity);
            quantityWindowBinding.quantity.setText(String.valueOf(quantity));
        }

        notifyDataSetChanged();
        cart.updateItem(productModel, productModel.getQuantity());
        cartListener.onQuantityChanged();
    }
});

quantityWindowBinding.decreaseBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        int quantity = productModel.getQuantity();
        if(quantity > 1){
            quantity--;
        }
        productModel.setQuantity(quantity);
        quantityWindowBinding.quantity.setText(String.valueOf(quantity));

        notifyDataSetChanged();
        cart.updateItem(productModel, productModel.getQuantity());
        cartListener.onQuantityChanged();
    }
});
```

Slika 52. Adapter košarice (CartAdapter)

Dodatno, adapter košarice sadrži i metodu kojom se sprema novi broj količine kojeg je korisnik odabrao. Koristeći metodu *notifyDataSetChanged*, novi broj količine proizvoda će se prikazati preko adaptera, a isto tako će se izvršiti i TinyCart metoda *updateItem* kojom će se proizvod u košarici u potpunosti ažurirati s novom zadanom količinom.

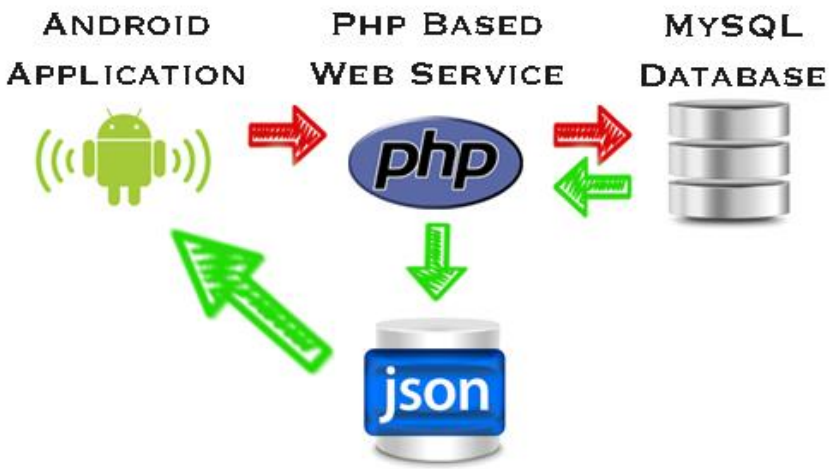
```
        quantityWindowBinding.confirmQuantity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                dialog.dismiss();

                notifyDataSetChanged();
                cart.updateItem(productModel, productModel.getQuantity());
                cartListener.onQuantityChanged();
            }
        });

        dialog.show();
    }
});
}
```

Slika 53. Metoda za potvrdu odabrane količine proizvoda u adapteru košarice

4.11. Programsko sučelje aplikacije (API)

»API je skup programskog koda koji omogućuje prijenos podataka između jednog softverskog proizvoda i drugog. Također sadrži uvjete za razmjenu podataka.«[28] REST (RESTful) API je programsko sučelje aplikacije »koji je u skladu s ograničenjima REST arhitektonskog stila u omogućuje interakciju s RESTful web uslugama. REST je kratica za reprezentativni prijenos stanja, a kreirao ga je računalni znanstvenik Roy Fielding.«[29]

Prethodno smo opisivali što aktivnosti uz pomoć modela i adaptera izvode i kakva je njihova uloga u radu mobilne aplikacije. Metode za dohvat (ili u slučaju metode za potvrdu narudžbe, slanje) podataka koje koriste Volley zahtjeve kako bi baratali podatcima su same po sebi beznačajne bez postavljanja programskog sučelja aplikacije (API) i servera na kojem su napisani programi u PHP-u koji obrađuju podatke koji se iz baze podataka šalju u mobilnu aplikaciju ili koji iz mobilne aplikacije dolaze kako bi se spremile u bazu podataka. Važno je naglasiti da se sva razmjena između mobilne aplikacije i PHP-a vrši isključivo preko JSON-a, koji se i inače koristi za razmjenu podataka između servera i aplikacije.



Slika 54. Grafičko objašnjenje veza između android aplikacije, PHP-a i baze podataka[30]

Najosnovnija stvar u mobilnoj aplikaciji prilikom definiranja API-a je kreiranje zasebne javne klase *Api* u kojoj su deklarirani javni statični stringovi koji su ustvari URL adrese koje vode na svaku potrebnu PHP skriptu. Osnovna varijabla *API_BASE_URL* sadrži glavni URL koji vodi do mape *mobile* u upravitelju datoteka na HostGator serveru. Ostale varijable koriste taj isti glavni URL uz dodatak puta (*path)* koji vodi do ciljane PHP datoteke. Varijable za dohvat fotografija (kategorija i proizvoda) i naslovnih fotografija imaju posebno zadane URL adrese jer oni izvlače potrebne fotografije iz drugih mapa u upravitelju datotekama.

[28] AltexSoft, "What is API: Definition, Types, Specifications, Documentation", altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/
[29] RedHat, "What is a REST API?", redhat.com/en/topics/api/what-is-a-rest-api
[30] TutorialsEye, "JSON with PHP", tutorialseye.com/json-with-php.html

```
public class Api {
    public static String API_BASE_URL = "https://abramovicluka-diplomski.com/Domus/mobile";
    public static String GET_CATEGORIES = API_BASE_URL + "/api/categories.php";
    public static String GET_TRENDING_PRODUCTS = API_BASE_URL + "/api/trending_products.php";
    public static String GET_PRODUCTS = API_BASE_URL + "/api/products.php?category_id=";
    public static String SEARCH_PRODUCTS = API_BASE_URL + "/api/search_products.php?query=";
    public static String GET_BANNERS = API_BASE_URL + "/api/banners.php";
    public static String GET_PRODUCT_DETAILS = API_BASE_URL + "/api/product_details.php?id=";
    public static String POST_ORDER = API_BASE_URL + "/api/product_order.php";

    public static String GET_UPLOADS = "https://abramovicluka-diplomski.com/Domus/web/uploads/";
    public static String GET_BANNER_IMG = "https://abramovicluka-diplomski.com/Domus/web/banners/";
}
```

Slika 55. Javna klasa Api

U upravitelju datotekama na HostGator serveru, u ranije spomenutoj mapi *mobile* stvorena je mapa *api* koja sadrži ukupno sedam skripti pisane u programskom jeziku PHP i svaka od tih skripti se koristi za obradu drugog zahtjeva koji se u slučaju šest skripti dohvaća (GET), i u slučaju jedne postavlja (POST).

```
<?php

if($_SERVER['REQUEST_METHOD'] == 'GET'){

    include('../config/connection.php');

    $select_categories = "SELECT * FROM categories";
    $select_categories_run = mysqli_query($con,$select_categories);

    $result = array();
    $result['categories'] = array();

    while($row = mysqli_fetch_assoc($select_categories_run)){
        $index['id'] = $row['id'];
        $index['name'] = $row['name'];
        $index['description'] = $row['description'];
        $index['image'] = $row['image'];
        $index['created_at'] = $row['created_at'];

        array_push($result['categories'], $index);
    }
    echo json_encode($result);
}
?>
```

Slika 56. PHP skripta *categories.php*

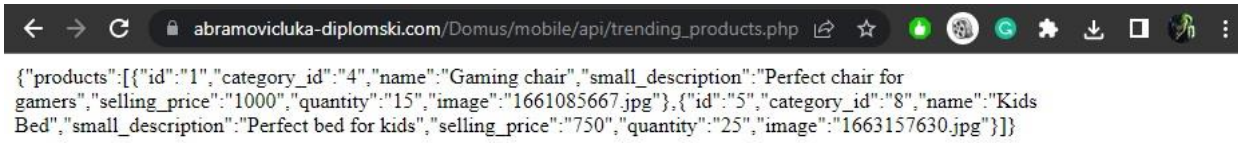
U klasi *Api* deklarirana je klasa *GET_CATEGORIES* kojoj je dodijeljena adresa koja na serveru vodi do datoteke *categories.php*. Skripta u startu sadrži if uvjet koji zahtjeva da vrsta zahtjeva bude GET, potom se implementira datoteka *connection.php* kojom se ostvaruje poveznica s bazom podataka. Potom se deklarira varijabla kojoj se zadaje SQL naredba SELECT kojom će se odabrati apsolutno svi zapisi u tablici kategorija (*categories*). Potom se pomoću varijable *result* koristi *while* petlja koja će proći kroz sve zapise te tablice i podatke potom kodirati u formatu JSON niza. Vrlo je jednostavno provjeriti ispravnost napisane skripte; upisivanjem pripadajuće URL adrese u web pretraživač. Rezultat koji se treba dobiti iz skripte su svi zapisi tablice *categories* formatirani u JSON-u. Ovako formatirani podaci se dohvaćaju metodom za dohvat u mobilnoj aplikaciji te tako dolaze do prikaza u korisničkom sučelju.

Slika 57. Provjera rada PHP skripte *categories.php*

Podaci o kategorijama koji se dobiju iz PHP skripte *categories.php* u korisničkom sučelju mobilne aplikacije prikazuju s u glavnoj aktivnosti, a isto vrijedi i za skriptu *trending_products.php,* koja na istovjetan način filtrira proizvode ovisno o tome jesu li oni u tablici označeni kao popularni. Jedan od stupaca u tablici proizvodi (*products*) u bazi podataka je *trending*, koji binarno označava je li određeni proizvod trenutno popularan. Shodno tome, proizvodi koji su u stupcu *trending* označeni s 1, to označava proizvod kao popularan. PHP skripta *trending_products.php* funkcionira na vrlo sličan način kao i skripta za kategorije, uz razliku da naredba SQL naredba SELECT sada ima zadatak pronaći sve redove u tablici proizvodi u kojima stupac *trending* nosi vrijednost 1. Potom se na isti način kao i skripti za kategorija koristi *while* petlja koja će proći kroz sve izabrane redove i kodirati ih u JSON niz koji možemo vidjeti jednom kada pokrenemo skriptu koristeći njegovu URL adresu u web pretraživaču.

```php
<?php

if($_SERVER['REQUEST_METHOD'] == 'GET'){

    include('../config/connection.php');

    $select_products = "SELECT * FROM products WHERE trending=1";
    $select_products_run = mysqli_query($con,$select_products);

    $result = array();
    $result['products'] = array();

    while($row = mysqli_fetch_assoc($select_products_run)){
        $index['id'] = $row['id'];
        $index['category_id'] = $row['category_id'];
        $index['name'] = $row['name'];
        $index['small_description'] = $row['small_description'];
        $index['selling_price'] = $row['selling_price'];
        $index['quantity'] = $row['quantity'];
        $index['image'] = $row['image'];

        array_push($result['products'], $index);
    }
    echo json_encode($result);
}
?>
```

Slika 58. PHP skripta *trending_products.php*



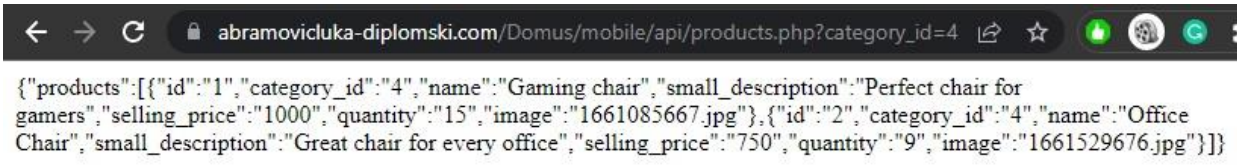
Slika 59. Provjera rada PHP skripte *trending_products.php*

Sljedeća PHP skripta *products.php* je funkcionira na sličan način, ali je specifična u jednom važnom detalju u odnosu na prethodne dvije. Zadaća ove skripte je filtrirati proizvode u JSON nizu prema identifikacijskom broju njihove kategorije. Međutim, kako PHP skripta ne može unaprijed znati za koju kategoriju će morati prikazati određene proizvode, taj problem je riješen korištenjem upita i parametra u URL adresi. Drugim riječima, mobilna aplikacija će dati do znanja o kojoj kategoriji se radi, a potom će se identifikacijski broj te kategorije upotrijebiti kao parametar URL adrese prilikom pozivanja PHP skripte.

```php
<?php

if($_SERVER['REQUEST_METHOD'] == 'GET'){

    include('../config/connection.php');

    $category_id = $_GET['category_id'];

    $select_products = "SELECT * FROM products WHERE category_id=$category_id";
    $select_products_run = mysqli_query($con,$select_products);

    $result = array();
    $result['products'] = array();

    while($row = mysqli_fetch_assoc($select_products_run)){
        $index['id'] = $row['id'];
        $index['category_id'] = $row['category_id'];
        $index['name'] = $row['name'];
        $index['small_description'] = $row['small_description'];
        $index['selling_price'] = $row['selling_price'];
        $index['quantity'] = $row['quantity'];
        $index['image'] = $row['image'];

        array_push($result['products'], $index);
    }
    echo json_encode($result);
}
?>
```

Slika 60. PHP skripta *products.php*

U skripti je zadana varijabla *category_id* u koju se sprema vrijednost globalne varijable *$_GET* koja zapravo poprima vrijednost parametra URL adrese, u ovom slučaju identifikacijskog broja tražene kategorije. Potom se klasično SELECT naredbom traže svi redovi u kojima je identifikacijski broj kategorije jednak vrijednosti varijable *category_id*. Po pronalasku svih odgovarajućih redova, podaci se ponovno korištenjem *while* petlje kodiraju u JSON niz.

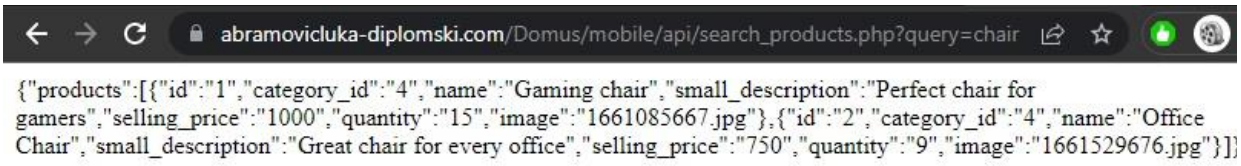


Slika 61. Provjera rada skripte *products.php* s 4 kao zadanim ID-em kategorije

Prethodno je spomenuto kako je u mobilnoj aplikaciji proizvode moguće filtrirati prema kategorijama, za što služi prethodno spomenuta PHP skripta, i prema tražilici. Filtriranje prema tražilici izvodi se preko PHP skripte *search_products.php*, koja funkcionira na gotovo identičan način kao i *products.php*, uz ključnu razliku da parametar URL adrese ovaj put nije identifikacijski broj kategorije već upit kojeg korisnik upisuje u tražilicu. Također, naredba SELECT ponovno je zadužena za odabir redova iz tablice proizvoda ali samo onih koji u svom imenu u cijelosti ili djelomično sadrže pojam koji je korisnik upisao. Za uspješno filtriranje imena proizvoda po korisnikovom upitu korišten je SQL operator LIKE te je varijabla *query,* koja sadrži vrijednost upita, smještena između dva znaka %, koji omogućuju fleksibilnije pretraživanje, odnosno da tražena riječ ne mora biti u cijelosti, već samo djelomično upisana kako bi se obavilo uspješno pretraživanje.

```php
$query = $_GET['query'];

$select_products = "SELECT * FROM products WHERE name LIKE '%$query%'";
$select_products_run = mysqli_query($con,$select_products);
```

Slika 62. Globalna varijabla GET i SELECT naredba u PHP skripti *search_products.php*



Slika 63. Provjera rada skripte *search_products.php* sa zadanim upitom "chair"

Dosadašnje PHP skripte imale su mogućnost u JSON niz spremiti više redova, odnosno odabrati sve moguće redove iz tablica koji odgovaraju uvjetima koje je zadavala naredba SELECT. S druge strane, PHP skripta *product_details.php* ima zadatak uzeti samo jedan red, s obzirom na to da se aktivnost proizvoda (*ProductDetailsActivity*, stranica 33.) može u jednom trenutku prikazati za samo jedan odabrani proizvod. Ipak, jedna sličnost koju ima s prethodnim PHP skriptama je da se i u ovom slučaju koristi parametar u URL adresi.

```
$id = $_GET['id'];

$select_products = "SELECT * FROM products WHERE id=$id";
$select_products_run = mysqli_query($con,$select_products);
```

Slika 64. Globalna varijabla GET i SELECT naredba u PHP skripti *product_details.php*

Identifikacijski broj proizvoda je primarni ključ u tablici proizvodi (*products*) i samim time je jedinstven. Shodno tome je i moguće odabrati samo jedan red iz tablice koji odgovara traženom identifikacijskom broju koji se nalazi u parametru URL adrese. Kao i u prethodnim slučajevima, ponovno se koristi globalna varijabla *$_GET* koja varijabli *id* daje vrijednost parametra i preko te varijable naredba SELECT ponovno traži odgovarajući red u tablici.

Slika 65. Provjera rada skripte *product_details.php*

Jedna od PHP skripta koja se najviše razlikuje od svih ostalih je skripta *product_order.php* koja je jedina PHP skripta koja koristi HTTP metodu POST, za razliku od ostalih koje koriste metodu GET. Jednom kada se u aktivnosti završavanja narudžbe (CheckoutActivity, stranica 38.) aktivira metoda za potvrdu narudžbe, svi podaci vezani za narudžbu, što uključuje podatke o korisniku i proizvodima unutar narudžbe se kodiraju u JSON niz kojeg PHP skripta dekodira korištenjem funkcije *file_get_contents* koja dotične podatke čita kao string, i potom se koristi i funkcija *json_decode* koji će dekodirati taj JSON niz kako bi se svi podaci unutar niza mogli spremiti na odgovarajuća mjesta unutar baze podataka. Potrebno je za naglasiti da su općeniti podaci o korisniku i narudžbi u JSON nizu spremljeni kao objekt zato što se tu radi samo o jednom novom redu koji će se spremiti u tablicu *mobile_orders*. S druge strane, podaci o naručenim proizvodima spremljeni u niz zato što postoji mogućnost da je korisnik naručio više proizvoda pa se ti podaci spremaju korištenjem petlje *foreach* koja će se pobrinuti da se svaki naručeni proizvod spremi u zasebni red u tablici *mobile_order_items.* Potrebno je naglasiti da je identifikacijski broj narudžbe primarni ključ u tablici *mobile_orders,* i taj identifikacijski broj

se pridodaje svakom redu tablici *mobile_order_items* kao *order_id*, kako bi se moglo razlučiti koji naručeni proizvod u spomenutoj tablici pripada kojoj narudžbi.

```php
<?php

if($_SERVER['REQUEST_METHOD'] == 'POST'){

    include('../config/connection.php');

    $jsondata = file_get_contents('php://input');

    $data = json_decode($jsondata, true);

    $name = $data['mobile_order']['name'];
    $email = $data['mobile_order']['email'];
    $phone = $data['mobile_order']['phone'];
    $address = $data['mobile_order']['address'];
    $date = $data['mobile_order']['date'];
    $comment = $data['mobile_order']['comment'];
    $tax = $data['mobile_order']['tax'];
    $total_price = $data['mobile_order']['total_price'];

    $insert = "INSERT INTO mobile_orders (name,email,phone,address,date,comment,tax,total_price)
               VALUES ('$name','$email','$phone','$address','$date','$comment','$tax','$total_price')";
    $insert_run = mysqli_query($con, $insert);
    $order_id = mysqli_insert_id($con);
    echo $order_id;

    foreach($data['mobile_order_items'] AS $array){
        $amount = $array['amount'];
        $price_item = $array['price_item'];
        $product_id = $array['product_id'];
        $product_name = $array['product_name'];

        $insert_items = "INSERT INTO mobile_order_items (order_id,amount,price_item,product_id,product_name)
                         VALUES ('$order_id','$amount','$price_item','$product_id','$product_name')";
        $insert_items_run = mysqli_query($con, $insert_items);
    }

    mysqli_close($con);
}
?>
```

Slika 66. PHP skripta *product_order.php*

```php
<?php

if($_SERVER['REQUEST_METHOD'] == 'GET'){

    include('../config/connection.php');

    $select_banners = "SELECT * FROM banners";
    $select_banners_run = mysqli_query($con,$select_banners);

    $result = array();
    $result['banners'] = array();

    while($row = mysqli_fetch_assoc($select_banners_run)){
        $index['id'] = $row['id'];
        $index['image'] = $row['image'];
        $index['created_at'] = $row['created_at'];

        array_push($result['banners'], $index);
    }
    echo json_encode($result);
}
?>
```

Slika 67. PHP skripta *banners.php*

Posljednja PHP skripta koja je preostala je *banners.php* koja služi za jednostavni dohvat naslovnih fotografija. Funkcionira na isti način kao i skripta za dohvat kategorija, što znači da naredba SELECT odabire apsolutno sve redove u tablici bez ikakvih uvjeta.

5. IMPLEMENTACIJA KORISNIČKOG DIJELA WEB APLIKACIJE

Dok je mobilna aplikacija predviđena da se koristi samo od strane standardnog korisnika, web aplikacija je kompleksnija jer se sastoji od dijela kojima može baratati samo standardni korisnik, dok su neki drugi dijelovi web aplikacije predviđeni da njima može pristupiti samo administrator. Prvotno je bilo zamišljeno da se web aplikacija razvije u okviru Vue.js, no od te ideje se na kraju odustalo te je kompletna web aplikacija razvijena pisanjem PHP skripti koje koriste HTML kao označni jezik korisničkog sučelja, uz korištenje CSS-a i programskog jezika JavaScript. Radi veće vizualne atraktivnosti i lakšeg korištenja, važno je naglasiti da web aplikacija koristi Bootstrap[31], najpoznatiji okvir za prednjeg korisničkog sučelja web aplikacija.

Kao i mobilna aplikacija, web aplikacija također ima mogućnost pregledavanja proizvodima i stvaranje narudžbi od strane korisnika, ali uz uvjet da narudžbe mogu stvarati samo registrirani korisnici. Korisnici također imaju mogućnost pregledavati prethodno stvorene narudžbe. Administrator također ima mogućnost koristiti web aplikaciju kao i standardni korisnik, uz dodatak da može pristupiti i administratorskoj ploči u kojoj ima mogućnost vršiti uvid u stvorene narudžbe, kao i direktno vršiti promjene u ponudi kategorija i proizvoda. Sve datoteke koje su dio web aplikacije smještene su u HostGator upravitelju datoteka u mapi *web.* U toj mapi su smještene PHP datoteke u kojima su razvijene sve stranice web aplikacije koje su dio tzv. korisničkog dijela web aplikacije. Uz spomenute datoteke, mapa *web* sadrži još osam pod-mapa, koje su sljedeće:

- *admin* – sadrži mape i datoteke koje su dio administratorskog dijela aplikacije
- *assets* – sadrži pod-mape u kojima se nalaze potrebne CSS i JavaScript datoteke
- *banners* – sadrži fotografije koje se u mobilnoj aplikaciji prikazuju kao naslovne fotografije
- *config* – sadrži samo PHP datoteku s kojom se uspostavlja veza s bazom podataka za mobilnu aplikaciju
- *functions* – sadrži PHP datoteke koje čine dodatne funkcije potrebne za rad web aplikacije
- *includes* – sadrži PHP datoteke *header.php, navbar.php,* i *footer.php* koje olakšavaju smještanje tih elemenata u svaku stranicu web aplikacije
- *middleware* – sadrži samo jednu PHP datoteku koja provjerava ima li prijavljeni korisnik administratorske privilegije prilikom korištenja web aplikacije
- *uploads* – sadrži fotografije kategorija i proizvoda čija se imena spremaju u odgovarajućim tablicama u bazi podataka

[31] Bootstrap, github.com/twbs/bootstrap

5.1. Zaglavlje, navigacijska traka, podnožje

Zaglavlje, navigacijska traka i podnožje su tri elementa koji su zasebno napisani u datotekama *header.php, navbar.php,* i *footer.php.* Ta tri elementa su napisani u zasebnim datotekama ne samo kako bi se olakšalo čitanje i provjera koda već i zato što su ti elementi dio svake stranice korisničkog dijela web aplikacije i stoga je mnogo jednostavnije koristiti u tim stranicama PHP naredbu *include* za uključivanje tih elemenata.

U praktičnoj upotrebi, zaglavlje web stranice »predstavlja uvodni sadržaj, obično skupinu uvodnih ili navigacijskih pomagala. Može sadržavati neke elemente naslova, ali i logotip, obrazac za pretraživanje, ime autora, i druge elemente.«[32] U ovom slučaju, zaglavlje sadrži oznaku za definiranje naslova *title* u kojoj je definiran naslov web aplikacije koji se pojavljuje u imenu kartice u web pretraživaču. Potom su preko oznaka *link* implementirani pomoćne skripte napisane u CSS-u i JavaScript-u i koji su korišteni za vizualno uređivanje stranica ili, u slučaju JavaScript-a, neke dodatne programske funkcionalnosti. Prva implementirana skripta je osnovna (i za potrebe projekta modificirana i spremljena u lokalnu mapu *assets*) CSS datoteka *bootstrap.min.css*[33]. Datoteka je licencirana pod MIT licencom[34] koja dozvoljava korištenje i modifikaciju datoteke u svrhu ovakvog projekta. Sljedeća oznaka link odnosi se na prilagođenu CSS datoteku *custom.css* u kojoj su napisane dodatne CSS naredbe koje su zapravo opcionalne, ali su dodatne radi jedinstvenog vizualnog identiteta web aplikacije. Sljedeća dodana oznaka *link* implementira vanjsku CSS datoteku *font-awesome.min.css* koja je dio Font Awesome, »internetske biblioteke ikona i skupa alata koje koriste milijuni dizajnera, programera i kreatora sadržaja.«[35], i čija licenca[36] također omogućuje slobodnu implementaciju datoteke u vlastitim projektima.

Datoteka zaglavlja također sadrži tri oznake link koje se umeću u projekt kako bi se omogućilo korištenje određenog fonta iz biblioteke Google Fonts, koja je »robusni katalog fontova i ikona otvorenog koda koji olakšava besprijekornu integraciju ekspresivnih slova i ikona.«[37] Korištenje spomenute biblioteke je izrazito jednostavno jer se sastoji samo od odabira željenog fonta u katalogu[38] i umetanja koda koji omogućuje integraciju fonta u web aplikaciju. Posljednje implementirate stavke u zaglavlju su dvije vanjske CSS datoteke koji dolaze iz biblioteke AlertifyJS[39], koja olakšava implementiranje dijaloških prozora i prikazivanje određenih obavijesti korisniku prilikom korištenja web aplikacije (npr. kada je potrebno obavijestiti korisnika da je prijava u sustav ili kreiranje narudžbe bilo uspješno). AlertifyJS je također praktičan prilikom programiranja prilagođenih (*custom*) funkcija u programskom jeziku JavaScript.

---

[32] MDN contributors, "Header", developer.mozilla.org/en-US/Web/HTML/Element/header
[33] Bootstrap, "bootstrap.min.css", github.com/twbs/bootstrap/blob/main/dist/css/bootstrap.min.css
[34] Bootstrap, "The Mit License", github.com/twbs/bootstrap/blob/main/LICENSE
[35] Fonticons, Inc., "Font Awesome", fontawesome.com
[36] Fonticons, Inc., "Font Awesome Pro License", fontawesome.com/license
[37] Google, "About Google Fonts", fonts.google.com/about
[38] Google, "Google Fonts", fonts.google.com
[39] Mohammad Younes, "AlertifyJS", alertifyjs.com

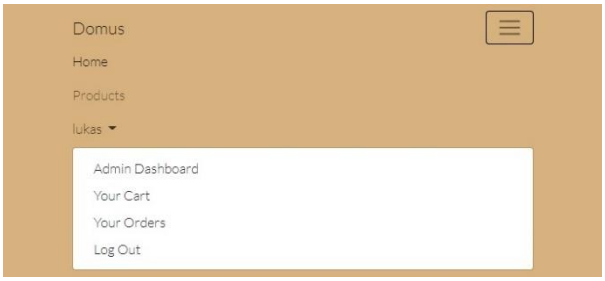
Nakon što se samo zaglavlje u datoteci *header.php* zatvori, koristi se oznaka za otvaranje tijela (*body*) web stranice, te se u posljednjem redu datoteke koristi PHP naredba *include* kako bi se automatski u svaku stranicu umetnula navigacijska traka koja je definirana u datoteci *navbar.php.*

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Domus Web App</title>
  <link href="assets/css/bootstrap.min.css" rel="stylesheet">
  <link href="assets/css/custom.css" rel="stylesheet">

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Lato:wght@300&display=swap" rel="stylesheet">

  <!-- AlertifyJS -->
  <link rel="stylesheet" href="//cdn.jsdelivr.net/npm/alertifyjs@1.13.1/build/css/alertify.min.css" />
  <link rel="stylesheet" href="//cdn.jsdelivr.net/npm/alertifyjs@1.13.1/build/css/themes/bootstrap.min.css" />


</head>

<body>
  <?php include('navbar.php') ?>
```

Slika 68. Implementacija zaglavlja (*header.php*)

Datoteka navigacijske trake *navbar.php* sastoji se isključivo od definiranja fiksnog izbornika koji se nalazi na vrhu svake stranice u korisničkom dijelu web aplikacije. Radi se o klasičnoj bootstrap navigacijskoj traci koja je napisana da prikazuje različite mogućnosti ovisno o tome je li korisnik ulogiran u sustav ili nije. Također, navigacijska traka je široka koliko i prozor web pretraživača, te zbog svoje responzivnosti se sav sadržaj navigacijske trake praktično kolapsira kada se prozor suzi toliko da navigacijska traka više ne može prikazati sve opcije u svom normalnom obliku.

Slika 69. Kolapsirani izbornik dok korisnik nije prijavljen



Slika 70. Kolapsirani izbornik dok je korisnik prijavljen

Implementiranje navigacijske trake počinje otvaranjem oznake *nav* u kojoj je kao parametar definirana i boja navigacijske trake. Sami izbornik je definiran klasičnim HTML oznakama koje se koriste za stvaranje liste. Glavna oznaka *ul* koja se koristi za liste koje nemaju određeni redoslijed, i oznaka *li* u koju se piše svaki pojedinačni član, odnosno opcija u toj listi, odnosno izborniku. Ključni dio je korištenje uvjeta *if* koji provjerava stanje sesije, odnosno provjerava je li korisnik trenutno prijavljen u sustav. Ukoliko je, navigacijska traka pokazuje i ime korisnika te se klikom na ime otvara padajući izbornik u kojem korisnik može pregledati svoju košaricu, narudžbe, ili se odjaviti iz sustava. U

padajućem izborniku je ponuđena i opcija odlaska u administratorski dio aplikacije, ali to je dozvoljeno samo korisnicima koji imaju dozvoljene administratorske privilegije.

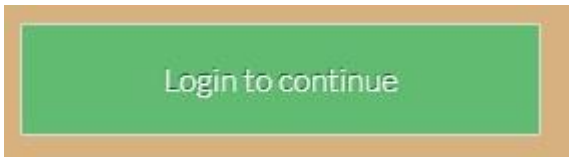
```
<nav class="navbar navbar-expand-lg bg-domus">
  <div class="container">
    <a class="navbar-brand" href="index.php">Domus</a>
    <button class="navbar-toggler ml-auto custom-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavDropdown"
       aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon my-toggler"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavDropdown">
      <ul class="navbar-nav ms-auto">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="index.php">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="categories.php">Products</a>
        </li>
        <?php
        if (isset($_SESSION['auth'])) {
        ?>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink" role="button" data-bs-toggle="dropdown" aria-expanded="false">
              <?= $_SESSION['auth_user']['name']; ?>
            </a>
            <ul class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
              <li><a class="dropdown-item" href="admin/index.php">Admin Dashboard</a></li>
              <li><a class="dropdown-item" href="cart.php">Your Cart</a></li>
              <li><a class="dropdown-item" href="myOrders.php">Your Orders</a></li>
              <li><a class="dropdown-item" href="logout.php">Log Out</a></li>
            </ul>
          </li>
        <?php
        } else {
        ?>
          <li class="nav-item">
            <a class="nav-link" href="registration.php">Registration</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="login.php">Login</a>
          </li>
        <?php
        }
        ?>
      </ul>
    </div>
  </div>
</nav>
```

Slika 71. Implementacija navigacijske trake (*navbar.php*)

Konačno, datoteka podnožja *footer.php* je treća i posljednja datoteka koja se nalazi u pod-mapi *include* za korisnički dio aplikacije. Za razliku od zaglavlja, u kojem su korištene oznake za implementiranje vanjskih CSS datoteka, u podnožju su definirane oznake za implementaciju vanjskih JavaScript datoteka čije su funkcije zbog pozicije podnožja također primjenjive u kompletnom korisničkom dijelu web aplikacije. Shodno implementiranom CSS datotekom u zaglavlju, u podnožju je implementirana i vanjska JavaScript datoteka koja omogućuje pravilnu implementaciju Bootstrapa, uz dodatak vanjske datoteke[40] koja implementira jQuery, »brzu, malu JavaScript biblioteku bogatu značajkama koja čini stvari kao što su pronalaženje i manipulacija HTML dokumentom.«[41] Spomenuta datoteka također koristi prethodno spomenutu MIT licencu[42] koja dozvoljava korištenje napisane datoteke u svrhu projekta.

[40] OpenJS Foundation and other jQuery contributors, "Downloading jQuery", jquery.com/download
[41] OpenJS Foundation and other jQuery contributors, "What is jQuery?", jquery.com
[42] OpenJS Foundation and other jQuery contributors, "License", jquery.org/license

U podnožju je također umetnuta zasebna oznaka koja implementira posebna JavaScript datoteka *custom.js* koja sadrži dodatne potrebne funkcije potrebne za rad web aplikacije. Konačno, u podnožju je i oznaka za implementiranje vanjske JavaScript datoteke koja je shodna CSS datoteci iz biblioteke AlertifyJS. Na dnu podnožja je napisana i jednostavna funkcija koja koristi AlertifyJS biblioteku kako bi izvršila zadatke te biblioteke, a to je prikazivanje dijaloškog prozora, odnosno obavijesti koje sustav šalje korisniku.



Slika 72. Primjer AlertifyJS dijaloškog prozora

```
<script src="assets/js/jquery-3.6.0.min.js"></script>
<script src="assets/js/bootstrap.bundle.min.js"></script>

<script src="assets/js/custom.js"></script>

<!-- AlertifyJS -->
<script src="//cdn.jsdelivr.net/npm/alertifyjs@1.13.1/build/alertify.min.js"></script>

<script>
    alertify.set('notifier', 'position', 'top-center');
    <?php
    if(isset($_SESSION['message'])){
    ?>
        alertify.success('<?= $_SESSION['message']; ?>');
    <?php
    unset($_SESSION['message']);
    }
    ?>
</script>

</body>

</html>
```

Slika 73. Implementacija podnožja (*footer.php*)

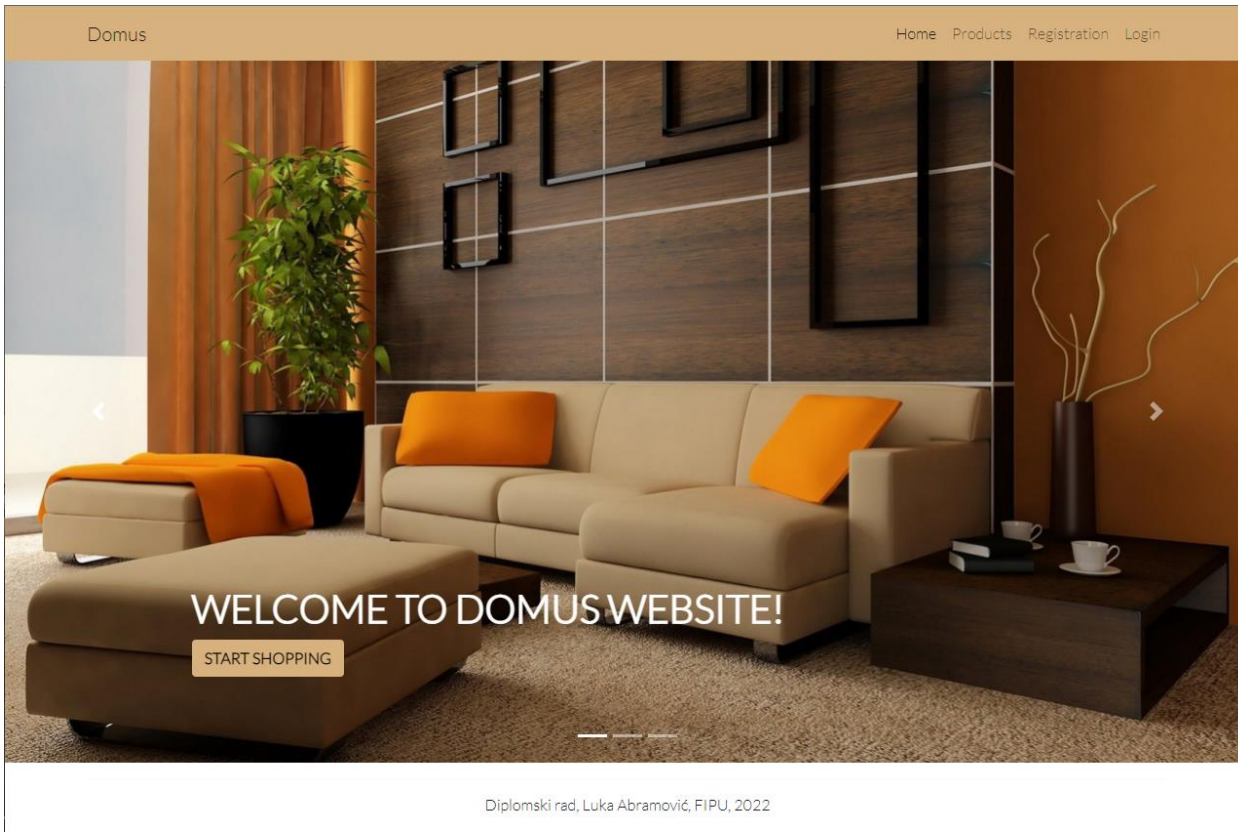
5.2. Naslovna stranica

Ono što je uobičajen slučaj prilikom razvoja web aplikacija i web sjedišta, naslovna stranica kompletne web aplikacije nosi ime *index.* Riječ je o glavnoj stranici u koju su preko PHP naredbe include uključeni zaglavlje i podnožje, a shodno sa zaglavljem i navigacijska traka. Shodno tome, sve CSS i JavaScript datoteke koje su umetnute u datoteke zaglavlja i podnožja sada su primjenjive i na naslovnoj stranici.

Naslovna stranica je ono prvo što korisnik vidi prilikom ulaska u web aplikaciju, i iz tog razloga naslovna stranica mora biti vizualno najatraktivniji dio aplikacije. Prosječni korisnik interneta danas u kratkom vremenu može proći kroz jako puno web stranica i aplikacija, i svaka od njih ima samo nekoliko dragocjenih sekundi kojima mora izazvati pažnju korisnika da bi on na toj stranici ostao što duže. Iz tog razloga se naslovne stranice danas ispunjavaju mnogim vizualno atraktivnim stavkama, a jedna od najkorištenijih je karusel ili vrtuljak, koji se koristi za prikazivanje i izmjenu više fotografija koje u većini slučajeva budu kombinirane s nekim kraćim tekstom koji može biti neka vrsta slogana, ili jednostavno korisniku dati neke prve informacije koje će natjerati korisnika na daljnje istraživanje onoga što web aplikacija nudi. U ovom slučaju, korišten je bootstrap karusel[43] koji se sastoji od tri slajda s tri različite fotografije i kraćim tekstom koji se direktno obraća

[43] Bootstrap, "Carousel", getbootstrap.com/docs/4.0/components/carousel

korisniku koji je otvorio stranicu. Iako je to moguće učiniti i preko navigacijske trake, na naslovni slajd je dodan i gumb koji korisnika automatski vodi na pregled ponude jer je vrlo bitno da aplikacija bude što jednostavnija i pristupačnija korisniku, i pritom što funkcionalnija. Važno je za napomenuti da karusel web aplikacije ne funkcionira na isti način kao i onaj u mobilnoj aplikaciji. Dok je fotografije u mobilnoj aplikaciji moguće izmjenjivati preko administratorske ploče, fotografije korištene u karuselu web aplikacije su fiksne i mogu se promijeniti samo izmjenom izvornog HTML koda.



Slika 74. Naslovna stranica web aplikacije (*index.php*)

5.3. Prijava i registracija korisnika

Iako standardni korisnik može pregledavati ponudu i bez korisničkog računa, korisnički račun je svakako potreban korisniku koji želi umetati željene proizvode u košaricu i stvarati narudžbe. Stvaranje korisničkih računa danas je jedna od najstandardnijih stvari koje web aplikacije traže od korisnika kako bi dobili potpuno iskustvo korištenja te web aplikacije. Uz to, registracija korisnika u sustavu pomaže i administratorima web aplikacija zbog vođenja statistike i prilagođavanja web aplikacije korisnikovim potrebama. Iako se prijave i registracija korisnika danas najlakše povezuje s društvenim mrežama i sličnim servisima, i internet trgovine danas često traže od korisnika da se prijave radi dodatnih pogodnosti koje određena internet trgovina može ponuditi članovima svojeg kluba. Proces prijave i registracije korisnika je danas na internetu sveprisutan i jednostavan za shvatiti.

Obrasci za registraciju i prijavu su napisani u dvije zasebne datoteke; *registration.php* i *login.php*. Korisnici koji već imaju postojeće korisničke račune koriste obrazac za prijavu u kojoj su potrebni samo email (u nekim slučajevima korisničko ime) i lozinka, dok je za registraciju osim toga potrebno i još nekoliko osobnih podataka, i uvijek se prilikom registracije upis lozinke traži dva puta radi sigurnosti.

Datoteka *registration.php* na svom početku i kraju koristi PHP naredbu *include* kojom se zaglavlje i podnožje umeću u tu stranicu. Sami obrazac je jednostavno stvoriti korištenjem bootstrap komponenti za stvaranje obrazaca.[44]

```
<form action="functions/authcode.php" method="POST">
    <div class="mb-3">
        <label class="form-label">Name</label>
        <input type="text" name="name" class="form-control" placeholder="Enter your name">
    </div>
    <div class="mb-3">
        <label class="form-label">Phone</label>
        <input type="number" name="phone" class="form-control" placeholder="Enter your phone number">
    </div>
    <div class="mb-3">
        <label for="exampleInputEmail1" class="form-label">Email address</label>
        <input type="email" name="email" class="form-control" placeholder="Enter your email" id="exampleInputEmail1" aria-describedby="emailHelp">
    </div>
    <div class="mb-3">
        <label for="exampleInputPassword1" class="form-label">Password</label>
        <input type="password" name="password" class="form-control" placeholder="Enter password" id="exampleInputPassword1">
    </div>
    <div class="mb-3">
        <label class="form-label">Confirm password</label>
        <input type="password" name="cpassword" class="form-control" placeholder="Confirm password">
    </div>
    <button type="submit" name="register_btn" class="btn btn-domus">Register</button>
</form>
```

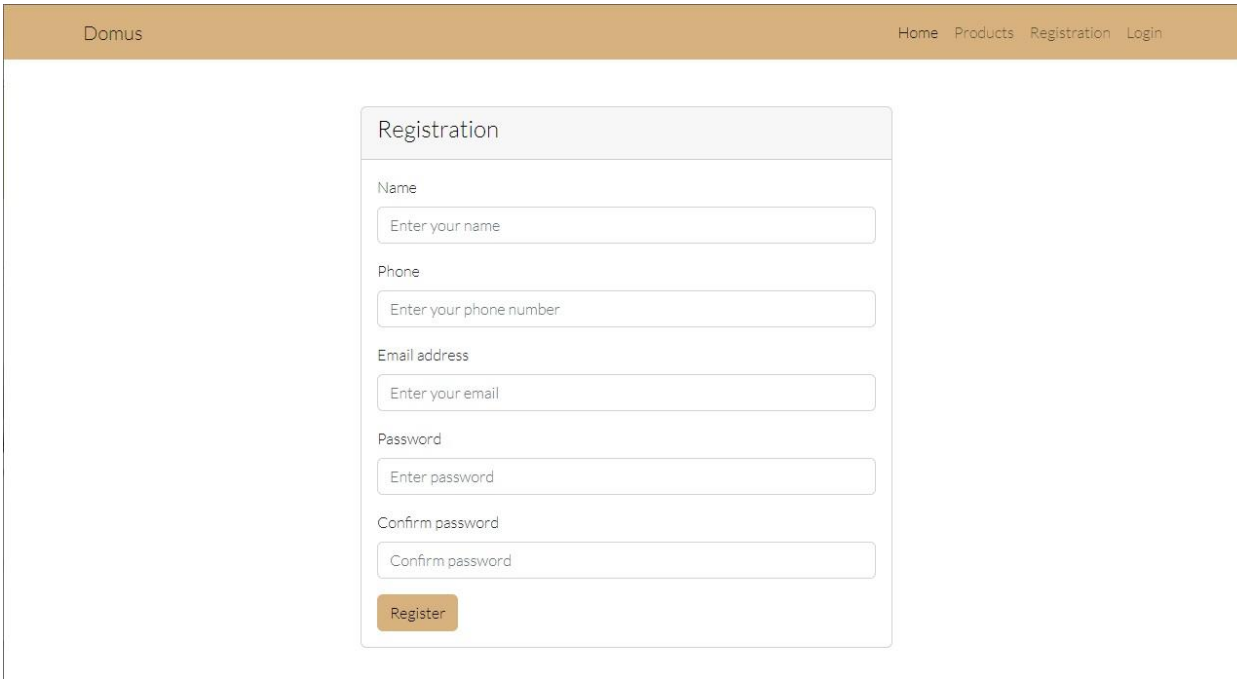
Slika 75. Implementacija obrasca za registraciju (*registration.php*)

Ono što je ključno je u oznaku *form* dodati atribut *action* koji aktivira posebni kod napisan u datoteci *authcode.php* unutar mape *functions.* Taj kod prepoznaje ime gumba kojim korisnik potvrđuje upisane podatke te upisane podatke sprema u zasebne PHP varijable. Prije same registracije, PHP program uz pomoć SQL naredbe SELECT provjerava je li upisani mail već prije bio korišten. Ukoliko je, program će javiti korisniku da registracija nije moguća jer se upisani mail već iskoristio. Ukoliko upisani email nije prethodno korišten, doći će to još jedne provjere, a to je provjera jesu li korisnik pravilo upisano lozinku po drugi put. Ukoliko se vrijednosti tih dviju varijabli koje se uspoređuju iste, izvršava se SQL naredba INSERT INTO koja upisane podatke sprema u tablicu korisnici (*users*).

[44] Bootstrap, "Forms", getbootstrap.com/docs/4.0/components/forms

```php
if(isset($_POST['register_btn'])){
    $name = mysqli_real_escape_string($con, $_POST['name']);
    $phone = mysqli_real_escape_string($con, $_POST['phone']);
    $email = mysqli_real_escape_string($con, $_POST['email']);
    $password = mysqli_real_escape_string($con, $_POST['password']);
    $cpassword = mysqli_real_escape_string($con, $_POST['cpassword']);


    $check_email_query = "SELECT email FROM users WHERE email='$email'";
    $check_email_query_run = mysqli_query($con, $check_email_query);

    if(mysqli_num_rows($check_email_query_run) > 0){
        $_SESSION['message'] = "This email is already registered";
        header('Location: ../registration.php');
    }
    else{
        if($password == $cpassword){
            $insert_query = "INSERT INTO users (name,email,phone,password) VALUES ('$name','$email','$phone','$password')";
            $insert_query_run = mysqli_query($con, $insert_query);

            if($insert_query_run){
                $_SESSION['message'] = "You are registered!";
                header('Location: ../login.php');
            }
            else{
                $_SESSION['message'] = "Something went wrong";
                header('Location: ../registration.php');
            }
        }
        else{
            $_SESSION['message'] = "Passwords do not match";
            header('Location: ../registration.php');
        }
    }
}
```

Slika 76. PHP program za provjeru i spremanje upisanih podatka za registraciju (*authcode.php*)



Slika 77. Obrazac za registraciju korisnika (*registration.php*)

Obrazac za prijavu korisnika u sustav je u datoteci *login.php* implementiran slično kao i obrazac registracije, korištenjem Bootstrap obrazaca, uz razliku da je u obrazac za prijavu potrebno upisati manje podataka pa je samim tim i kod za implementaciju obrasca kraći. Ključna razlika između registracije i prijave je u PHP programu *authcode.php* koji sadrži

if petlju kojom prvo preko imena gumba raspoznaje radi li se o registraciji novog korisnika ili o prijavi postojećeg. Upisani podaci za prijavi, email i lozinka, ponovno se spremaju kao vrijednosti u sebi pripadajuće varijable u programu, s tim da se sada koristi SQL naredba SELECT koja provjerava da su i email i lozinka točno upisani. Potom se uzimaju svi podaci prijavljenog korisnika i varijable koje sadrže podatke o tom korisniku u niz varijabli koje će biti aktivne tijekom cijelog trajanja sesije, odnosno dokle god je korisnik prijavljen. Još jedan dodatak je da se u tablici korisnika provjerava ima li korisnik administratorske privilegije. Ukoliko ima, automatski će biti preusmjeren prema administratorskom dijelu aplikacije, u suprotnom će se ponovno naći na naslovnoj stranici aplikacije, uz razliku da će ovaj put biti prijavljen u sustav.

```
else if(isset($_POST['login_btn'])){
    $email = mysqli_real_escape_string($con, $_POST['email']);
    $password = mysqli_real_escape_string($con, $_POST['password']);

    $login_query = "SELECT * FROM users WHERE email='$email' AND password='$password'";
    $login_query_run = mysqli_query($con, $login_query);

    if(mysqli_num_rows($login_query_run) > 0){
        $_SESSION['auth'] = true;

        $userdata = mysqli_fetch_array($login_query_run);
        $userid = $userdata['id'];
        $username = $userdata['name'];
        $useremail = $userdata['email'];
        $role_as = $userdata['role_as'];

        $_SESSION['auth_user'] = [
            'user_id' => $userid,
            'name' => $username,
            'email' => $useremail
        ];

        $_SESSION['role_as'] = $role_as;

        if($role_as == 1){
            redirect("../admin/index.php", "Welcome to Admin Dashboard!");
        }
        else{
            redirect("../index.php", "Logged in!");
        }
    }
    else{
        redirect("../login.php", "Wrong email or password");
    }
}
?>
```

Slika 78. PHP program za provjeru i prijavu korisnika (*authcode.php*)

5.4. Prikaz kategorija

Slično kao i u mobilnoj aplikaciji, web aplikacije kompletnu ponudu povlači iz baze podataka. Datoteka *categories.php* u sebi ima definirane bootstrap klase koje oblikuju korisničko sučelje prikaza kategorija, dok se korištenjem PHP-a uzimaju svi podaci iz tablice kategorija (*categories*) i korištenjem PHP konstrukta *foreach* se svaki od prikupljenih podataka prikazuje u korisničkom sučelju.

```
function getAllActive($table){
    global $con;
    $query = "SELECT * FROM $table WHERE status='0'";
    return $query_run = mysqli_query($con, $query);
}
```

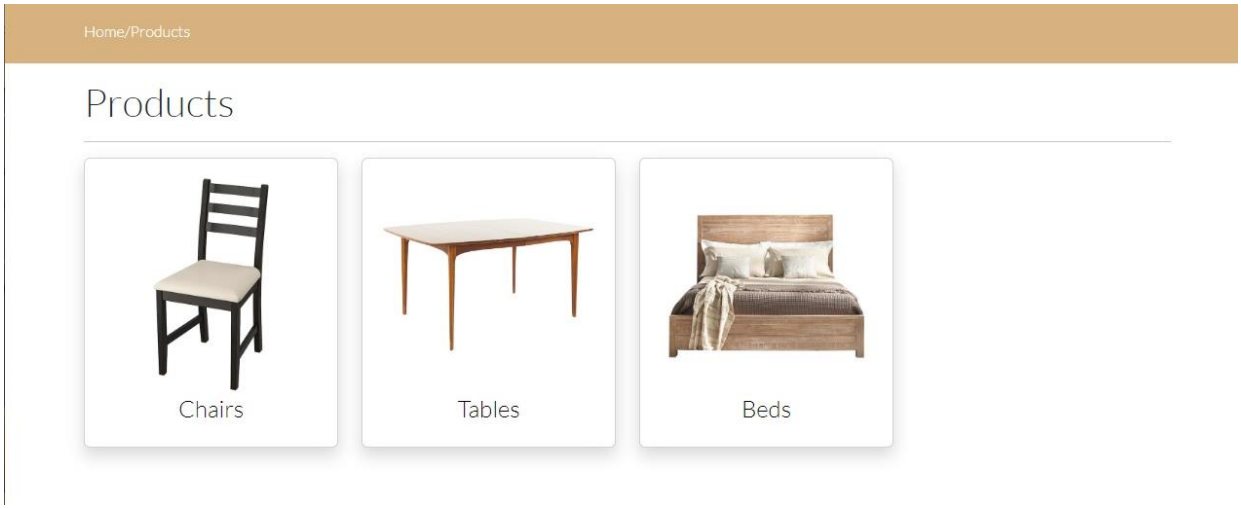
Slika 79. Funkcija *getAllActive* (*userFunctions.php*)

Važno je za naglasiti da se to uzimanje podataka vrši uz pomoć funkcije *getAllActive* koja je deklarirana u datoteci korisničkih funkcija *userFunctions.php*, koja se nalazi u pod-mapi *functions.*

Datoteka korisničkih funkcija na svom početku koristi naredbu *include* s kojom se ostvaruje umetanje već više puta navedene datoteke *connection.php* s kojom se ostvaruje povezanost s bazom podataka. Datoteka korisničkih funkcija sadrži ukupno osam napisanih funkcija od kojih svaka ima svoju zasebnu zadaću, no u slučaju kategorija proizvoda se poziva samo funkcija *getAllActive* u kojoj je zadano da se iz određene tablice uzmu svi podaci koji su trenutno na stanju (moguće je imati kategoriju ili proizvod smješten u bazi podataka, ali zadanog statusa da se ne prikazuje korisniku). U datoteci *categories.php* je nužno pozvati tu funkciju i navesti tablicu kategorija kao atribut iz razloga jer je *getAllActive* višenamjenska funkcija koja kao atribut može imati zadanu bilo koju tablicu.

```
<div class="row">
    <?php
        $categories = getAllActive("categories");

        if(mysqli_num_rows($categories) > 0){
            foreach($categories as $item){
                ?>
                <div class="col-md-3 mb-2">
                    <a href="products.php?category=<?= $item['slug']; ?>">
                        <div class="card shadow">
                            <div class="card-body">
                                <img src="uploads/<?= $item['image']; ?>" alt="Category Image" class="w-100">
                                <h4 class="text-center"><?= $item['name']; ?></h4>
                            </div>
                        </div>
                    </a>
                </div>
                <?php
            }
        }
        else{
            echo "Nothing to show!";
        }
    ?>
</div>
```

Slika 80. Implementacija reda za prikaz kategorija u web aplikaciji (*categories.php*)



Slika 81. Prikaz kategorija u web aplikaciji (*categories.php*)

5.5. Prikaz proizvoda po kategorijama

Kao i u slučaju mobilne aplikacije, prilikom pretraživanja kataloga, po odabiru kategorije je nužno da se prikažu svi dostupni proizvodi koji spadaju u tu kategoriju.

```
function getProductByCategory($category_id){
    global $con;
    $query = "SELECT * FROM products WHERE category_id='$category_id' AND status='0'";
    return $query_run = mysqli_query($con, $query);
}
```

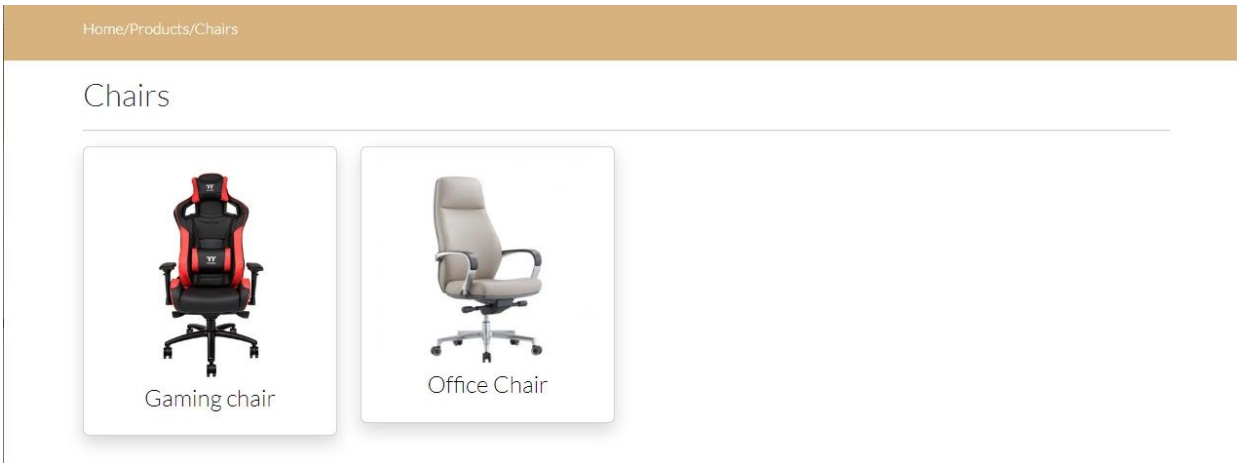
Slika 82. Funkcija *getProductByCategory* (*userFunctions.php*)

Za to je zadužena datoteka proizvoda *products.php* koja je po ideji vrlo slična datoteci kategorija. Neke od razlika su da se u naslovu uvijek prikazuje ime odabrane kategorije, te se umjesto funkcije *getAllActive* koristi funkcija *getProductByCategory*, koja se također poziva iz datoteke korisničkih funkcija u kojoj je deklarirana. Ta funkcija koristi identifikacijski broj odabrane kategorije kao atribut, te koristi SQL naredbu SELECT u kojoj se traže svi proizvodi koji su na stanju i kojima je identifikacijski broj kategorije jednak identifikacijskom broju odabrane kategorije.

```
<div class="row">
    <?php
        $products = getProductByCategory("$category_id");

        if(mysqli_num_rows($products) > 0){
            foreach($products as $item){
                ?>
                    <div class="col-md-3 mb-2">
                        <a href="productView.php?product=<?= $item['slug']; ?>">
                            <div class="card shadow">
                                <div class="card-body">
                                    <img src="uploads/<?= $item['image']; ?>" alt="Product Image" class="w-100">
                                    <h4 class="text-center"><?= $item['name']; ?></h4>
                                </div>
                            </div>
                        </a>
                    </div>
                <?php
            }
        }
        else{
            echo "Nothing to show!";
        }
    ?>
</div>
```

Slika 83. Implementacija reda za prikaz proizvoda po kategorijama (*products.php*)

Kao i u slučaju kategorija, i datoteka *products.php* prikazuje artikle u kartičnom prikazu, te se koristi PHP konstruktor *foreach* koji prolazi kroz sve odabrane proizvode iz tablice da bi ih sve pravilno prikazao u korisničkom sučelju. Klikom na određeni proizvod, korisnik je upućen na stranicu pregleda jednog proizvoda čija URL adresa sadrži ime proizvoda kao tzv. *slug.*



Slika 84. Prikaz proizvoda po kategorijama u web aplikaciji (*products.php*)

5.6. Prikaz detalja jednog proizvoda

Stranica za pregled detalja o jednom proizvodu *productView.php* je najdalje koliko standardni korisnik može doći u web aplikaciji bez prijave. Sljedeći korak, dodavanje proizvoda u košaricu, može se izvesti samo ukoliko je korisnik prijavljen u sustav. Odabirom određenog proizvoda na stranici *products.php* korisnik dolazi na ovu stranicu gdje može istražiti detalje odabranog proizvoda. Ono što je ključni detalj na ovoj stranici je tzv. *slug,* »jedinstveni identifikacijski dio web adrese, obično na kraju URL-a.«[45] *Slug* odabire i unosi administrator prilikom upisivanja proizvoda u bazu podataka. S obzirom da se radi o URL adresi, preporučljivo je da *slug* ne sadrži velika slova i ne smije sadržavati razmak. Za *slug* se uglavnom koristi ime proizvoda, ukoliko se radi o imenu s više riječi, riječi se mogu odvojiti crticom, ali ne nužno, najvažnije je da je *slug* jedinstven.

Slika 85. URL adresa stranice *productView.php,* sa označenim *slugom*

Slug odabranog proizvoda prikazuje se na kraju URL adrese stranice *productView.php.* Prilikom implementacije te stranice, koristi se globalna PHP varijabla *$_GET* koja će uzeti taj slug iz URL adrese i upotrijebiti ga kao atribut u funkciji *getActiveSlug* koji se kao i ostale funkcije nalaze u datoteci korisničkih funkcija. Na osnovu te funkcije, odnosno tog *slug-a,* SELECT naredba prepoznaje potrebni proizvod u tablici čije podatke mora prikazati u korisničkom sučelju stranice.

```
function getActiveSlug($table, $slug){
    global $con;
    $query = "SELECT * FROM $table WHERE slug='$slug' AND status='0' LIMIT 1";
    return $query_run = mysqli_query($con, $query);
}
```

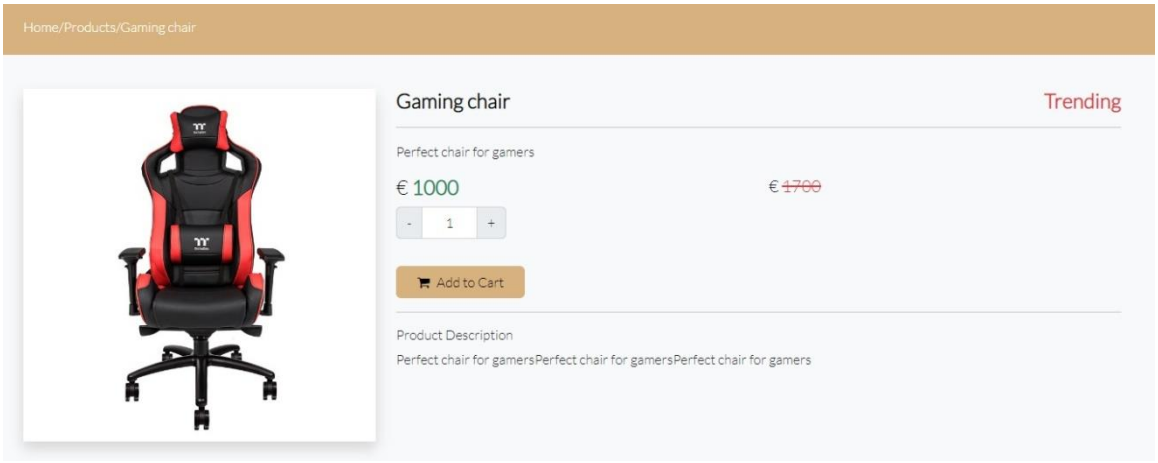
Slika 86. Funkcija *getActiveSlug* (*userFunctions.php*)

[45] MDN Web Docs Glossary, "Slug", developer.mozilla.org/en-US/docs/Glossary/Slug

Prikaz detalja proizvoda prikazuje korisniku sve potrebne informacije o proizvodu; ime, opis, cijenu, ali i podatak je li proizvod trenutno popularan. Ukoliko je određeni proizvod popularan, u mobilnoj aplikaciji se prikazuje u glavnoj aktivnosti, dok u web aplikaciji desno od svog imena ima oznaku *Trending,* kao što je i slučaj na slici 88. Prikaz detalja također sadrži i mogućnost promjene količine kao i gumb za dodavanje određenog gumba u košaricu.

```
<div class="bg-light py-4">
    <div class="container product_data mt-3">
        <div class="row">
            <div class="col-md-4">
                <div class="shadow">
                    <img src="uploads/<?= $product['image']; ?>" alt="Product image" class="w-100">
                </div>
            </div>
            <div class="col-md-8">
                <h4 class="fw-bold"><?= $product['name']; ?>
                    <span class="float-end text-danger"><?php if($product['trending']){ echo "Trending";} ?></span>
                </h4>
                <hr>
                <p><?= $product['small_description']; ?></p>
                <div class="row">
                    <div class="col-md-6">
                        <h4>€ <span class="text-success fw-bold"><?= $product['selling_price']; ?></span></h4>
                    </div>
                    <div class="col-md-6">
                        <h5>€ <s class="text-danger"><?= $product['original_price']; ?></s></h5>
                    </div>
                </div>

                <div class="row">
                    <div class="col-md-4">
                        <div class="input-group mb-3" style="width: 130px;">
                            <button class="input-group-text decrease-btn">-</button>
                            <input type="text" class="form-control text-center quantity-input bg-white" value="1" disabled>
                            <button class="input-group-text increase-btn">+</button>
                        </div>
                    </div>
                </div>
                <div class="row mt-3">
                    <div class="col-md-6">
                        <button class="btn btn-domus px-4 cartButton" value="<?= $product['id']; ?>"><i class="fa fa-shopping-cart me-2"></i>Add to Cart</button>
                    </div>
                </div>
                <hr>

                <h6>Product Description</h6>
                <p><?= $product['description']; ?></p>
            </div>
        </div>
    </div>
</div>
```

Slika 87. Implementacija prikaza detalja proizvoda (*productView.php*)



Slika 88. Prikaz detalja proizvoda (*productView.php*)

Funkcionalnosti gumba za povećanje i smanjenje količine proizvoda te funkcionalnost gumba za dodavanje proizvoda u košaricu definirano je u programskom jeziku JavaScript u ranije spomenutoj vanjskoj datoteci *custom.js* koja je umetnuta u podnožje korisničkog dijela web aplikacije. U toj datoteci su deklarirane funkcije koja se svaka zasebno aktivira klikom na odgovarajući gumb.

Funkcije za povećanje ili smanjenje količine proizvoda su načelno dosta slične. Obje funkcije se aktiviraju klikom na odgovarajući gumb te se time pokreće određena događaj. Obje funkcije sadrže sljedeće metode:

- *preventDefault()* — metoda koja »govori korisničkom agentu da ako se događaj eksplicitno ne obradi, njegova zadana radnja ne bi trebala biti poduzeta kao što bi inače bila.«[46]
- *closest()* — metoda koja »prelazi element i njegove roditelje (usmjeravajući se prema korijenu dokumenta) dok ne pronađe čvor koji odgovara selektoru.«[47] U ovom slučaju, selektor je *product_data,* ime HTML klase *div-a* u datoteci *cart.php.*
- *find()* — metoda koja pronalazi *quantity-input,* označava mjesto u kojem je upisana količina proizvoda.
- *val()* — metoda koja se koristi za dohvat vrijednosti količine.
- *parseInt()* — metoda koja »analizira vrijednost kao string i vraća prvi cijeli broj.«[48]
- *isNaN()* — metoda koja provjerava je li zadana vrijednost ne-broj (*not-a-number*). Dodatno se koristi operator uvjetni operator (znak upitnika) koji koristi metodu *isNan* kao uvjet, te vraća nulu ako je uvjet istinit.

```
$(document).on('click','.increase-btn', function(e){
    e.preventDefault();

    var quantity = $(this).closest('.product_data').find('.quantity-input').val();

    var value = parseInt(quantity, 10);
    value = isNaN(value) ? 0 : value;
    if(value < 10){
        value++;
        $(this).closest('.product_data').find('.quantity-input').val(value);
    }
});

$(document).on('click','.decrease-btn', function(e){
    e.preventDefault();

    var quantity = $(this).closest('.product_data').find('.quantity-input').val();

    var value = parseInt(quantity, 10);
    value = isNaN(value) ? 0 : value;
    if(value > 1){
        value--;
        $(this).closest('.product_data').find('.quantity-input').val(value);
    }
});
```

Slika 89. JavaScript funkcije za povećanje i smanjenje količine proizvoda

[46] MDN Web Docs, "preventDefault", developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault
[47] MDN Web Docs, "closest", developer.mozilla.org/en-US/docs/Web/API/Element/closest
[48] W3Schools, "JavaScript parseInt", w3schools.com/jsref/jsref_parseint.asp

Konačno, obje funkcija sadrže klasične *if* uvjete koji u slučaju funkcije za povećanje količine uvjetuju da odabrana količina mora biti manja od deset, a u slučaju funkcije za smanjenje odabrana količina mora biti veća od jedan. Novo-odabrana količina se potvrđuje dodavanjem proizvoda u košaricu s pripadajućom količinom.

U datoteci *custom.js* je deklarirana i funkcija koja je od ključnog značaja za dodavanje proizvoda u košaricu i koja se aktivira klikom na sopstveni gumb. Ta funkcija postaje zanimljiva kada na red doše korištenje Ajaxa. »Asinkroni JavaScript i XML, ili Ajax, nije tehnologija sama po sebi, već prije pristup zajedničkom korištenju brojnih postojećih tehnologija (...). Kada se te tehnologije kombiniraju u Ajax modelu, web aplikacije mogu izvršiti brza, inkrementalna ažuriranja korisničkog sučelja bez ponovnog učitavanja cijele stranice preglednika. Time aplikacija postaje brža i bolje reagira na radnje korisnika.«[49]

```
$(document).on('click','.cartButton', function(e){
    e.preventDefault();

    var quantity = $(this).closest('.product_data').find('.quantity-input').val();
    var product_id = $(this).val();

    $.ajax({
        method: "POST",
        url: "functions/cartFunction.php",
        data: {
            "product_id": product_id,
            "product_quantity": quantity,
            "scope": "add"
        },
        success: function(response){
            if(response == 201){
                alertify.success("Product added to cart");
            }
            else if(response == "exists"){
                alertify.success("Product already in cart");
            }
            else if(response == 401){
                alertify.success("Login to continue");
            }
            else if(response == 500){
                alertify.success("Something went wrong");
            }
        }
    });
});
```

Slika 90. JavaScript funkcija za dodavanje proizvoda u košaricu

Korištenjem jQuery Ajax metode zadana je POST metoda s URL adresom datoteke funkcija košarice (*cartFunction.php*) prema kojoj se šalje zahtjev. Potrebni podaci su identifikacijski broj proizvoda i količine proizvoda koji se dodaje u košarici, te je kao djelokrug (*scope*) napisano da se radi o dodavanju. Po uspješnom izvršenju Ajax zahtjeva, doći će do jednog od četiri moguća povratna poziva. Prvi, najosnovniji povratni poziv, odnosno poruka koja će se prikazati korisniku je da je proizvod uspješno dodan u

[49] MDN Web Docs, "Ajax", developer.mozilla.org/en-US/docs/Web/Guide/AJAX

košaricu. Druga mogućnost je da je proizvod koji se želi dodati već u košarici što onemogućuje još jedno dodavanje. Treća moguća poruka javlja neprijavljenom korisniku koji pokuša dodati proizvod u košaricu da je potrebno prijaviti se kako bi se moglo nastaviti. Četvrta moguća poruka koju korisnik može dobiti je da je došlo do greške prilikom izvršenja funkcije.

Spomenuta jQuery Ajax metoda poziva PHP funkciju koja je definirana u datoteci *cartFunction.php* (pod-mapa *functions*). U toj datoteci je definirana *switch* naredba u kojoj su svi mogući zadaci te funkcije napisani prema djelokrugu (*scope*) koji je definiran u jQuery Ajax metodi. U slučaju dodavanja proizvoda u košaricu (djelokrug *add*), deklariraju se varijable koje iz Ajax metode uzimaju podatke o identifikacijskom broju proizvoda i količini proizvoda. Potom se deklarira i nova varijabla koja uzima identifikacijski broj prijavljenog korisnika.

Za razliku od mobilne aplikacije koja koristi biblioteku TinyCart za dodavanje proizvoda u košaricu i sveobuhvatno baratanje košaricom, u web aplikaciji je košarica zapravo posebna tablica u bazi podataka (*carts*) u koju se dodaju proizvodi koji su trenutno u košarici te ostaju tamo ili do završetka narudžbe ili dok se ne uklone iz košarice. Po završavanju narudžbe, svi proizvodi u košarici se prebacuju z tablicu naručenih proizvoda (*order_items*) te košarica za određenog korisnika ostaje prazna do eventualnog kreiranja sljedeće narudžbe.

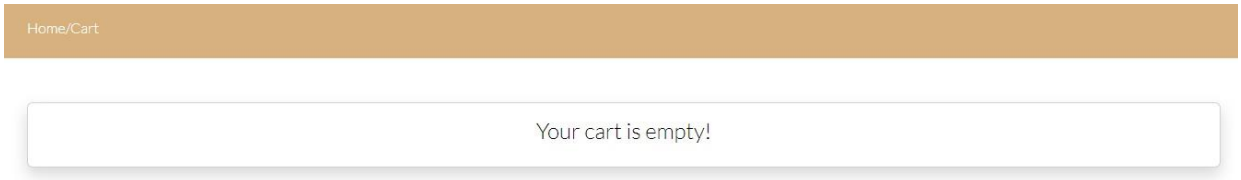
```
$scope = $_POST['scope'];
switch ($scope){
    case "add":
        $product_id = $_POST['product_id'];
        $product_quantity = $_POST['product_quantity'];

        $user_id = $_SESSION['auth_user']['user_id'];

        $check_cart = "SELECT * FROM carts WHERE product_id='$product_id' AND user_id='$user_id' ";
        $check_cart_run = mysqli_query($con, $check_cart);

        if(mysqli_num_rows($check_cart_run) > 0){
            echo "exists";
        }
        else{
            $insert_query = "INSERT INTO carts (user_id, product_id, product_quantity) VALUES ('$user_id','$product_id','$product_quantity')";
            $insert_query_run = mysqli_query($con, $insert_query);

            if($insert_query_run){
                echo 201;
            }
            else{
                echo 500;
            }
        }

        break;
```

Slika 91. PHP funkcija za dodavanje proizvoda u tablicu košarica (*carts*)

Korištenjem SQL naredbe SELECT provjerava se je li zadani proizvod već u košarici, odnosno tablici košarica te je li ga dodao isti korisnik (isti proizvod može biti više puta dodan u košaricu samo ako su dodani od strane različitih korisnika). Ako je to slučaj, PHP funkcija će ispisati poruku koja će javiti korisniku da se proizvod već nalazi u košarici. U suprotnom, koristi se SQL naredba INSERT INTO kojom se podatci o odabranom proizvodu dodaju u tablicu te se proizvod time dodaje u košaricu.

5.7. Košarica

Kao i u slučaju mobilne aplikacije, jedan od najvažnijih elemenata web aplikacije je baratanje s košaricom. U slučaju web aplikacije, samo prijavljeni korisnici imaju mogućnost dodavanja proizvoda u košaricu, što kasnije vodi i do mogućnosti završavanja i potvrde narudžbe. Ako korisnik odluči pregledati košaricu prije nego je barem jedan proizvod stavio u nju, stranica košarice *cart.php* će pokazati da je košarica prazna.



Slika 92. Prikaz prazne košarice (*cart.php*)

Jednom kada korisnik doda željeni proizvod ili više njih u košaricu, na stranici košarice će se izlistati svi proizvodi koji su od strane trenutno prijavljenog korisnika dodani u tablicu košarice. Proizvodi koji se nalaze u tablici košarice dohvaćaju se korištenjem funkcije *getCart* koja koristi varijablu čija je vrijednost identifikacijski broj prijavljenog korisnika, te SQL naredbu SELECT kako bi zajednički dohvatila podatke o proizvodu iz dvije tablice; tablice košarice (*carts*) i izvorne tablice proizvoda (*products*).

```
function getCart(){
    global $con;
    $userId = $_SESSION['auth_user']['user_id'];
    $query = "SELECT c.id as cid, c.product_id, c.product_quantity, p.id as pid, p.name, p.image, p.selling_price
                FROM carts c, products p WHERE c.product_id=p.id AND c.user_id='$userId' ORDER BY c.id DESC ";
    return $query_run = mysqli_query($con, $query);
}
```

Slika 93. Funkcija *getCart* (*userFunctions.php*)

Na sličan način kako su se u korisničkom sučelju prikazivale kategorije i proizvodi, implementacija stranice košarice bazira se na pozivanju funkcije *getCart* te korištenjem *foreach* naredbe koja će sve odabrane proizvode prikazati izlistane u korisničkom sučelju, uz mogućnosti promjene količine određenog proizvoda u košarici te brisanje određenog proizvoda iz košarice.

Slika 94. Prikaz košarice s dva dodana proizvoda (*cart.php*)

```
<div id="myCart">
    <?php
    $items = getCart();
    if(mysqli_num_rows($items) > 0){
        ?>
        <div class="row align-items-center">
            <div class="col-md-5">
                <h5>Product</h5>
            </div>
            <div class="col-md-2">
                <h6>Price</h6>
            </div>
            <div class="col-md-2">
                <h6>Quantity</h6>
            </div>
            <div class="col-md-3">
                <h6>Remove</h6>
            </div>
        </div>
        <div id="">
            <?php
                foreach ($items as $cart_item){
                ?>
                <div class="card product_data shadow-sm mb-3">
                    <div class="row align-items-center">
                        <div class="col-md-2">
                            <img src="uploads/<?= $cart_item['image'] ?>" alt="Image" width="80px">
                        </div>
                        <div class="col-md-3">
                            <h5><?= $cart_item['name'] ?></h5>
                        </div>
                        <div class="col-md-3">
                            <h5>€<?= $cart_item['selling_price'] ?></h5>
                        </div>
                        <div class="col-md-2">
                            <input type="hidden" class="productId" value="<?= $cart_item['product_id'] ?>">
                            <div class="input-group mb-3" style="width: 130px;">
                                <button class="input-group-text decrease-btn updateQuantity">-</button>
                                <input type="text" class="form-control text-center quantity-input bg-white" value="<?= $cart_item['product_quantity'] ?>" disabled>
                                <button class="input-group-text increase-btn updateQuantity">+</button>
                            </div>
                        </div>
                        <div class="col-md-2">
                            <button class="btn btn-danger btn-sm deleteItem" value="<?= $cart_item['cid'] ?>">
                            <i class="fa fa-trash me-2"></i>Remove</button>
                        </div>
                    </div>
                </div>
                <?php
                }
            ?>
        </div>
        <div class="float-end">
            <a href="checkout.php" class="btn btn-outline-domus">Checkout</a>
        </div>
```

Slika 95. Implementacija stranice košarice (*cart.php*)

U prethodno spomenutoj skripti *custom.js,* napisane su dodatne dvije funkcije koje provode metode za promjenu količine određenog proizvoda te za brisanje određenog proizvoda iz košarice. Funkcija za promjenu količine sastoji se od dvije varijable čije su vrijednosti trenutna količina i identifikacijski broj proizvoda, potom se na osnovu te dvije varijable provodi jQuery Ajax metoda koja poziva funkciju košarice iz već spomenute datoteke *cartFunction.php* sa zadanim djelokrugom (*scope*) za izmjenu (*update*).

Funkcija za brisanje proizvoda iz košarice u sebi ima deklariranu samo jednu varijablu čija je vrijednost identifikacijski broj košarice. Potom se na osnovu te varijable provodi jQuery Ajax metoda koja poziva PHP funkciju za brisanje proizvoda sa zadanim djelokrugom za brisanje (*delete*). Po uspješno izvršenom uklanjanju proizvoda iz košarice, aktivira se jQuery *load()* metoda koja će osvježiti dio korisničkog sučelja koji prikazuje sastav košarice. Ta metoda je važna zato što bi se bez nje brisanje iz košarice izvršilo uspješno, ali ta promjena ne bi bila vidljiva u korisničkom sučelju bez ručnog osvježavanja stranice. Na ovaj način je brisanje proizvoda iz košarice napravljeno znatno dinamičnije.

```
$(document).on('click','.updateQuantity', function(){
    var quantity = $(this).closest('.product_data').find('.quantity-input').val();
    var product_id = $(this).closest('.product_data').find('.productId').val();

    $.ajax({
        method: "POST",
        url: "functions/cartFunction.php",
        data: {
            "product_id": product_id,
            "product_quantity": quantity,
            "scope": "update"
        },
        success: function(response){
            //alert(response);
        }
    });
});

$(document).on('click', '.deleteItem', function(){
    var cart_id = $(this).val();
    //alert(cart_id);

    $.ajax({
        method: "POST",
        url: "functions/cartFunction.php",
        data: {
            "cart_id": cart_id,
            "scope": "delete"
        },
        success: function(response){
            if(response == 200){
                alertify.success("Product deleted");
                $('#myCart').load(location.href + " #myCart");
            }
            else{
                alertify.success(response);
            }
        }
    });
});
```

Slika 96. JavaScript funkcija za promjenu količine i brisanje proizvoda iz košarice

Kao što je i prethodno spomenuto, PHP funkcije koje se pozivaju preko jQuery Ajax metoda napisane su kao opcije *switch* naredbe u datoteci *cartFunction.php*. Ovisno o djelokrugu, aktivira se određena funkcija koja će izvršiti potrebne promjene u tablici košarice (*carts*). U slučaju djelokruga izmjene (*update*) koriste se varijable identifikacijskog broja proizvoda te varijabla vrijednosti trenutne količine, uz potrebnu varijablu čija je vrijednost identifikacijski broj prijavljenog korisnika. Korištenjem SQL naredbe SELECT se traži odgovarajući redu kojem identifikacijski brojevi proizvoda i prijavljenog korisnika odgovaraju vrijednostima varijabli. Potom se koristi SQL naredba UPDATE kojom se izvršava promjena količine u redu odabranog proizvoda u tablici košarice.

```
case "update":
    $product_id = $_POST['product_id'];
    $product_quantity = $_POST['product_quantity'];

    $user_id = $_SESSION['auth_user']['user_id'];

    $check_cart = "SELECT * FROM carts WHERE product_id='$product_id' AND user_id='$user_id' ";
    $check_cart_run = mysqli_query($con, $check_cart);

    if(mysqli_num_rows($check_cart_run) > 0){
        $update_query = "UPDATE carts SET product_quantity='$product_quantity' WHERE product_id='$product_id' AND user_id='$user_id' ";
        $update_query_run = mysqli_query($con, $update_query);
        if($update_query_run){
            echo 200;
        }
        else{
            echo 500;
        }
    }
    else{
       echo "something went wrong";
    }

    break;

case "delete":
    $cart_id = $_POST['cart_id'];

    $user_id = $_SESSION['auth_user']['user_id'];

    $check_cart = "SELECT * FROM carts WHERE id='$cart_id' AND user_id='$user_id' ";
    $check_cart_run = mysqli_query($con, $check_cart);

    if(mysqli_num_rows($check_cart_run) > 0){
        $delete_query = "DELETE FROM carts WHERE id='$cart_id'  ";
        $delete_query_run = mysqli_query($con, $delete_query);
        if($delete_query_run){
            echo 200;
        }
        else{
            echo "something went wrong";
        }
    }
    else{
       echo "something went wrong";
    }

    break;

    default:
        echo 500;
```

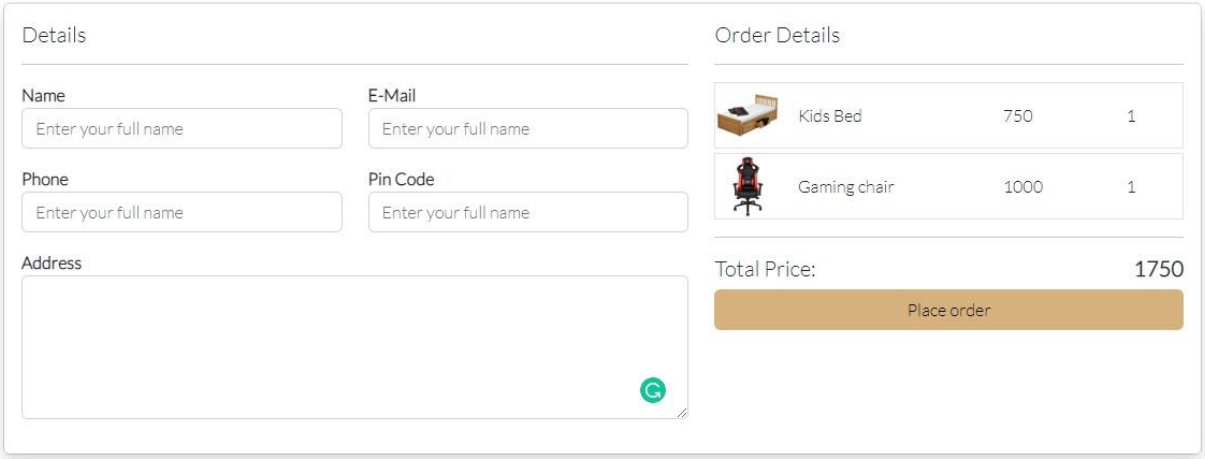
Slika 97. PHP funkcija za izmjenu količine i brisanje proizvoda iz košarice (*cartFunction.php*)

PHP funkcija za brisanje proizvoda iz košarice koristi varijablu čija je vrijednost identifikacijski broj košarice. Korištenjem te vrijednosti se preko SELECT naredbe traži red u tablici košarice u kojem identifikacijski broj reda odgovara vrijednosti varijable, te se dodatno provjerava da je identifikacijski broj korisnika u tom redu jednak vrijednosti varijable čija je vrijednost identifikacijski broj trenutno prijavljenog korisnika. Po pronalasku odgovarajućeg reda, koristi se SQL naredba DELETE koja, shodno svom imenu, će obrisati traženi red iz tablice.

Jednom kada je korisnik siguran u sadržaj košarice, klikom na gumb za završavanje narudžbe (*checkout*), dolazi do završnog koraka u stvaranju narudžbe.

5.8. Završavanje narudžbe

Završavanje narudžbe korisnik provodi jednom kada je siguran da je sadržaj košarice u skladu s njegovim željama. Klikom na ponuđeni gumb korisnik odlazi na stranicu za završavanje narudžbe *checkout.php.* Korisničko sučelje te stranica je dizajnirano kao kartični kontejner koji sadrži obrazac u koji korisnik upisuje svoje osobne podatke potrebne za dovršavanje narudžbe. Dodatno, desna strana tog kartičnog prikaza sadrži detalje o narudžbi, uključujući odabrane proizvode, njihovu cijenu, i količinu. Konačno, korisniku se pokazuje ukupna cijena cijele narudžbe te gumb za potvrdu narudžbe.



Slika 98. Prikaz korisničkog sučelja u stranici za potvrdu narudžbe (*checkout.php*)

Korisničko sučelje je implementirano korištenjem Bootstrap obrazaca koji su se također prethodno koristili za stvaranje obrasca prijave i registracije. Detalji narudžbe, koji obuhvaćaju prikaz proizvoda i košarice su implementirani na isti način kao i korisničko sučelje u stranici košarice. Ponovno je korišteno pozivanje funkcije *getCart,* s tim da je deklarirana dodatna varijabla čija će vrijednost biti ukupna cijena narudžbe. Izlistavanje proizvoda u košarici u korisničkom sučelju se ponovno implementira korištenjem naredbe *foreach.* Pritiskom na gumb za pohranu narudžbe aktivira se vanjska PHP funkcija u datoteci *orderFunctions.php.*

PHP funkcija za pohranu narudžbe prvo definira varijable čije će vrijednosti postati podaci koje je korisnik upisao u spomenuti obrazac. Prisutna je *if* naredba koja kao uvjet postavlja da sva polja u obrascu moraju biti popunjena inače se narudžba neće moći završiti. Potom se deklarira varijabla čija je vrijednost identifikacijski broj korisnika, te se nakon toga provodi naredba SELECT kojom se uzimaju potrebni proizvodi te petlja *foreach* koja prolazi kroz te proizvode kako bi došla do vrijednosti ukupne cijene koja se sprema u zasebnu varijablu. Nakon toga, deklarira se nova varijabla za praćenje narudžbe (*tracking_id*), i to na način da joj vrijednost sadrži naziv "domusorder", uz dodatak nasumičnog broja između 1111 i 9999 i jednog dijela telefonskog broja korisnika koji se izvlači funkcijom *substr.* Jednom kada sve te varijable dobiju svoje potrebne vrijednosti, koristi se naredba INSERT INTO kojom se podaci o korisniku i općeniti podaci o narudžbi spremaju u tablicu narudžbi (*orders*).

Korištenjem *if* petlje provjerava se je li stvaranje novog reda u tablici narudžbi bilo uspješno, ukoliko je, prelazi se na drugi dio funkcije u kojoj je deklarirana varijabla čija je vrijednost identifikacijski broj upravo spremljene narudžbe, te korištenjem naredbe *foreach* se prolazi kroz svaki proizvod koji je dio te narudžbe te se naredbom INSERT INTO svaki od tih proizvoda sprema u zasebni red u tablici naručenih proizvoda *(order_items*), uz pripadajući identifikacijski broj narudžbe.

```
if (isset($_SESSION['auth'])) {

    if (isset($_POST['placeOrder'])) {

        $name = mysqli_real_escape_string($con, $_POST['name']);
        $email = mysqli_real_escape_string($con, $_POST['email']);
        $phone = mysqli_real_escape_string($con, $_POST['phone']);
        $pin = mysqli_real_escape_string($con, $_POST['pin']);
        $address = mysqli_real_escape_string($con, $_POST['address']);
        $payment_mode = mysqli_real_escape_string($con, $_POST['payment_mode']);
        $payment_id = mysqli_real_escape_string($con, $_POST['payment_id']);


        if ($name == "" || $email == "" || $phone == "" || $pin == "" || $address == "") {
            $_SESSION['message'] = "Make sure all fields are filled";
            header('Location: ../checkout.php');
            exit(0);
        }

        $userId = $_SESSION['auth_user']['user_id'];
        $query = "SELECT c.id as cid, c.product_id, c.product_quantity, p.id as pid, p.name, p.image, p.selling_price
                    FROM carts c, products p WHERE c.product_id=p.id AND c.user_id='$userId' ORDER BY c.id DESC ";

        $query_run = mysqli_query($con, $query);

        $totalPrice = 0;
        foreach ($query_run as $cart_item) {
            $totalPrice += $cart_item['selling_price'] * $cart_item['product_quantity'];
        }

        $tracking_id = "domusorder" . rand(1111,9999).substr($phone,2);
        $insert_query = "INSERT INTO orders (tracking_id, user_id, name, email, phone, address, pin, total_price, payment_mode, payment_id)
                        VALUES ('$tracking_id', '$userId', '$name', '$email', '$phone', '$address', '$pin', '$totalPrice', '$payment_mode', '$payment_id')";
        $insert_query_run = mysqli_query($con, $insert_query);

        if($insert_query_run){
            $order_id = mysqli_insert_id($con);
            foreach ($query_run as $cart_item) {
                $product_id = $cart_item['product_id'];
                $product_quantity = $cart_item['product_quantity'];
                $price = $cart_item['selling_price'];

                $insert_items_query = "INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
                ('$order_id','$product_id','$product_quantity','$price')";
                $insert_items_query_run = mysqli_query($con, $insert_items_query);

                $product_query = "SELECT * FROM products WHERE id='$product_id' LIMIT 1";
                $product_query_run = mysqli_query($con, $product_query);

                $productData = mysqli_fetch_array($product_query_run);
                $find_quantity = $productData['quantity'];

                $new_quantity = $find_quantity - $product_quantity;

                $updateQty_query = "UPDATE products SET quantity='$new_quantity' WHERE id='$product_id' ";
                $updateQty_query_run = mysqli_query($con, $updateQty_query);
            }

            $deleteCartQuery = "DELETE FROM carts WHERE user_id = '$userId'";
            $deleteCartQuery_run = mysqli_query($con, $deleteCartQuery);

            $_SESSION['message'] = "Order successfully placed!";
            header('Location: ../myOrders.php');
            die();
        }
    }
} else {
    header('Location: ../index.php');
}
```

Slika 99. Implementacija PHP funkcije za pohranu narudžbe (*orderFunctons.php*)

Po izvršenju spremanja potrebnih podataka u tablicu narudžbi i tablicu naručenih proizvoda, ponovno će se upotrijebiti SELECT naredba u tablici proizvoda (*products*), kako bi se pronašla količina proizvoda koja je trenutno na stanju te kako bi se ta količina smanjila za onu količinu koja je upravo naručena, jer naručena roba se automatski po potvrdi narudžbe miče sa zaliha, tada se vrši naredba UPDATE koja sprema novi podatak o količini proizvoda na zalihi. I konačno, po potvrdi narudžbe, svi proizvodi iz tablice košarice (*carts*) koje je odabrao taj određeni korisnik se brišu jer su sada svi proizvodi iz košarice prebačeni u tablicu naručenih proizvoda.

5.9. Pregled vlastitih narudžbi

Dok administrator ima mogućnost pregledati sve narudžbe koje su stvorili svi korisnici, sam korisnik ima mogućnost pregledati narudžbe koje je samo on prethodno stvorio. Stranica za pregled vlastitih narudžbi *myOrders.php* implementirana je korištenjem Bootstrap tablice[50] te je ponovno korištena naredba *foreach* preko koje su izlistane sve narudžbe koje je u tom trenutnu prijavljeni korisnik stvorio do tada.

Ključni dio implementacije stranice za pregled vlastitih narudžbi je pozivanje vanjske PHP funkcije *getOrders* (*userFunctions.php*) koja koristi varijablu, čija je vrijednost identifikacijski broj korisnika, koji uvjetuje naredbi SELECT da odabire specifične redove iz tablice narudžbi (*orders*) kako bi korisnik imao uvid samo u one narudžbe koje je on stvorio.

```
function getOrders(){
    global $con;
    $userId = $_SESSION['auth_user']['user_id'];

    $query = "SELECT * FROM orders WHERE user_id='$userId' ORDER BY id DESC";
    return $query_run = mysqli_query($con, $query);
}
```

Slika 100. PHP funkcija za dohvat narudžbi određenog korisnika

Stranica za pregled vlastitih narudžbi prikazuje tablicu koja sadrži četiri osnovne informacije o narudžbama; identifikacijski broj narudžbe, kod za praćenje narudžbe, ukupnu cijenu, i datum stvaranja narudžbe. No, postoji i peti stupac u kojem se nalazi gumb čijim klikom korisnik može dobiti sve detalje narudžbe na posebnoj stranici *viewOrder.php*.

ID	Tracking ID	Price	Date	View
2	domusorder4785579631	1750	2022-11-13 15:36:30	Details
1	domusorder566150173	2720	2022-08-27 14:40:08	Details

Slika 101. Prikaz vlastitih narudžbi (*myOrders.php*)

[50] Bootstrap, "Tables", getbootstrap.com/docs/4.1/content/tables

Klikom na odabrani gumb u stranici pregleda vlastitih narudžbi otvara se stranica *viewOrder.php* čija URL adresa sadrži *slug* koji je u ovom slučaju kod za praćenje te narudžbe. Taj kod za praćenje koji se nalazi u URL adresu je vrlo važan jer i implementacija te stranica započinje korištenjem globalne varijable *$_GET* koja će iskoristiti taj kod za praćenje kao atribut prilikom pozivanja vanjske PHP funkcije *checkTracking* koja je definirana u datoteci korisničkih funkcija (*userFunctions.php*). Ta funkcija u sebi ima deklariranu i varijablu identifikacijskog broja korisnika koja se zajedno s kodom za praćenje koristi kao uvjet SELECT naredbi kako bi odabrala pravi redak iz tablice narudžbi (*orders*), čije će podatke morati prikazati u korisničkom sučelju.

```php
function checkTracking($trackingId){
    global $con;
    $userId = $_SESSION['auth_user']['user_id'];

    $query = "SELECT * FROM orders WHERE tracking_id='$trackingId' AND user_id='$userId' ";
    return mysqli_query($con, $query);
}
```

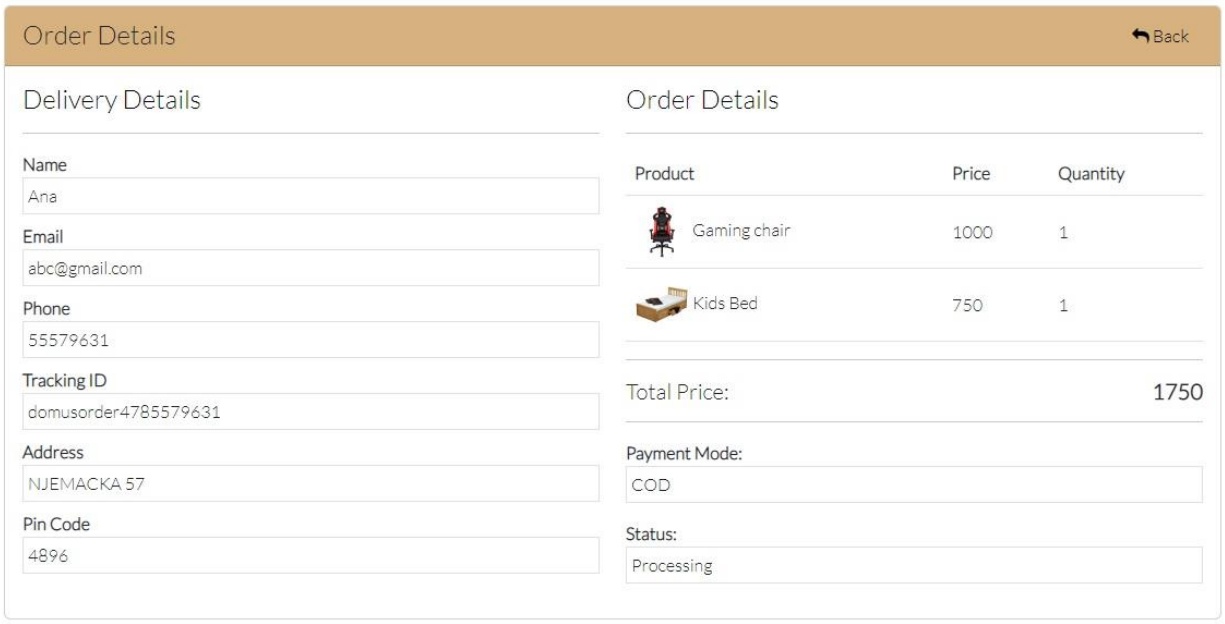
Slika 102. Funkcija *checkTracking* (*userFunctions.php*)

Korisničko sučelje stranice za pregled određene vlastite narudžbe (*viewOrder.php*) dizajnirano je također korištenjem Bootstrapa. Dok se funkcija *checkTracking* koristi za dohvat podataka iz tablice narudžbi (*orders*), važno je dohvatiti i podatke iz tablice naručenih proizvoda (*order_items*), za što se koristila SELECT naredba koja je kombinirano pretraživala tri tablice (*orders, order_items, products*) kako bi korištenjem naredbe *foreach* pravilno izlistala sve proizvode koji spadaju pod određenu narudžbu.

```php
<?php
    $userId = $_SESSION['auth_user']['user_id'];
    $order_query = "SELECT o.id as oid, o.tracking_id, o.user_id, oi.*, oi.quantity as oquantity, p.* FROM  orders o, order_items oi,
        products p WHERE o.user_id='$userId' AND oi.order_id=o.id AND p.id=oi.product_id
        AND o.tracking_id='$tracking_id' ";
    $order_query_run = mysqli_query($con, $order_query);

    if(mysqli_num_rows($order_query_run) > 0){
        foreach($order_query_run as $item){
            ?>
                <tr>
                    <td class="align-middle">
                        <img src="uploads/<?= $item['image']; ?>" width="50px" height="50px" alt="<?= $item['name']; ?>">
                        <?= $item['name']; ?>
                    </td>
                    <td class="align-middle">
                        <?= $item['price']; ?>
                    </td>
                    <td class="align-middle">
                        <?= $item['oquantity']; ?>
                    </td>
                </tr>
            <?php
        }
    }
?>
```

Slika 103. Dohvat naručenih proizvoda (*viewOrder.php*)

S obzirom na to da se jednom pohranjena narudžba više ne može izmijeniti, stranica za pregled detalja određene narudžbe može se isključivo i samo pregledavati. Jedino što korisnik može je vratiti se natrag na stranicu pregleda narudžbi klikom na gumb za povratak u gornjem desnom kutu kontejnera u kojem su prikazane informacije o narudžbi.



Slika 104. Pregled detalja narudžbe (*viewOrder.php*)

5.10. Odjava korisnika

Prijavljeni korisnik se po završetku rada s web aplikacijom može odjaviti iz sustava. PHP skripta za odjavu korisnika *logout.php* je jedan od najjednostavnijih napisanih programa u sjedištu web aplikacije, služeći se metodom *unset* kojom se korisnikova prijava u sustav prekida. Potom se prikazuje korisniku poruka da je uspješno odjavljen, te ga preusmjerava na naslovnu stranicu web aplikacije, odakle je rad i počeo.

```php
<?php
session_start();

    if(isset($_SESSION['auth'])){
        unset($_SESSION['auth']);
        unset($_SESSION['auth_user']);
        $_SESSION['message'] = "Logged out";
    }


    header('Location: index.php');

?>
```

Slika 105. Implementacija PHP funkcije za odjavu korisnika (*logout.php*)

6. IMPLEMENTACIJA ADMINISTRATORSKOG DIJELA WEB APLIKACIJE

Administratorski dio web aplikacije je dio web aplikacije kojem može pristupiti samo korisnik s administratorskim privilegijama. Sve datoteke i komponente administratorskog dijela web aplikacije su spremljene u mapi *admin,* dok ta mapa sadrži samo dvije pod-mape (*assets* i *includes,* koje su specifične samo za administratorski dio aplikacije, za razliku od istoimenih mapa u korisničkom dijelu, stranica 51.), te ukupno 13 datoteka od kojih su 12 stranice, dok je samo jedna skripta koja sadrži potrebne PHP funkcije.

6.1. Zaglavlje, izbornici, podnožje

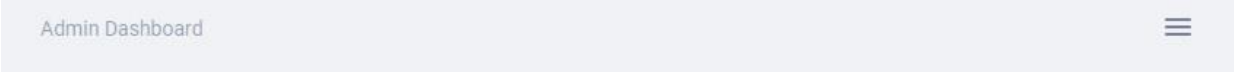
Kao i u slučaju korisničkog dijela web aplikacije, i administratorski dio sadrži stalne komponente koje su definirane u zasebnim datotekama te se kasnije samo umeću u stranice korištenjem PHP naredbe *include.* Administratorski dio aplikacije također sadrži tri glavne komponente kao i korisnički dio, uz dodatnu komponentu koja je definirana u datoteci bočnog izbornika (*sidebar.php*). Zaglavlje se i u administratorskom dijelu aplikacije sastoji od naslova koje je u ovom slučaju *Domus Admin Panel,* te su umetnute poveznice na vanjske CSS datoteke koje omogućuju korištenje biblioteke Google Fonts, te korištenje biblioteke AlertifyJS kakve su korištene i u zaglavlju korisničkog dijela web aplikacije.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>
        Domus Admin Panel
    </title>
    <!--     Fonts and icons     -->
    <link rel="stylesheet" type="text/css" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700,900|Roboto+Slab:400,700" />

    <!-- Font Awesome Icons -->
    <script src="https://kit.fontawesome.com/42d5adcbca.js" crossorigin="anonymous"></script>
    <!-- Material Icons -->
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons+Round" rel="stylesheet">
    <!-- CSS Files -->
    <link id="pagestyle" href="assets/css/material-dashboard.min.css" rel="stylesheet" />

    <!-- AlertifyJS -->
    <link rel="stylesheet" href="//cdn.jsdelivr.net/npm/alertifyjs@1.13.1/build/css/alertify.min.css"/>
    <link rel="stylesheet" href="//cdn.jsdelivr.net/npm/alertifyjs@1.13.1/build/css/themes/bootstrap.min.css"/>


    <style>
        .form-control, .form-select{
            border: 1px solid black !important;
            padding: 8px 10px;
        }
    </style>

</head>

<body class="g-sidenav-show  bg-gray-200">
    <?php include('sidebar.php') ?>
    <main class="main-content position-relative max-height-vh-100 h-100 border-radius-lg ">
    <?php include('navbar.php') ?>
```

Slika 106. Implementacija zaglavlja u administratorskom dijelu web aplikacije (*header.php*)

Na dnu skripte kojom se implementira zaglavlje korisničkog dijela web aplikacije nalaze se dvije PHP naredbe include koje umeću bočnu traku, odnosno bočni izbornik, te gornju navigacijsku traku ovog dijela web aplikacije.

Iako je implementirana korištenjem Bootstrap oznake *nav*[51], za razliku od one u korisničkom dijelu aplikacije, ovdje nije riječ o klasičnoj navigacijskoj traci, već samo o dijelu korisničkog sučelja aplikacije koji sadrži link za povratak na naslovnu stranicu administratorskog dijela aplikacije te ikonicu koja služi za otvaranje bočnog izbornika kada je, zbog responzivnosti, prozor web pretraživača dovoljno sužen da bočni izbornik nije vidljiv.



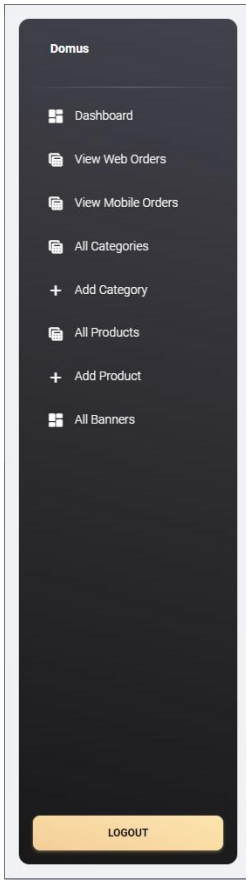
Slika 107. Navigacijska traka u administratorskom dijelu aplikacije u responzivnom načinu rada

```
<nav class="navbar navbar-main navbar-expand-lg px-0 mx-4 shadow-none border-radius-xl" id="navbarBlur" data-scroll="true">
  <div class="container-fluid py-1 px-3">
    <nav aria-label="breadcrumb">

      <ol class="breadcrumb bg-transparent mb-0 pb-0 pt-1 px-0 me-sm-6 me-5">
        <li class="breadcrumb-item text-sm"><a class="opacity-5 text-dark" href="index.php">Admin Dashboard</a></li>
      </ol>

    </nav>
    <div class="collapse navbar-collapse mt-sm-0 mt-2 me-md-0 me-sm-4" id="navbar">
      <div class="ms-md-auto pe-md-3 d-flex align-items-center">
      </div>
      <ul class="navbar-nav  justify-content-end">
        <li class="nav-item d-xl-none ps-3 d-flex align-items-center">
          <a href="javascript:;" class="nav-link text-body p-0" id="iconNavbarSidenav">
            <div class="sidenav-toggler-inner">
              <i class="sidenav-toggler-line"></i>
              <i class="sidenav-toggler-line"></i>
              <i class="sidenav-toggler-line"></i>
            </div>
          </a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

Slika 108. Implementacija navigacijske trake u administratorskom dijelu web aplikacije (*navbar.php*)

Bočni izbornik (*sidebar.php*) je vrsta vertikalne navigacijske trake preko koje se može pristupiti svim stranicama i mogućnostima administratorskog dijela aplikacije. Sastoji se od osam glavnih linkova i dva sporedna od kojih jedan omogućuje korisniku da se vrati u korisnički dio aplikacije (klikom na naslov), dok drugi služi kao gumb za odjavu korisnika iz sustava. Spomenutih osam linkova koji se nalaze u glavnom dijelu bočnog izbornika zasebno vode na određene stranice unutar administratorskog dijela aplikacije, dok najgornji (link *Dashboard*) vraća administratora na naslovnu stranicu administratorskog dijela aplikacije.

[51] Bootstrap, "Navbar", getbootstrap.com/docs/4.0/components/navbar



Slika 109. Bočni izbornik admin. dijela aplikacije

```
<aside class="sidenav navbar navbar-vertical navbar-expand-xs border-0 border-radius-xl my-3 fixed-start ms-3   bg-gradient-dark" id="sidenav-main">
  <div class="sidenav-header">
    <i class="fas fa-times p-3 cursor-pointer text-white opacity-5 position-absolute end-0 top-0 d-none d-xl-none" aria-hidden="true" id="iconSidenav"></i>
    <a class="navbar-brand m-0" href="../index.php">
      <span class="ms-1 font-weight-bold text-white">Domus</span>
    </a>
  </div>
  <hr class="horizontal light mt-0 mb-2">
  <div class="collapse navbar-collapse  w-auto  max-height-vh-100" id="sidenav-collapse-main">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link <?= $page == "index.php"? 'active':''; ?>" href="index.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">dashboard</i>
          </div>
          <span class="nav-link-text ms-1">Dashboard</span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white <?= $page == "orders.php"? 'active':''; ?>" href="orders.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">table_view</i>
          </div>
          <span class="nav-link-text ms-1">View Web Orders</span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white <?= $page == "mobileOrders.php"? 'active':''; ?>" href="mobileOrders.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">table_view</i>
          </div>
          <span class="nav-link-text ms-1">View Mobile Orders</span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white <?= $page == "category.php"? 'active':''; ?>" href="category.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">table_view</i>
          </div>
          <span class="nav-link-text ms-1">All Categories</span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white <?= $page == "addCategory.php"? 'active':''; ?>" href="addCategory.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">add</i>
          </div>
          <span class="nav-link-text ms-1">Add Category</span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white <?= $page == "products.php"? 'active':''; ?>" href="products.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">table_view</i>
          </div>
          <span class="nav-link-text ms-1">All Products</span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white <?= $page == "addProduct.php"? 'active':''; ?>" href="addProduct.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">add</i>
          </div>
          <span class="nav-link-text ms-1">Add Product</span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-white <?= $page == "banners.php"? 'active':''; ?>" href="banners.php">
          <div class="text-white text-center me-2 d-flex align-items-center justify-content-center">
            <i class="material-icons opacity-10">dashboard</i>
          </div>
          <span class="nav-link-text ms-1">All Banners</span>
        </a>
      </li>
    </ul>
  </div>
  <div class="sidenav-footer position-absolute w-100 bottom-0 ">
    <div class="mx-3">
      <a class="btn bg-gradient-domus mt-4 w-100" href="../logout.php">
        Logout</a>
    </div>
  </div>
</aside>
```

Slika 110. Implementacija bočnog izbornika u administratorskom dijelu aplikacije (*sidebar.php*)

Slično kao u slučaju korisničkog dijela aplikacije, i podnožje administratorskog dijela web aplikacije se sastoji od umetnutih vanjskih skripti napisanih u programskom jeziku JavaScript. Prve dvije oznake *script* koriste se za umetanje vanjskih datoteka *jquery-3.6.0.min.js* i *bootstrap.bundle.min.js,* koje se pod istim licencama kao i jQuery (stranica 55.) i Bootstrap (stranica 53.) datoteke u podnožju korisničkog dijela aplikacije.

```
<footer class="footer pt-5">
    <div class="container-fluid">
        <div class="row align-items-center justify-content-lg-between">
            <div class="col-lg-12 text-center">
                Luka Abramović, Diplomski Rad, FIPU, 2022
            </div>
        </div>
    </div>
</footer>
</main>

<script src="assets/js/jquery-3.6.0.min.js"></script>
<script src="assets/js/bootstrap.bundle.min.js"></script>
<script src="assets/js/smooth-scrollbar.min.js"></script>
<script src="assets/js/perfect-scrollbar.min.js"></script>

<script src="assets/js/material-dashboard.min.js"></script>


<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>

<script src="assets/js/custom.js"></script>


<!-- AlertifyJS -->
<script src="//cdn.jsdelivr.net/npm/alertifyjs@1.13.1/build/alertify.min.js"></script>

<script>

    <?php
    if(isset($_SESSION['message'])){
    ?>
    alertify.set('notifier', 'position', 'top-center');
    alertify.success('<?= $_SESSION['message']; ?>');
    <?php
    unset($_SESSION['message']);
    }
    ?>
</script>

</body>

</html>
```

Slika 111. Implementacija podnožja administratorskog dijela web aplikacije (*footer.php*)

Preostale tri *script* oznake koriste se za umetanje tri vanjske datoteke (*smooth-scrollbar.min.js, perfect-scrollbar.min.js, material-dashboard.min.js*)[52] koje zajednički daju vizualni identitet i funkcionalnost bočnom izborniku u administratorskom dijelu aplikacije. Te tri korištene datoteke su pod MIT licencom[53] koja omogućuje korištenje, modifikaciju, i distribuciju spomenutih datoteka.

U podnožju je također umetnuta oznaka za vanjsku datoteku SweetAlert[54] koja je vizualno atraktivni element koji se koristi za prikazivanje dijaloških prozora s pitanjem za potvrdu određene akcije (npr. prilikom brisanja određenog proizvoda ili kategorije iz tablice u bazi podataka).

Podnožje ktome sadrži i oznaku kojom je umetnuta vanjska datoteka koja omogućuje korištenje biblioteke AlertifyJS (stranica 53.) koja je jednako definirana i u podnožju korisničkog dijela aplikacije (zajedno s kratkom funkcijom koja poziva spomenutu biblioteku za prikazivanje određene poruke korisniku. Konačno, u podnožju je definirana i oznaka koja umeće posebnu datoteku *custom.js* u kojoj su definirane dodatne JavaScript funkcije potrebne za rad administratorskog dijela aplikacije.

6.2. Administratorska ploča

Naslovna stranica administratorskog dijela web aplikacije, ili administratorska ploča, je glavna stranica u dijelu aplikacije kojem može pristupiti isključivo administrator. S obzirom da je administratorski dio aplikacije mapom odvojen od korisničkog, i ova naslovna

[52] Creative Tim, "material-dashboard", github.com/creativetimofficial/material-dashboard/tree/master/assets/js
[53] Creative Tim, "End User License Agreement", creative-tim.com/license
[54] Tristan Edwards, "SweetAlert", sweetalert.js.org/guides

stranica se u upravitelju datotekama zove *index.php.* Stranica sadrži opcije i mogućnosti koje korisnik može obavljati u radu s web aplikacijom, a to su sljedeće:

- Pregledavanje narudžbi koje je korisnik stvorio u web aplikaciji
- Pregledavanje narudžbi koje je korisnik stvorio u mobilnoj aplikaciji
- Pregledavanje (uz mogućnost izmjene) postojećih kategorija
- Dodavanje nove kategorije
- Pregledavanje (uz mogućnost izmjene) postojećih proizvoda
- Dodavanje novog proizvoda
- Rad s naslovnim fotografijama (dodavanje novih ili brisanje postojećih)

Kod implementacije bilo koje stranice u administratorskom dijelu aplikacije je bitno za napomenuti da one odmah prilikom učitavanja koriste PHP naredbu *include* koja umeće funkcije datoteke *adminMiddle.php,* kojoj je primarna zadaća ne samo sprječavanje standardnog korisnika da uđe u administratorski dio aplikacije, nego i koristi naredbu *include* kako bi došao do potrebnih funkcija koje su definirane u datoteci *functions.php.* Drugim riječima, naredbom *include* se u stranicu administratorskog dijela aplikacije umeće sastav datoteke *adminMiddle.php,* u čijem je sastavu pak druga naredba *include* koja umeće sastav datoteke funkcija *functions.php*. Time je moguće preko administratorskih stranica koristiti te funkcije po principu druge ruke, odnosno nasljeđivanja. Na isti način, u administratorske stranice se naredbom *include* umeću samo datoteke zaglavlja i podnožja, jer datoteka zaglavlja je ta koja koristi istu tu naredbu da u sebe, a samim time i u stranicu, umetne datoteke navigacijske trake i bočnog izbornika.

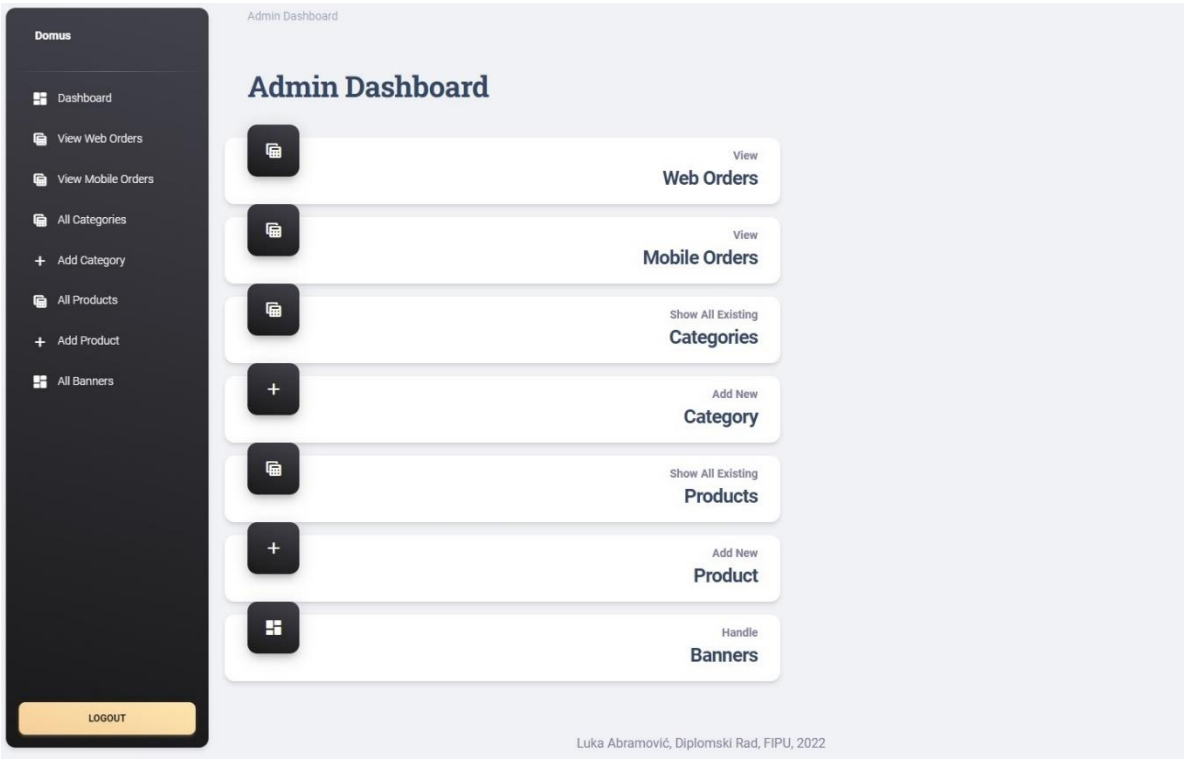
Slika 112. Korištenje PHP naredbe *include* u stranici administratorskog dijela aplikacije

Slika 113. Implementacija datoteke *adminMiddle.php*

Korisničko sučelje svake stranice administratorskog dijela aplikacije implementirano je korištenjem Bootstrap elemenata na gotovo jednak način kao i u korisničkom dijelu aplikacije. U slučaju administratorske ploče, definiran je kontejner čiji sastav sadrži nekoliko ugniježđenih *div-ova* od kojih svaki ima svoju ulogu u definiranju veličine ili pozicije u korisničkom sučelju, dok onaj najdublji *div* u sebi sadrži naslov administratorske ploče. Ispod toga, definiran je poseban *div* reda (*row*), u kojem su u zasebnim kartičnim pogledima zasebno definirane sve opcije, odnosno linkovi koji vode na ostale administratorske stranice u aplikaciji.

```
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <div class="row mt-4">
                <div class="col-lg-7 position-relative z-index-2">
                    <div class="card card-plain mb-4">
                        <div class="card-body p-3">
                            <div class="row">
                                <div class="col-lg-6">
                                    <div class="d-flex flex-column">
                                        <h2 class="font-weight-bolder mb-0">Admin Dashboard</h2>
                                    </div>

                                </div>
                            </div>
                        </div>
                    </div>

                    <div class="row">

                            <div class="card  mb-3">
                                <div class="card-header p-3 pt-2">
                                    <div class="icon icon-lg icon-shape bg-gradient-dark shadow-dark shadow text-center border-radius-xl mt-n4 position-absolute">
                                        <a href="orders.php"><i class="material-icons opacity-10">table_view</i></a>
                                    </div>
                                    <div class="text-end pt-1">
                                        <p class="text-sm mb-0 text-capitalize font-weight-bold">View</p>
                                        <a href="orders.php"><h4 class="mb-0">Web Orders</h4></a>
                                    </div>
                                </div>
                            </div>

                            <div class="card  mb-3">
                                <div class="card-header p-3 pt-2">
                                    <div class="icon icon-lg icon-shape bg-gradient-dark shadow-dark shadow text-center border-radius-xl mt-n4 position-absolute">
                                        <a href="mobileOrders.php"><i class="material-icons opacity-10">table_view</i></a>
                                    </div>
                                    <div class="text-end pt-1">
                                        <p class="text-sm mb-0 text-capitalize font-weight-bold">View</p>
                                        <a href="mobileOrders.php"><h4 class="mb-0">Mobile Orders</h4></a>
                                    </div>
                                </div>
                            </div>
```

Slika 114. Implementacija administratorske ploče (*index.php*)



Slika 115. Naslovna stranica administratorske ploče

6.3. Pregled narudžbi

Već je više puta prethodno istaknuto da administrator preko administratorske ploče ima mogućnost pregledavati sve narudžbe koje su korisnici stvorili. Narudžbe stvorene u mobilnoj i web aplikaciji spremljene su u različitim tablicama i shodno tome se te narudžbe i pregledavaju koristeći dvije različite stranice u administratorskom dijelu aplikacije. No, bez obzira na to, obje stranice, za prikaz mobilnih i web narudžbi, implementirane su i prikazane u korisničkom sučelju na gotovo istovjetan način.

```
function getAllOrders($table){
    global $con;
    $query = "SELECT * FROM $table";
    return $query_run = mysqli_query($con, $query);
}
```

Slika 116. Funkcija za dohvat svih narudžbi iz tablice (*functions.php*)

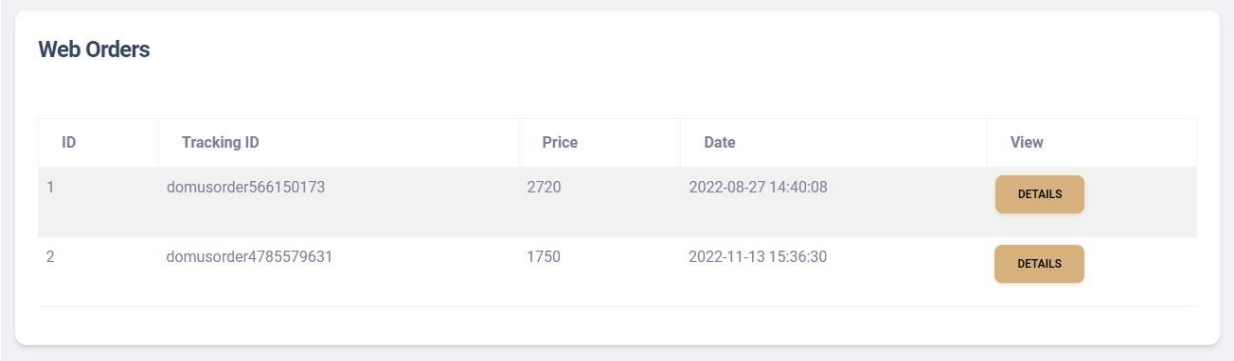
Prilikom implementacije stranica narudžbi (*orders.php* i *mobileOrders.php*), nužno je da se na početku koristi PHP naredba *include* za umetanje sadržaja datoteke *adminMiddle.php,* a shodno tome i funkcije koji se nalaze u datoteci *functions.php.* Obje stranice pozivaju funkciju *getAllOrders* koja je vrlo slična onoj funkciji koju smo koristili za dohvat svih narudžbi određenog korisnika u korisničkom dijelu aplikacije, uz razliku da se naredba SELECT sada ne ograničava na identifikacijski broj korisnika, nego dohvaća apsolutno sve redove u tablici. Važno je samo definirati ime tablice kao atribut (*orders* u slučaju web aplikacije i *mobile_orders* u slučaju mobilne aplikacije) prilikom definiranja funkcije kako bi varijabla *table* dobila vrijednost imena tražene tablice i prema tome SELECT naredba zna iz koje tablice treba dohvatiti podatke.

Korisnička sučelja stranice web narudžbi i stranice mobilnih narudžbi implementirani su na gotovo jednak način uz tek minimalne, ali ne zanemarive razlike. Korisničko sučelje je i u ovom slučaju dizajnirano korištenjem Bootstrap elemenata, dok je ključna razlika, osim u atributu prilikom pozivanja funkcije *getAllOrders* i ta što korisničko sučelje web narudžbe u tablici prikazuje identifikacijski broj, kod za praćenje narudžbe, ukupnu cijenu i datum kreiranja narudžbe, dok se u slučaju mobilnih narudžbi prikazuje identifikacijski broj, ime korisnika, email, korisnika, adresa i ukupna cijena. Tablice u obje stranice imaju u zadnjem stupcu gumb s kojim se može pristupiti detaljima pojedine narudžbe, uz razliku što se za raspoznavanje određene narudžbe u slučaju web narudžbe koristi kod za praćenje narudžbe, dok se u slučaju mobilne narudžbe koristi identifikacijski broj.

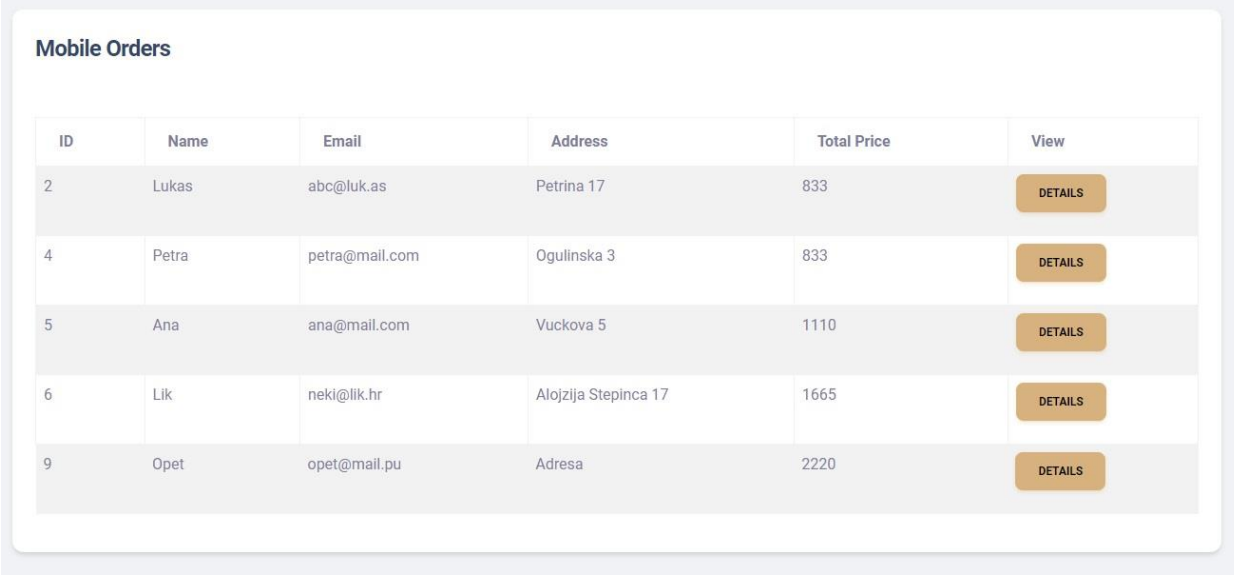
```
<?php
$orders = getAllOrders("orders");

if(mysqli_num_rows($orders) > 0){
    foreach($orders as $item){
    ?>
        <tr>
            <td> <?= $item['id']; ?> </td>
            <td> <?= $item['tracking_id']; ?> </td>
            <td> <?= $item['total_price']; ?> </td>
            <td> <?= $item['created_at']; ?> </td>
            <td>
                <a href="orderDetails.php?t=<?= $item['tracking_id'];
            </td>
        </tr>
```

```
<?php
$orders = getAllOrders("mobile_orders");

if(mysqli_num_rows($orders) > 0){
    foreach($orders as $item){
    ?>
        <tr>
            <td> <?= $item['id']; ?> </td>
            <td> <?= $item['name']; ?> </td>
            <td> <?= $item['email']; ?> </td>
            <td> <?= $item['address']; ?> </td>
            <td> <?= $item['total_price']; ?> </td>
            <td>
                <a href="mobileOrderDetails.php?id=<?= $item['id'];
            </td>
        </tr>
```

Slika 117. i 118. Razlika u implementaciji stranica web narudžbi i mobilnih narudžbi

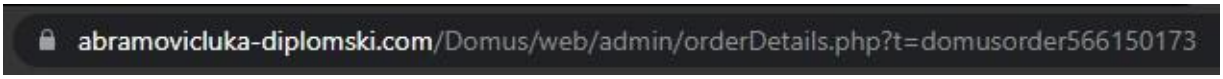


Slika 119. Prikaz web narudžbi (*orders.php*)

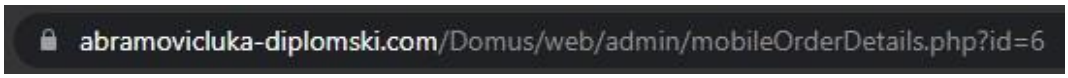


Slika 120. Prikaz mobilnih narudžbi (*mobileOrders.php*)

Klikom na gumb za provjeru detalja, otvara se ili stranica *orderDetails.php* ili *mobileOrderDetails.php,* ovisno o tome jesno li na stranici za pregled web ili mobilnih narudžbi. U slučaju web narudžbi, uzima se kod za praćenje odabrane narudžbe te se dodaje kao *slug* URL adresi stranice *orderDetails.php*, dok se u slučaju mobilne narudžbe uzima identifikacijski broj koji će također biti vidljiv kao *slug* u URL adresi stranice *mobileOrderDetails.php.*



Slika 121. Primjer URL adrese stranice za detalje web narudžbe



Slika 122. Primjer URL adrese stranice za detalje mobilne narudžbe

Implementacija stranica za prikaz detalja o narudžbi vrši se na istovjetan način kao i u slučaju prikaza detalja narudžbe u korisničkom dijelu aplikacije. Slično kao i u tom slučaju, detalji o pojedinoj narudžbi se dohvaćaju pozivanjem funkcije koja kao atribut koristi

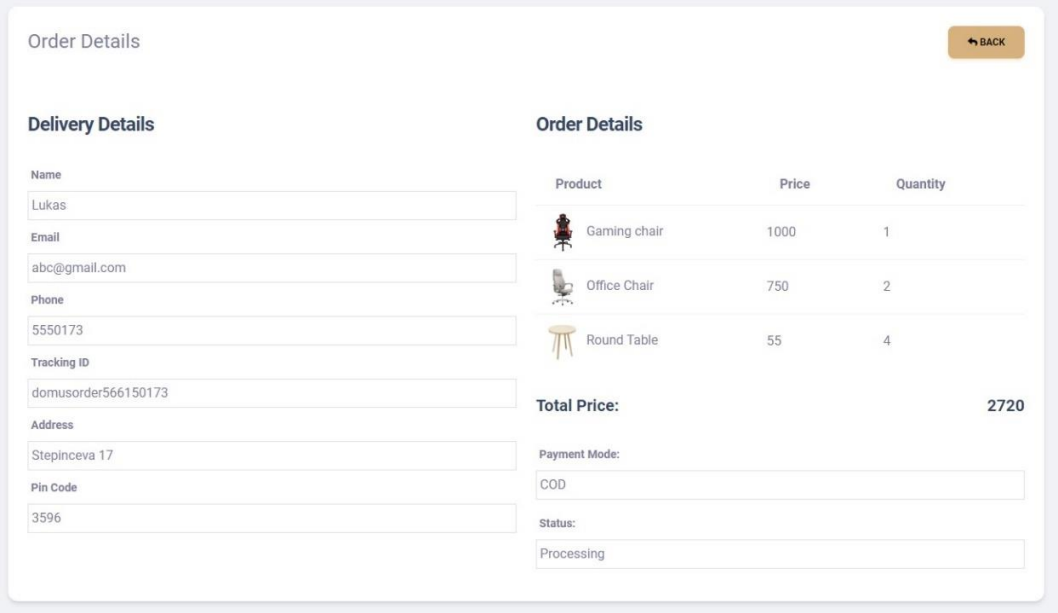
određeni *slug.* S obzirom da su kod za praćenje, odnosno identifikacijski broj jedinstveni, SELECT naredba ih koristi kao atribut da pronađe pravi redak u tablici čiji se podaci prikazuju u korisničkom sučelju.

```php
function checkTracking($trackingId){
    global $con;

    $query = "SELECT * FROM orders WHERE tracking_id='$trackingId' ";
    return mysqli_query($con, $query);
}
```

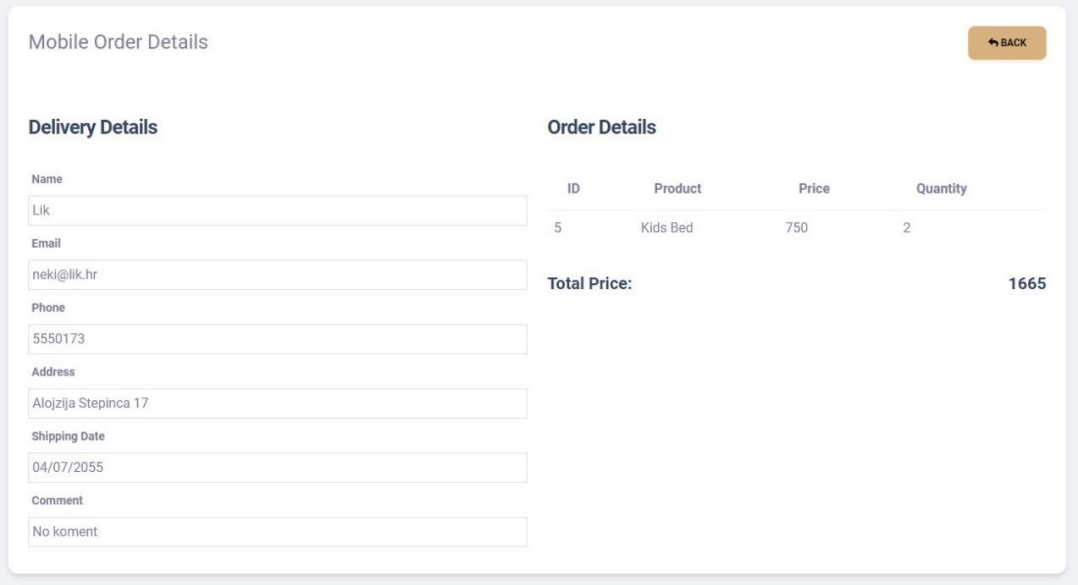
Slika 123. PHP funkcija za dohvat web narudžbi (*functions.php*)

```php
function checkMobileOrder($orderId){
    global $con;

    $query = "SELECT * FROM mobile_orders WHERE id='$orderId' ";
    return mysqli_query($con, $query);
}
```

Slika 124. PHP funkcija za dohvat mobilnih narudžbi (*functions.php*)



Slika 125. Prikaz detalja web narudžbe



Slika 126. Prikaz detalja mobilne narudžbe (mobile)

6.4. Pregled kategorija

Pored mogućnosti pregledavanja narudžbi koje je korisnik stvorio, jedna od zadaća administratora je i briga o ponudi robe korisniku.

```
function getAll($table){
    global $con;
    $query = "SELECT * FROM $table";
    return $query_run = mysqli_query($con, $query);
}
```

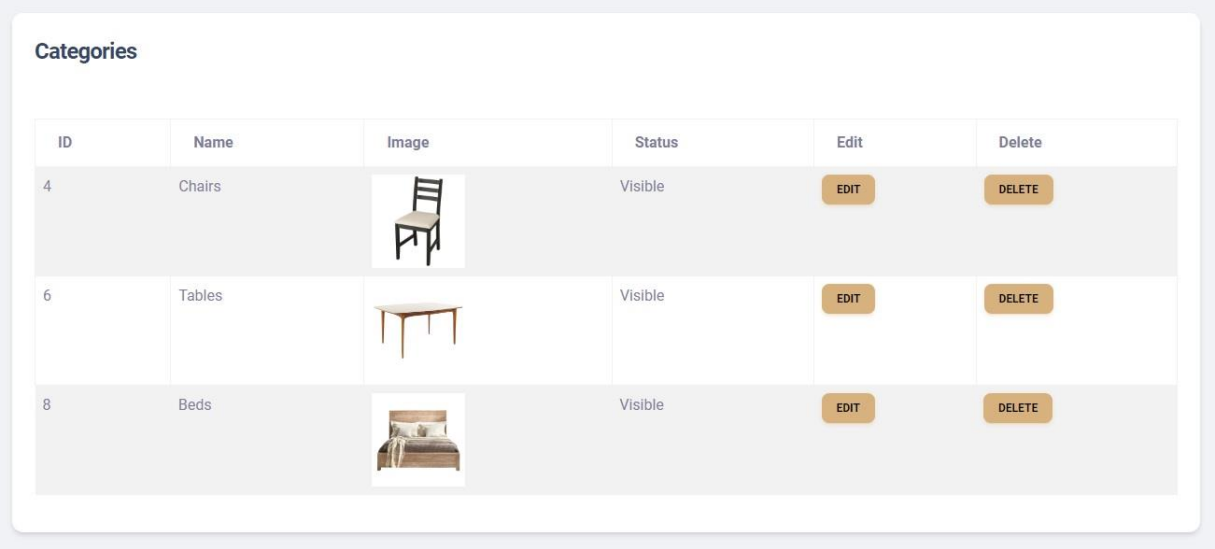
Slika 127. Funkcija *getAll* (*functions.php*)

Već je poznato da su proizvodi u ponudi pohranjeni u sebi odgovarajućim kategorijama, a administrator ima mogućnost izvoditi CRUD (*create, read, update, delete*) operacije nad cjelokupnom ponudom robe. Postojeće kategorije prikazuju se u korisničkom sučelju stranice *categories.php* u administratorskom dijelu aplikacije, tako da se izlistavaju u Bootstrap tablici korištenjem već poznate *foreach* naredbe nakon pozivanja vanjske funkcije *getAll* u kojoj se kao atribut koristi ime tablice u bazi podataka (*categories*).

```
<?php
include('../middleware/adminMiddle.php');
include('includes/header.php');
?>


<div class="container">
    <div class="row">
        <div class="col-md-12">
            <div class="card">
                <div class="card-header">
                    <h4>Categories</h4>
                </div>
                <div class="card-body" id="category_table">
                    <table class="table table-bordered table-striped">
                        <thead>
                            <tr>
                                <th>ID</th>
                                <th>Name</th>
                                <th>Image</th>
                                <th>Status</th>
                                <th>Edit</th>
                                <th>Delete</th>
                            </tr>
                        </thead>
                        <tbody>
                            <?php
                                $category = getAll("categories");

                                if(mysqli_num_rows($category) > 0){
                                    foreach($category as $item){
                                        ?>
                                    <tr>
                                        <td> <?= $item['id']; ?> </td>
                                        <td> <?= $item['name']; ?> </td>
                                        <td>
                                            <img src="../uploads/<?= $item['image']; ?>" width="100px" height="100px" alt="<?= $item['image']; ?>">
                                        </td>
                                        <td>
                                            <?= $item['status'] == '0'?"Visible":"Hidden" ?>
                                        </td>
                                        <td>
                                            <a href="editCategory.php?id=<?= $item['id']; ?>" class="btn btn-sm btn-domus">Edit</a>
                                        </td>
                                        <td>
                                            <button type="button" class="btn btn-sm btn-domus delete_category_btn" value="<?= $item['id']; ?>">Delete</button>
                                        </td>

                                    </tr>
                                        <?php

                                    }
                                }
                                else{
                                    echo "Nothing to show";
                                }
                            ?>
```

Slika 128. Implementacija stranice za pregled kategorija (*categories.php*)

Korisničko sučelje administratorske stranice za pregled kategorija prikazuje kategorije u tablici od šest stupaca od kojih krajnja dva sadrže gumbe za izmjenu ili brisanje postojećih kategorija. Četiri stupca sadrže osnovne informacije; identifikacijski broj, ime, sliku, i status kategorije. Status kategorije može biti vidljiv ili skriven, što daje mogućnost da se kategorija može zadržati u tablici baze podataka ali da istovremeno ostane skrivena u korisničkom dijelu aplikacije sve dok se status ne promjeni. Ova mogućnost uvedena je zbog ideje da određena kategorija u nekom trenutku može ne biti u opticaju, pa je jednostavnije sakriti kategoriju od krajnjeg korisnika i zadržati ju u tablici nego obrisati kategoriju i ponovno ju dodavati jednom kada se status promjeni.



Slika 129. Prikaz svih kategorija u administratorskom dijelu aplikacije

Klikom na gumb za izmjenu kategorije otvara se stranica *editCategory.php* koja u svojoj URL adresi sadrži identifikacijski broj odabrane kategorije, te se otvara stranica koja u isto vrijeme prikazuje trenutne podatke u tablici, ali sada i daje mogućnost da se ti podaci izmjene.

```
function getById($table, $id){
    global $con;
    $query = "SELECT * FROM $table WHERE id='$id' ";
    return $query_run = mysqli_query($con, $query);
}
```

Slika 130. Funkcija *getById* (*functions.php*)

Implementacija stranice za izmjenu kategorija sastoji se korištenja globalne varijable *$_GET* koja će uzeti identifikacijski broj u URL adresi stranice te ga zajedno s imenom potrebne tablice iskoristiti kao parametar u funkciji *getById,* u kojoj naredba SELECT pronalazi potrebni redak, odnosno kategoriju. Također, podaci traženog retka se u ovom slučaju ne izlistavaju u tablici već se svaki podatak prikazuje u sebi odgovarajućem elementu obrasca za izmjenu kategorije (izuzetak je slika). Time se olakšava izmjena jer je moguće izmijeniti samo jednu ili dvije stavke, a da ostali prethodni podaci ostanu u obrascu kako bi se ponovno spremili na isto mjesto nakon provedbe izmjene. Isto tako je jednostavno izmijeniti status kategorije jer se radi o standardnom potvrdom okviru (*checkbox*).

```
<form action="code.php" method="POST" enctype="multipart/form-data">
    <div class="row">
        <div class="col-md-6">
            <input type="hidden" name="category_id" value="<?= $data['id'] ?>">
            <label for="">Name</label>
            <input type="text" name="name" value="<?= $data['name'] ?>" placeholder="Enter Category Name" class="form-control">
        </div>
        <div class="col-md-6">
            <label for="">Slug</label>
            <input type="text" name="slug" value="<?= $data['slug'] ?>" placeholder="Enter Slug" class="form-control">
        </div>
        <div class="col-md-12">
            <label for="">Description</label>
            <textarea rows="3" name="description" placeholder="Enter Description" class="form-control"><?= $data['description'] ?></textarea>
        </div>
        <div class="col-md-12">
            <label for="">Upload Image</label>
            <input type="file" name="image" class="form-control">
            <label for="">Current Image</label>
            <input type="hidden" name="old_image" value="<?= $data['image'] ?>">
            <img src="../uploads/<?= $data['image'] ?>" height="50px" width="50px" alt="">
        </div>
        <div class="col-md-6">
            <label for="">Status</label>
            <input type="checkbox" <?= $data['status'] ? "checked":"" ?> name="status">
        </div>
        <div class="col-md-6">
            <label for="">Popular</label>
            <input type="checkbox" <?= $data['popular'] ? "checked":"" ?> name="popular">
        </div>
        <div class="col-md-12">
            <button type="submit" class="btn btn-domus" name="update_category_btn">Update</button>
        </div>
    </div>
</form>
```

Slika 131. Implementacija obrasca za izmjenu kategorije (*editCategory.php*)

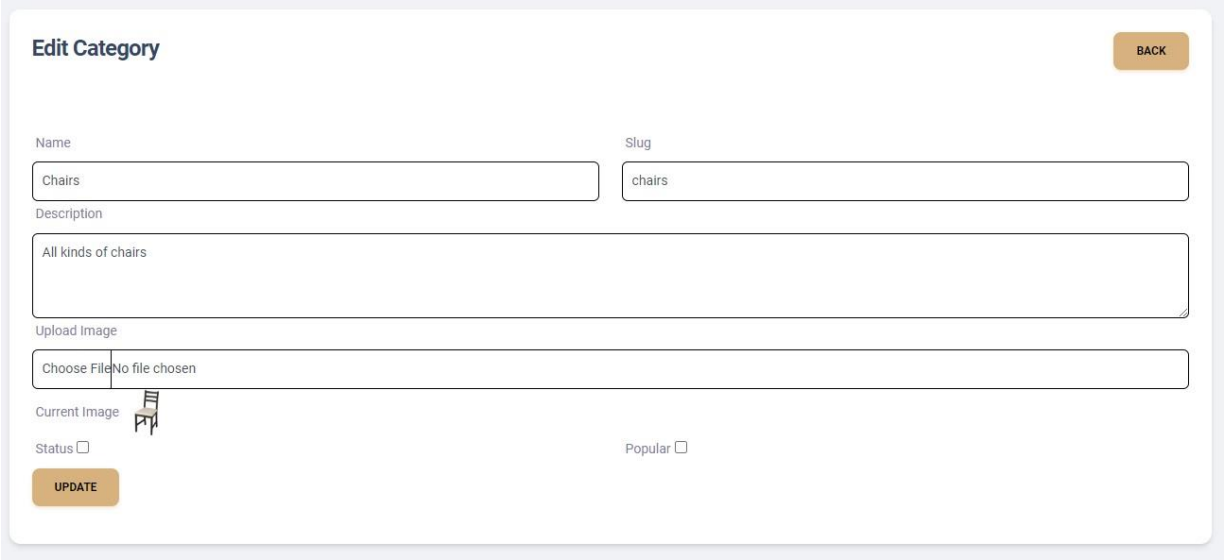
Jednom kada korisnik izvrši željene izmjene u postojećoj kategoriji, klikom na gumb izmjene (*update*) aktivira se PHP skripta *code.php* u kojoj je definirano što se događa kada se spomenuti gumb klikne.

```
else if(isset($_POST['update_category_btn'])){
    $category_id = $_POST['category_id'];
    $name = $_POST['name'];
    $slug = $_POST['slug'];
    $description = $_POST['description'];
    $status = isset($_POST['status']) ? '1':'0';
    $popular = isset($_POST['popular']) ? '1':'0';

    $new_image = $_FILES['image']['name'];
    $old_image = $_POST['old_image'];

    if($new_image != ""){
        $image_ext = pathinfo($new_image, PATHINFO_EXTENSION);
        $update_filename = time().'.'.$image_ext;
    }
    else{
        $update_filename = $old_image;
    }

    $path = "../uploads";

    $update_query = "UPDATE categories SET name='$name', slug='$slug', description='$description',
    status='$status', popular='$popular', image='$update_filename' WHERE id='$category_id' ";

    $update_query_run = mysqli_query($con, $update_query);

    if($update_query_run){
        if($_FILES['image']['name'] != ""){
            move_uploaded_file($_FILES['image']['tmp_name'], $path.'/'.$update_filename);
            if(file_exists("../uploads/".$old_image)){
                unlink("../uploads/".$old_image);
            }
        }
        redirect("editCategory.php?id=$category_id", "Category Updated");
    }
    else{
        redirect("editCategory.php?id=$category_id", "Something went wrong");
    }
}
```

Slika 132. PHP skripta za izmjenu kategorije (*code.php*)

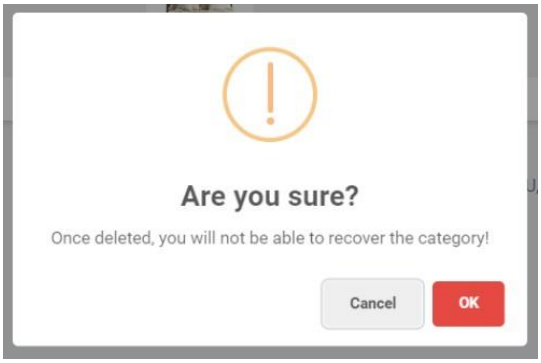
Uzimaju se svi podaci iz obrasca, novi i stari, te se spremaju u sebi odgovarajuće varijable čije vrijednosti se preko SQL naredbe UPDATE spremaju u odgovarajući redak u tablici. Time se izmijenjeni podaci u retku ažuriraju, a neizmijenjeni ostaju kakvi su i bili. Specifična situacija je izmjena slike jer se tu radi o datoteci čije je ime spremljeno u bazu podataka, ali sama datoteka se nalazi u mapi *uploads.* Shodno tome je potrebno izvesti i odgovarajuću promjenu u mapi ukoliko se fotografija izmjenjivala. Zbog toga je taj dio skripte uvjetovan *if… **else*** petljom jer ukoliko se nije mijenjala slika, već samo tekstualni podaci, onda nema potrebe za aktiviranjem tog dijela skripte. Po

izvršenoj promjeni, administrator će dobiti poruku u AlertifyJS dijaloškom prozoru da je izmjena uspješno izvršena.



Slika 133. Obrazac za izmjenu kategorije

Osim izmjene, administrator ima mogućnost i potpuno obrisati kategoriju iz tablice u bazi podataka. Pretpostavlja se da do takvog slučaja dolazi rijetko, ali je svejedno moguć. Klikom na gumb za brisanje (*delete*) u administratorskoj stranici za pregled kategorija aktivira se SweetAlert (stranica 80.) dijaloški prozor koji traži od administratora potvrdu da želi obrisati kategoriju.



Slika 134. Primjer SweetAlert dijaloškog prozora

S obzirom da se SweetAlert bazira na korištenju programskog jezika JavaScript, po pritisku gumba za brisanje se ne poziva odmah PHP funkcija kao što je to slučaj s izmjenom podataka. U ovom slučaju, aktivira se JavaScript funkcija u koja zapravo stvara spomenuti dijaloški prozor, i tek po potvrdi brisanja od strane administratora, JavaScript funkcija će korištenjem jQuery Ajax metode pozvati PHP funkciju u kojoj je deklarirana varijabla koja će poprimiti vrijednost identifikacijskog broja tražene funkcije. Primjenom SQL naredbe DELETE obrisat će se cijeli redak koji nosi taj identifikacijski broj. Jednom kada je brisanje završeno, PHP funkcija će dati JavaScript funkciji do znanja da je operacija izvršena i JavaScript će tako administratoru prikazati poruku da je brisanje retka iz tablice, odnosno kategorije prošlo uspješno. Uz to, JavaScript će iskoristiti metodu *load* kojom će dinamički ponovno učitati tablicu u kojoj se nalaze podaci o postojećim kategorijama kako bi promjena u tablici odmah bila vidljiva administratoru.

```javascript
$(document).on('click','.delete_category_btn', function (e){

    e.preventDefault();

    var id = $(this).val();
    //alert(id);

    swal({
        title: "Are you sure?",
        text: "Once deleted, you will not be able to recover the category!",
        icon: "warning",
        buttons: true,
        dangerMode: true,
      })
      .then((willDelete) => {
        if (willDelete) {

            $.ajax({
                method: "POST",
                url: "code.php",
                data: {
                    'category_id':id,
                    'delete_category_btn': true
                },
                success: function (response){
                    if(response == 200){
                        swal("Success!", "Category deleted!", "success");
                        $("#category_table").load(location.href + " #category_table");
                    }
                    else if(response == 500){
                        swal("Error!", "Something went wrong!", "error");
                    }
                }
            });

        }
      });
});
```

Slika 135. JavaScript funkcija za brisanje određene kategorije (*custom.js*)

```php
else if(isset($_POST['delete_category_btn'])){
    $category_id = mysqli_real_escape_string($con, $_POST['category_id']);

    $category_query = "SELECT * FROM categories WHERE id='$category_id' ";
    $category_query_run = mysqli_query($con, $category_query);
    $category_data = mysqli_fetch_array($category_query_run);
    $image = $category_data['image'];

    $delete_query = "DELETE FROM categories WHERE id='$category_id' ";
    $delete_query_run = mysqli_query($con, $delete_query);

    if($delete_query_run){
        if(file_exists("../uploads/".$image)){
            unlink("../uploads/".$image);
        }
        echo 200;
    }
    else{
        echo 500;
    }
}
```

Slika 136. PHP funkcija za brisanje određene kategorije (*code.php*)

6.5. Dodavanje nove kategorije

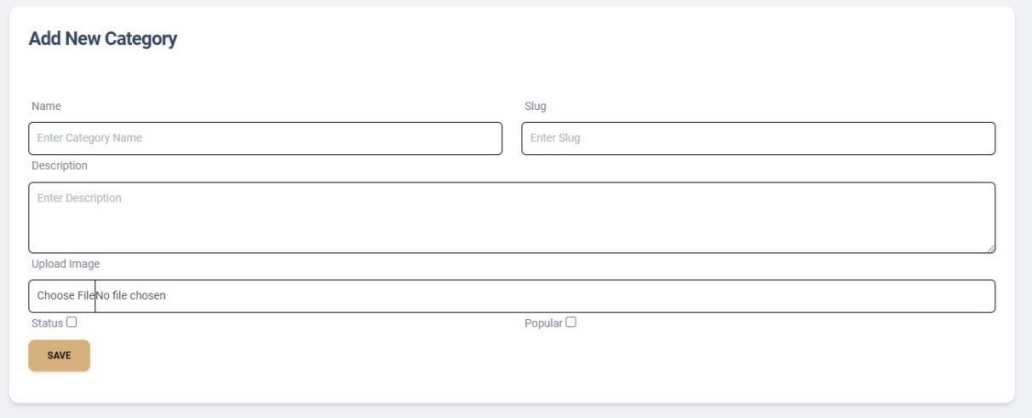
Dok administrator može provoditi pregled, izmjenu, i brisanje kategorija iz stranice prikaza kategorija (*categories.php*) u administratorskom dijelu web aplikacije, mogućnost dodavanja nove kategorije je zasebna ne samo kao stranica, nego i kao opcija u bočnom izborniku.

```php
<form action="code.php" method="POST" enctype="multipart/form-data">
    <div class="row">
        <div class="col-md-6">
            <label for="">Name</label>
            <input type="text" name="name" placeholder="Enter Category Name" class="form-control">
        </div>
        <div class="col-md-6">
            <label for="">Slug</label>
            <input type="text" name="slug" placeholder="Enter Slug" class="form-control">
        </div>
        <div class="col-md-12">
            <label for="">Description</label>
            <textarea rows="3" name="description" placeholder="Enter Description" class="form-control"></textarea>
        </div>
        <div class="col-md-12">
            <label for="">Upload Image</label>
            <input type="file" name="image" class="form-control">
        </div>
        <div class="col-md-6">
            <label for="">Status</label>
            <input type="checkbox" name="status">
        </div>
        <div class="col-md-6">
            <label for="">Popular</label>
            <input type="checkbox" name="popular">
        </div>
        <div class="col-md-12">
            <button type="submit" class="btn btn-domus" name="add_category_btn">Save</button>
        </div>
    </div>
</form>
```

Slika 137. Implementacija obrasca za dodavanje nove kategorije (*addCategory.php*)

Implementacija stranica za dodavanje nove kategorije uglavnom se sastoji samo od kontejnera u kojem se nalazi obrazac u koji se unose potrebni podaci za dodavanje nove kategorije. Implementacija ove stranice je stoga dosta jednostavna, ključno je samo zadati obrascu da aktivira PHP funkciju za dodavanje podataka u bazu podataka jednom kada se klikne na gumb spremanje kategorije.

Jednom kada se klikne na taj gumb i funkcija se aktivira, u biti se provodi klasična operacija koju programski jezik PHP provodi kada se radi o dodavanju novih poda-taka u tablicu baze podataka. Deklariraju se potrebne varijable čije vrijednosti postaju upisani podaci u obrascu. Za dodavanje slike potrebne je i deklarirati varijablu koja u sebi sadrži put, odnosno mjesto pohrane datoteke slike u upravitelju datoteka. Koristi se klasična SQL naredba INSERT INTO koja upisane podatke sprema u tablicu te odabranu sliku sprema u ciljanu mapu za pohranu slika. Po uspješnom dodavanju nove kategorije, administrator će dobiti poruku u AlertifyJS dijaloškom prozoru da je spremanje nove kategorije prošlo uspješno.



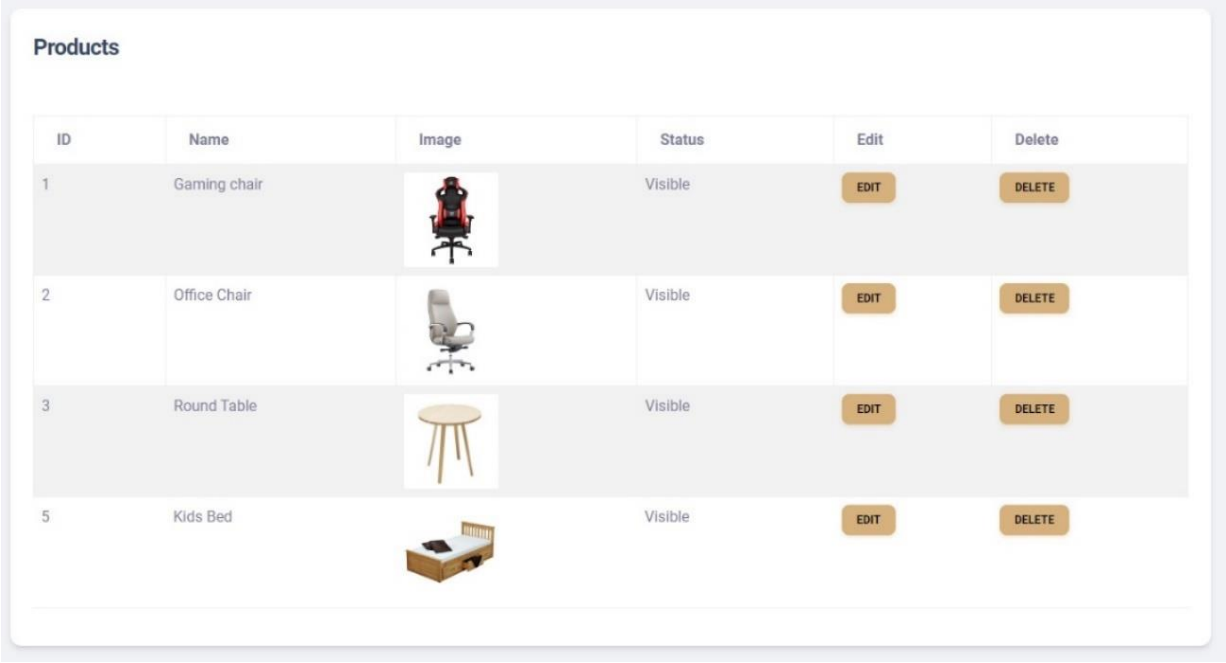
Slika 138. Prikaz obrasca za dodavanje nove kategorije (*addCategory.php*)

```php
if(isset($_POST['add_category_btn'])){
    $name = $_POST['name'];
    $slug = $_POST['slug'];
    $description = $_POST['description'];
    $status = isset($_POST['status']) ? '1':'0';
    $popular = isset($_POST['popular']) ? '1':'0';

    $image = $_FILES['image']['name'];

    $path = "../uploads";

    $image_ext = pathinfo($image, PATHINFO_EXTENSION);
    $filename = time().'.'.$image_ext;

    $categories_query = "INSERT INTO categories
    (name,slug,description,status,popular,image)
    VALUES ('$name','$slug','$description','$status','$popular','$filename')";

    $categories_query_run = mysqli_query($con, $categories_query);

    if($categories_query_run){
        move_uploaded_file($_FILES['image']['tmp_name'], $path.'/'.$filename);

        redirect("addCategory.php", "Category Added!");
    }
    else{
        redirect("addCategory.php", "Something Went Wrong!");
    }

}
```

Slika 139. PHP skripta za dodavanje nove kategorije (*code.php*)

6.6. Pregled proizvoda

Stranica za pregled proizvoda (*products.php*) u administratorskom dijelu web aplikacije suštinski je istovjetna stranici za pregled kategorija, uz tek minimalne razlike koje zapravo samo predstavljaju razliku između rada s kategorijama i rada s proizvodima, dok su implementacije funkcijski u oba slučaja iste. Implementacija stranice za prikaz proizvoda također se sastoji od pozivanja *getAll* funkcije, s tim da se sad kao atribut funkcije koristi ime tablice proizvoda (*products*) iz koje se SELECT naredbom uzimaju svi redci iz spomenute tablice te se korištenjem *foreach* naredbe izlistavaju unutar Bootstrap tablice u korisničkom sučelju.



Slika 140. Prikaz svih proizvoda u administratorskom dijelu aplikacije

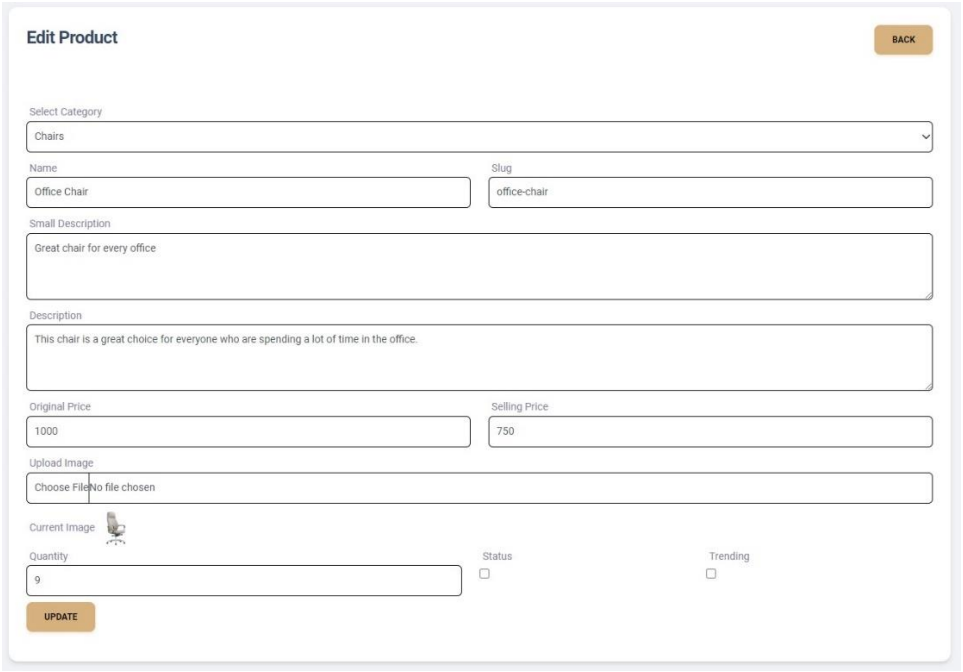
Kao i u slučaju kategorija, sadržaj tablice proizvodi (*products*) je prikazan u korisničkom sučelju unutar Bootstrap tablice od šest stupaca od kojih ponovno četiri stupca prikazuju identifikacijski broj, ime, sliku, i status proizvoda, dok dva krajnja stupca sadrže gumbe za izmjenu podataka određenog proizvoda i brisanje određenog proizvoda iz baze podataka. Klikom na gumb za izmjenu podataka o proizvodu otvara se stranica *editProduct.php* čija URL adresa, kao i u slučaju kategorije, kao parametar sadrži identifikacijski broj odabranog proizvoda. Korištenjem globalne varijable *$_GET,* stranica za izmjenu proizvoda uzima taj identifikacijski broj i koristi ga kao atribut zajedno s imenom tablice (*products*) prilikom pozivanja vanjske PHP funkcije *getById* u kojoj naredba SELECT pronalazi odgovarajući redak čiji podaci se prikazuju u obrascu gdje mogu biti podvrgnuti određenim izmjenama.

```
<form action="code.php" method="POST" enctype="multipart/form-data">
    <div class="row">
        <div class="col-md-12">
            <label class="mb-0" for="">Select Category</label>
            <select name="category_id" class="form-select mb-2">
                <option selected>Select Category</option>
                <?php
                    $categories = getAll("categories");

                    if(mysqli_num_rows($categories) > 0){
                        foreach ($categories as $item){
                            ?>
                                <option value="<?= $item['id']; ?>" <?= $data['category_id'] == $item['id']?'selected':'' ?> ><?= $item['name']; ?></option>
                            <?php
                        }
                    }
                    else{
                        echo "No category available";
                    }

                ?>
            </select>
        </div>
        <input type="hidden" name="product_id" value="<?= $data['id']; ?>">
        <div class="col-md-6">
            <label class="mb-0" for="">Name</label>
            <input type="text" required name="name" value="<?= $data['name']; ?>" placeholder="Enter Category Name" class="form-control mb-2">
        </div>
        <div class="col-md-6">
            <label class="mb-0" for="">Slug</label>
            <input type="text" required name="slug" value="<?= $data['slug']; ?>" placeholder="Enter Slug" class="form-control mb-2">
        </div>
        <div class="col-md-12">
            <label class="mb-0" for="">Small Description</label>
            <textarea rows="3" required name="small_description" placeholder="Enter Small Description" class="form-control mb-2"><?= $data['small_description']; ?></textarea>
        </div>
        <div class="col-md-12">
            <label class="mb-0" for="">Description</label>
            <textarea rows="3" required name="description" placeholder="Enter Description" class="form-control mb-2"><?= $data['description']; ?></textarea>
        </div>
        <div class="col-md-6">
            <label class="mb-0" for="">Original Price</label>
            <input type="text" required name="original_price" value="<?= $data['original_price']; ?>" placeholder="Enter Original Pric Name" class="form-control mb-2">
        </div>
        <div class="col-md-6">
            <label class="mb-0" for="">Selling Price</label>
            <input type="text" required name="selling_price" value="<?= $data['selling_price']; ?>" placeholder="Enter Selling Price" class="form-control mb-2">
        </div>
        <div class="col-md-12">
            <label class="mb-0" for="">Upload Image</label>
            <input type="hidden" name="old_image" value="<?= $data['image']; ?>">
            <input type="file" name="image" class="form-control mb-2">
            <label class="mb-0" for="">Current Image</label>
            <img src="../uploads/<?= $data['image']; ?>" alt="Product Image" height="50px" width="50px">
        </div>

        <div class="row">
            <div class="col-md-6">
                <label class="mb-0" for="">Quantity</label>
                <input type="number" required name="quantity" value="<?= $data['quantity']; ?>" placeholder="Enter Quantity" class="form-control mb-2">
            </div>
            <div class="col-md-3">
                <label class="mb-0" for="">Status</label> <br>
                <input type="checkbox" name="status" <?= $data['status'] == '0'?'':'checked' ?>>
            </div>
            <div class="col-md-3">
                <label class="mb-0" for="">Trending</label> <br>
                <input type="checkbox" name="trending" <?= $data['trending'] == '0'?'':'checked' ?>>
            </div>
        </div>
        <div class="col-md-12">
            <button type="submit" class="btn btn-domus" name="update_product_btn">Update</button>
        </div>
    </div>
</form>
```

Slika 141. Implementacija obrasca za izmjenu proizvoda (*editProduct.php*)

```php
else if(isset($_POST['update_product_btn'])){
    $product_id = $_POST['product_id'];
    $category_id = $_POST['category_id'];

    $name = $_POST['name'];
    $slug = $_POST['slug'];
    $small_description = $_POST['small_description'];
    $description = $_POST['description'];
    $original_price = $_POST['original_price'];
    $selling_price = $_POST['selling_price'];
    $quantity = $_POST['quantity'];
    $status = isset($_POST['status']) ? '1':'0';
    $trending = isset($_POST['trending']) ? '1':'0';

    $path = "../uploads";


    $new_image = $_FILES['image']['name'];
    $old_image = $_POST['old_image'];

    if($new_image != ""){
        $image_ext = pathinfo($new_image, PATHINFO_EXTENSION);
        $update_filename = time().'.'.$image_ext;
    }
    else{
        $update_filename = $old_image;
    }

    $update_product_query = "UPDATE products SET name='$name', slug='$slug', small_description='$small_description',
    description='$description', original_price='$original_price', selling_price='$selling_price', quantity='$quantity',
    status='$status', trending='$trending', image='$update_filename' WHERE id='$product_id'";

    $update_product_query_run = mysqli_query($con, $update_product_query);

    if($update_product_query_run){
        if($_FILES['image']['name'] != ""){
            move_uploaded_file($_FILES['image']['tmp_name'], $path.'/'.$update_filename);
            if(file_exists("../uploads/".$old_image)){
                unlink("../uploads/".$old_image);
            }
        }
        redirect("editProduct.php?id=$product_id", "Product Updated");
    }
    else{
        redirect("editProduct.php?id=$product_id", "Something went wrong");
    }
}
```

Slika 142. PHP skripta za izmjenu podataka o proizvodu (*code.php*)



Slika 143. Obrazac za izmjenu podataka o proizvodu

Kao i u slučaju kategorija, i proizvodi uz mogućnost izmjene mogu biti i potpuno obrisani iz sustava, i ponovno se taj proces provodi istovjetno procesu brisanja kategorija. Po odabiru gumba za brisanje određene kategorije, aktivira se JavaScript funkcija koja stvara SweetAlert dijaloški prozor koji traži potvrdu od strane administratora za brisanje proizvoda iz baze podataka. Jednom kada administrator potvrdi svoj naum, JavaScript funkcija će korištenjem jQuery Ajax metode pozvati PHP funkciju koja će provesti proces brisanja deklariranjem jedne varijable koja će poprimiti vrijednost identifikacijskog broja ciljanog proizvoda, potom će tu varijablu iskoristiti kao uvjet SELECT naredbi za pronalazak odgovarajućeg retka, i u konačnici će se primijeniti naredba DELETE koja će obrisati redak iz tablice.

```
$(document).on('click','.delete_product_btn', function (e){
    e.preventDefault();

    var id = $(this).val();
    //alert(id);

    swal({
        title: "Are you sure?",
        text: "Once deleted, you will not be able to recover the product!",
        icon: "warning",
        buttons: true,
        dangerMode: true,
      })
      .then((willDelete) => {
        if (willDelete) {

            $.ajax({
                method: "POST",
                url: "code.php",
                data: {
                    'product_id':id,
                    'delete_product_btn': true
                },
                success: function (response){
                    if(response == 200){
                        swal("Success!", "Product deleted!", "success");
                        $("#product_table").load(location.href + " #product_table");
                    }
                    else if(response == 500){
                        swal("Error!", "Something went wrong!", "error");
                    }
                }
            });

        }
      });
});
```

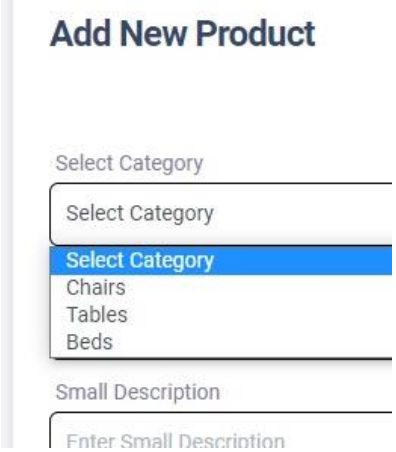
Slika 144. JavaScript funkcija za brisanje određenog proizvoda (*custom.js*)

```
else if(isset($_POST['delete_product_btn'])){
    $product_id = mysqli_real_escape_string($con, $_POST['product_id']);

    $product_query = "SELECT * FROM products WHERE id='$product_id' ";
    $product_query_run = mysqli_query($con, $product_query);
    $product_data = mysqli_fetch_array($product_query_run);
    $image = $product_data['image'];

    $delete_query = "DELETE FROM products WHERE id='$product_id' ";
    $delete_query_run = mysqli_query($con, $delete_query);

    if($delete_query_run){
        if(file_exists("../uploads/".$image)){
            unlink("../uploads/".$image);
        }
        echo 200;
    }
    else{
        echo 500;
    }
}
```

Slika 145. PHP funkcija za brisanje određenog proizvoda (*code.php*)

6.7. Dodavanje novih proizvoda

Još jedna zajednička stvar u administratorskom radu s kategorijama i proizvodima uz pregled, izmjenu, i brisanje, je dakako dodavanje novog proizvoda, odnosno novog reda u tablici proizvodi (*products*) čiji se podaci prikazuju standardnom korisniku. Iako se na prvu možda čini da je taj proces istovjetan dodavanju nove kategorije, postoji jedna velika razlika. Iako se radi o stranici za unos novih podataka, implementacija stranice *addProduct.php* na početku implementacije obrasca se poziva ranije spomenuta funkcija *getAll* koja koristi ime tablice kategorija kao atribut kako bi se dohvatila imena svih postojećih kategorija. To se čini iz razloga jer administrator prilikom unosa novog proizvoda mora odrediti u koju kategoriju je taj proizvod smješten te shodno tome se prilikom spremanja reda uz jedinstveni identifikacijski broj proizvoda u zasebni stupac sprema i identifikacijski broj odabrane kategorije.

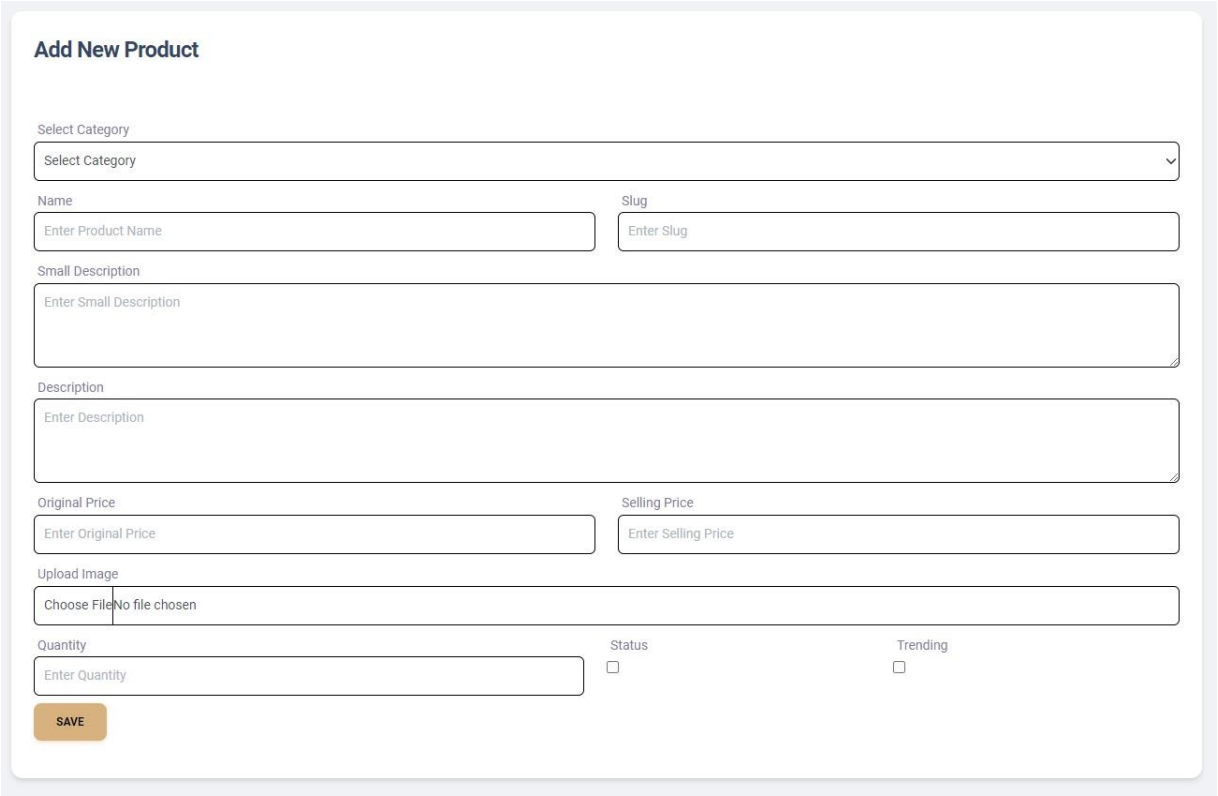


Slika 146. Odabir kategorije prilikom dodavanja novog proizvoda

```
<label class="mb-0" for="">Select Category</label>
<select name="category_id" class="form-select mb-2">
    <option selected>Select Category</option>
    <?php
        $categories = getAll("categories");

        if(mysqli_num_rows($categories) > 0){
            foreach ($categories as $item){
                ?>
                    <option value="<?= $item['id']; ?>"><?= $item['name']; ?></option>
                <?php
            }
        }
        else{
            echo "No category available";
        }

    ?>
</select>
```

Slika 147. Implementacija izbornika za odabir kategorije prilikom dodavanja novog proizvoda (*addProduct.php*)

Ostatak implementacije provodi se na gotovo jednak način kao i u slučaju dodavanja nove kategorije, s tim da dodavanje novog proizvoda zahtjeva više informacija. Također je važno za napomenuti da u ovom obrascu administrator upisuje *slug* koji je vitalan za prikazivanje detalja proizvoda u korisničkom dijelu aplikacije. Preporučljivo je da se za unos *slug-a* koriste samo mala slova i ukoliko se radi o više riječi, da se nikako ne koriste razmaci.



Slika 148. Obrazac za dodavanje novog proizvoda

```php
else if(isset($_POST['add_product_btn'])){
    $category_id = $_POST['category_id'];

    $name = $_POST['name'];
    $slug = $_POST['slug'];
    $small_description = $_POST['small_description'];
    $description = $_POST['description'];
    $original_price = $_POST['original_price'];
    $selling_price = $_POST['selling_price'];
    $quantity = $_POST['quantity'];
    $status = isset($_POST['status']) ? '1':'0';
    $trending = isset($_POST['trending']) ? '1':'0';

    $image = $_FILES['image']['name'];

    $path = "../uploads";

    $image_ext = pathinfo($image, PATHINFO_EXTENSION);
    $filename = time().'.'.$image_ext;

    if($name != "" && $slug != "" && $description != ""){
        $product_query = "INSERT INTO products(category_id, name, slug, small_description, description, original_price, selling_price,
        quantity, status, trending, image) VALUES
        ('$category_id','$name','$slug','$small_description','$description','$original_price','$selling_price','$quantity','$status','$trending','$filename')";

        $product_query_run = mysqli_query($con, $product_query);

        if($product_query_run){
            move_uploaded_file($_FILES['image']['tmp_name'], $path.'/'.$filename);

            redirect("addProduct.php", "Product Added!");
        }
        else{
            redirect("addProduct.php", "Something went wrong");
        }
    }
    else{
        redirect("addProduct.php", "Something is missing");
    }
}
```

Slika 149. PHP skripta za dodavanje novog proizvoda (*code.php*)

6.8. Rad s naslovnim fotografijama mobilne aplikacije

Dok administrator preko administratorske ploče suštinski može izvoditi sve CRUD operacije nad kategorijama i proizvodima, rad s naslovnim fotografijama koje se prikazuju u karuselu u glavnoj aktivnosti mobilne aplikacije je pojednostavljen isključivo na dodavanje novih i brisanje već postojećih naslovnih fotografija.

```php
else if(isset($_POST['add_banner_btn'])){
    $image = $_FILES['image']['name'];

    $path = "../banners";

    $image_ext = pathinfo($image, PATHINFO_EXTENSION);
    $filename = time().'.'.$image_ext;

    $banner_query = "INSERT INTO banners (image) VALUES ('$filename')";

    $banner_query_run = mysqli_query($con, $banner_query);

    if($banner_query_run){
        move_uploaded_file($_FILES['image']['tmp_name'], $path.'/'.$filename);

        redirect("banners.php", "Banner Added!");
    }
    else{
        redirect("banners.php", "Something Went Wrong!");
    }

}
```

Slika 150. PHP skripta za dodavanje nove naslovne fotografije (*code.php*)

Zbog tog jednostavnijeg pristupa prilikom rada s naslovnim fotografijama, mogućnost dodavanja i brisanja naslovnih fotografija izvodi se preko samo jedne stranice (*banners.php*). Budući da tablica naslovnih fotografija sadrži samo tri stupca (identifikacijski broj, ime fotografije, i vrijeme kada je redak stvoren), od kojih se dva popunjavaju automatski, implementacija stranice za rad s naslovnim fotografijama sastoji se od samo jedne ulazne jedinice (*input*) preko koje se umeće nova naslovna fotografija. Klikom na gumb za spremanje aktivira se PHP funkcija za dodavanje nove naslovne fotografije koja sadrži dvije deklarirane varijable; jedna čija vrijednost uzima ime naslovne fotografije te druga

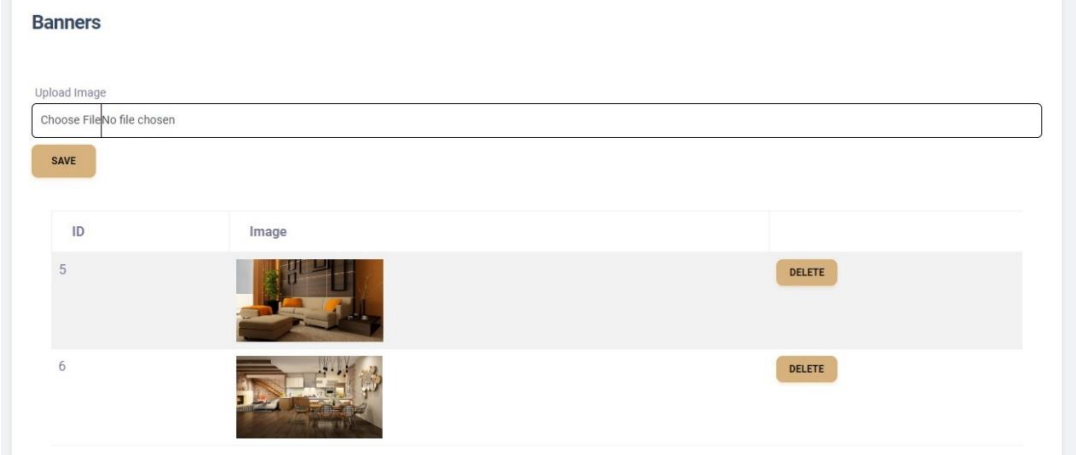
čija je vrijednost put u upravitelju datotekama gdje će se datoteka te naslovne fotografije spremiti. Standardnom SQL naredbom dodaje se samo ime fotografije u bazu podataka, a jednom kada se i sama datoteka fotografije spremi na ciljano mjesto (u mapu *banners*), administrator će dobiti informaciju da je dodavanje naslovne fotografije uspješno izvršeno. Brisanje naslovne fotografije provodi se iz iste stranice kao i dodavanje, i kao i u slučaju kategorija i proizvoda, pritiskom na gumb za brisanje ponovno se aktivira JavaScript funkcija koja prikazuje SweetAlert dijaloški prozor koji još jednom pita korisnika je li siguran da želi obrisati naslovnu fotografiju. Potvrdom administratora, aktivira se jQuery Ajax metoda kojom se poziva PHP funkcija za brisanje koja sadrži varijablu s vrijednosti identifikacijskog broja naslovne fotografije. SQL naredbom SELECT se traži ciljani redak u tablici i naredbom DELETE se taj redak, odnosno naslovna fotografija uklanja iz baze podataka. PHP potom javlja JavaScript funkciji da je proces obavljen i shodno tome JavaScript funkcija administratoru prikazuje poruku da je proces brisanja naslovne fotografije izvršen.

```
$(document).on('click','.delete_banner_btn', function (e){

    e.preventDefault();

    var id = $(this).val();

    swal({
        title: "Are you sure?",
        text: "Once deleted, you will not be able to recover the banner!",
        icon: "warning",
        buttons: true,
        dangerMode: true,
      })
      .then((willDelete) => {
        if (willDelete) {

            $.ajax({
                method: "POST",
                url: "code.php",
                data: {
                    'banner_id':id,
                    'delete_banner_btn': true
                },
                success: function (response){
                    if(response == 200){
                        swal("Success!", "Banner deleted!", "success");
                        $("#banners_table").load(location.href + " #banners_table");
                    }
                    else if(response == 500){
                        swal("Error!", "Something went wrong!", "error");
                    }
```

Slika 151. JavaScript funkcija za brisanje naslovne fotografije (*custom.js*)

```
else if(isset($_POST['delete_banner_btn'])){
    $banner_id = mysqli_real_escape_string($con, $_POST['banner_id']);

    $banner_query = "SELECT * FROM banners WHERE id='$banner_id' ";
    $banner_query_run = mysqli_query($con, $banner_query);
    $banner_data = mysqli_fetch_array($banner_query_run);
    $image = $banner_data['image'];

    $delete_query = "DELETE FROM banners WHERE id='$banner_id' ";
    $delete_query_run = mysqli_query($con, $delete_query);

    if($delete_query_run){
        if(file_exists("../banners/".$image)){
            unlink("../banners/".$image);
        }
        echo 200;
    }
    else{
        echo 500;
    }
}

else{
    header('Location: ../index.php');
}

?>
```

Slika 152. PHP funkcija za brisanje naslovne fotografije (*code.php*)



Slika 153. Dodavanje i prikaz naslovnih fotografija

```php
<?php
include('../middleware/adminMiddle.php');
include('includes/header.php');
?>

<div class="container">
    <div class="row">
        <div class="col-md-12">
            <div class="card">
                <div class="card-header">
                    <h4>Banners</h4>
                </div>
                <div class="card-body">
                    <form action="code.php" method="POST" enctype="multipart/form-data">

                        <div class="row">
                            <div class="col-md-12">
                                <label class="mb-0" for="">Upload Image</label>
                                <input type="file" required name="image" class="form-control mb-2">
                            </div>

                            <div class="col-md-12">
                                <button type="submit" class="btn btn-domus" name="add_banner_btn">Save</button>
                            </div>
                        </div>

                    </form>

                    <div class="card-body" id="banners_table">
                    <table class="table table-bordered table-striped">
                        <thead>
                            <tr>
                                <th>ID</th>
                                <th>Image</th>
                            </tr>
                        </thead>
                        <tbody>
                            <?php
                                $banners = getAll("banners");

                                if(mysqli_num_rows($banners) > 0){
                                    foreach($banners as $item){
                                        ?>
                                    <tr>
                                        <td> <?= $item['id']; ?> </td>
                                        <td>
                                            <img src="../banners/<?= $item['image']; ?>" height="100px" alt="<?= $item['image']; ?>">
                                        </td>
                                        <td>
                                            <button type="button" class="btn btn-sm btn-domus delete_banner_btn" value="<?= $item['id']; ?>">Delete</button>
                                        </td>
                                    </tr>
                                        <?php

                                    }
                                }
                                else{
                                    echo "Nothing to show";
                                }
                            ?>
                        </tbody>
                        </table>
                </div>
                </div>
            </div>
        </div>
    </div>
</div>

<?php include('includes/footer.php') ?>
```

Slika 154. Implementacija stranice za dodavanje i brisanje naslovnih fotografija (*banners.php*)

7. ZAKLJUČAK

Ovaj cjelokupni projekt je razvijen kao sustav dviju aplikacija, mobilne i web aplikacije, koje međusobno komuniciraju i tako ostvaruju indirektnu komunikaciju između administratora i krajnjeg korisnika. Iako se u slučaju ovog projekta radi o fiktivnom salonu namještaja *Domus,* valja naglasiti da se ovakav projekt funkcionalno može primijeniti na gotovo bilo koju vrstu internetske trgovine. Obavljanje kupovine preko interneta postalo je dio svakodnevice, i svima koji nude svoje proizvode je u interesu da njihova ponuda bude dostupna korisnicima bilo kada, bilo gdje, i na bilo kojem pametnom uređaju. Većina internetskih usluga koje su nekada bile dostupne samo na računalima danas su mobilne i korisnici im kao takvima mogu pristupiti korištenjem pametnog telefona. Iz tog razloga mnoge kompanije razvijaju vlastite mobilne aplikacije kako bi bili što pristupačniji korisnicima.

Mobilna aplikacija razvijena je u programu Android Studio, dok je web aplikacija razvijena i pohranjena na mrežnoj usluzi poslužitelja HostGator, preko kojeg je također razvijena i potrebna baza podataka korištenjem softverskog alata phpMyAdmin. Spomenuti alat je jedan od najpoznatijih načina za razvoj i upravljanje MySQL bazom podataka. Baza podataka koja je razvijena za ovaj projekt broji ukupno devet tablica od kojih na nekima promjene može vršiti samo korisnik, a na nekima samo administrator (administrator u svom svojstvu može samo pregledavati, ali ne i vršiti promjene u tablici narudžbi, dok za korisnika isto vrijedi u slučaju ponude kategorija i proizvoda.)

U razvoju ovog projekta korišteni su označni jezici koji su se koristili za dizajn korisničkog sučelja aplikacija te programski jezici koji su tim aplikacijama davali potrebnu funkcionalnost. Za razvoj mobilne aplikacije korišten je označni jezik XML te programski jezik Java, dok je za razvoj web aplikacije korišten HTML, CSS, te programski jezici PHP i JavaScript. PHP se prilikom razvoja koristio znatno više jer su u njemu programirane i skripte koje omogućuju komuniciranje i razmjenu podataka u JSON formatu između mobilne aplikacije i servera. U razvoju mobile aplikacije koristila se i HTTP biblioteka Volley koja je jedna od dva najpoznatija načina kojim se mobilnim aplikacijama omogućuje povezanost s bazom podataka. Postoji još jedan način, zvan Retrofit, koji je u ranim fazama razvoja projekta testiran kao moguće inicijalno rješenje, ali je kasnije zamijenjen Volleyem.

U ovom projektu je realizirano da krajnji korisnik ima mogućnost koristiti se mobilnom aplikacijom i korisničkim dijelom web aplikacije, dok administrator obavlja sebi sopstvene zadatke korištenjem administratorskog dijela web aplikacije kojem standardni korisnici ne mogu pristupiti. Korisnik ima mogućnost razgledati ponudu, kategorije proizvoda i same proizvode, i stvarati narudžbe, dok administrator ima mogućnost pregledavati narudžbe koje korisnici stvore i ktome može izvoditi CRUD operacija nam ponudom.

8. LITERATURA

1. Booch, Grady; Rumbauch, James; Jacobson, Ivar, The Unified Modeling Language User Guide, Addison Wesley Professional, 2005
2. Lucid Software Inc., "UML Use Case Diagram Tutorial", lucidchart.com/pages/uml-use-case-diagram
3. Bell, Donald, "Explore the UML sequence diagram", developer.ibm.com/articles/the-sequence-diagram
4. Lucid Software Inc., "UML Class Diagram Tutorial", lucidchart.com/pages/uml-class-diagram
5. "Activities", stuff.mit.edu/afs/sipb/project/android/docs/guide/components/activities.html
6. Android Developers, "AppCompatActivity", developer.android.com/reference/androidx/appcompat/app/AppCompatActivity
7. TinyCart, github.com/hishd/TinyCart
8. Abhishek Saini, "Adapter Tutorial With Example In Android Studio", abhiandroid.com/ui/adapter
9. Android Developers, "Context", developer.android.com/reference/android/content/Context
10. L. Wilson, Jeffrey; Zamora, Gabriel, "HostGator Web Hosting Review", pcmag.com/reviews/hostgator-web-hosting
11. abramovicluka-diplomski.com
12. PCMag, "PCMag Encyclopedia, File Manager", pcmag.com/encyclopedia/term/file-manager
13. W3Schools, "PHP Include Files", w3schools.com/php/php_includes.asp
14. TechTerms, "Localhost", techterms.com/definition/localhost
15. TechTarget Contributor, "Android Studio", techtarget.com/searchmobilecomputing/definition/Android-Studio
16. Glide, github.com/bumptech/glide
17. MaterialSearchBar, github.com/mancj/MaterialSearchBar
18. Why Not! Image Carousel, github.com/ImaginativeShohag/Why-Not-Image-Carousel
19. Volley, github.com/google/volley
20. TinyCart, github.com/hishd/TinyCart
21. Google, "Volley", google.github.io/volley
22. Android Developers, "Intent", developer.android.com/reference/android/content/Intent
23. Android Developers, "View Binding", developer.android.com/topic/libraries/view-binding
24. Florian, Walther, Coding in Flow, "View Binding – Getting Started + Differences – Android Studio Tutorial", youtube.com/c/CodinginFlow
25. Android Developers, "BaseAdapter", developer.android.com/reference/android/widget/BaseAdapter
26. Oracle, "The Map Interface", docs.oracle.com/javase/tutorial/collections/interfaces/map.html

27. Codepath, "Using an ArrayAdapter with ListView", guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView
28. AltexSoft, "What is API: Definition, Types, Specifications, Documentation", altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/
29. RedHat, "What is a REST API?", redhat.com/en/topics/api/what-is-a-rest-api
30. TutorialsEye, "JSON with PHP", tutorialseye.com/json-with-php.html
31. Bootstrap, github.com/twbs/bootstrap
32. MDN contributors, "Header", developer.mozilla.org/en-US/Web/HTML/Element/header
33. Bootstrap, "bootstrap.min.css", github.com/twbs/bootstrap/blob/main/dist/css/bootstrap.min.css
34. Bootstrap, "The Mit License", github.com/twbs/bootstrap/blob/main/LICENSE
35. Fonticons, Inc., "Font Awesome", fontawesome.com
36. Fonticons, Inc., "Font Awesome Pro License", fontawesome.com/license
37. Google, "About Google Fonts", fonts.google.com/about
38. Google, "Google Fonts", fonts.google.com
39. Mohammad Younes, "AlertifyJS", alertifyjs.com
40. OpenJS Foundation and other jQuery contributors, "Downloading jQuery", jquery.com/download
41. OpenJS Foundation and other jQuery contributors, "What is jQuery?", jquery.com
42. OpenJS Foundation and other jQuery contributors, "License", jquery.org/license
43. Bootstrap, "Carousel", getbootstrap.com/docs/4.0/components/carousel
44. Bootstrap, "Forms", getbootstrap.com/docs/4.0/components/forms
45. MDN Web Docs Glossary, "Slug", developer.mozilla.org/en-US/docs/Glossary/Slug
46. MDN Web Docs, "preventDefault", developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault
47. MDN Web Docs, "closest", developer.mozilla.org/en-US/docs/Web/API/Element/closest
48. W3Schools, "JavaScript parseInt", w3schools.com/jsref/jsref_parseint.asp
49. MDN Web Docs, "Ajax", developer.mozilla.org/en-US/docs/Web/Guide/AJAX
50. Bootstrap, "Tables", getbootstrap.com/docs/4.1/content/tables
51. Bootstrap, "Navbar", getbootstrap.com/docs/4.0/components/navbar
52. Creative Tim, "material-dashboard", github.com/creativetimofficial/material-dashboard/tree/master/assets/js
53. Creative Tim, "End User License Agreement", creative-tim.com/license
54. Tristan Edwards, "SweetAlert", sweetalert.js.org/guides

9. SLIKE

10. SAŽETAK

Mobilna aplikacija *Domus* i istoimena web aplikacija dvije su aplikacije za fiktivni salon namještaja koje ostvaruju povezanost s MySQL bazom podataka preko koje vrše komunikaciju i razmjenu potrebnih podataka. Mobilna aplikacija razvijena je u programu Android Studio, korištenjem programskog jezika Java i označnog jezika XML, dok je web aplikacija razvijena na HostGator serveru korištenjem programskog jezika PHP i označnog jezika HTML, uz dodatno korištenje stilskog jezika CSS i programskog jezika JavaScript. Baza podataka razvijena je na HostGator serveru korištenjem alata phpMyAdmin.

Mobilna aplikacija stvorena je za korištenje samo krajnjim korisnicima koji u njoj mogu pregledavati ponudu koja je pohranjena u bazi podataka te se prikazuje mobilnoj aplikaciji korištenjem programskog jezika PHP, čije skripte potrebne podatke iz baze šalju u JSON formatu prema mobilnoj aplikaciji koja koristi HTTP biblioteku Volley za iščitavanje tih podataka i prikaz istih u korisničkom sučelju mobilne aplikacije. Korisnik ima mogućnosti stvarati narudžbe koje se također preko Volley-a u JSON formatu šalju prema serveru gdje ih onda PHP dekodira i dobivene podatke pohranjuje u bazu podataka.

Web aplikacija stvorena je da bude pogodna za rad administratoru, ali i krajnjim korisnicima. Podijeljena je u dva dijela, s tim da administratorskom dijelu aplikacije mogu pristupiti samo korisnici koji imaju administratorske privilegije. Kao i u slučaju mobilne aplikacije, standardni korisnik može pregledavati ponudu, ali za razliku od mobilne aplikacije, narudžbe može stvarati samo uz prijavu. Korisnik u web aplikaciji također ima mogućnost pregledavati i svoje prethodno stvorene narudžbe. Administrator koristi administratorsku ploču za pregled svih narudžbi, te također može vršiti CRUD operacije nad ponudom.

Ključne riječi: Diplomski rad, Domus, Salon namještaja, Namještaj, Android Studio, Java, HostGator, MySQL, phpMyAdmin, PHP, JavaScript, Mobilna aplikacija, Web aplikacija

11. ABSTRACT

The Domus mobile application and the homonymous web application are two applications for a fictitious furniture salon that connect to a MySQL database through which they communicate and exchange data. The mobile application was developed in the Android Studio program, using the Java programming language and the XML markup language, while the web application was developed on the HostGator server using the PHP programming language and the HTML markup language, with the additional use of the CSS style sheet language and the JavaScript programming language. The database was developed on the HostGator server using the phpMyAdmin administration tool.

The mobile application was created to be used only by end users who can view the offer stored in the database and displayed to the mobile application using the PHP programming language, whose scripts send the necessary data from the database in JSON format to the mobile application using the Volley HTTP library for reading this data and displaying it in the mobile application's user interface. The user has the ability to create orders that are also sent via Volley in JSON format to the server, where PHP then decodes them and stores the resulting data in the database.

The web application was created to be convenient for the administrator, but also for end users. It is divided into two parts, with the fact that the administrative part of the application can only be accessed by users who have administrator privileges. As in the case of the mobile application, the standard user can view the offer, but unlike the mobile application, they can only create orders by logging in. In the web application, the user also has the possibility to view his previously created orders. The administrator uses the admin panel to view all orders, and can also perform CRUD operations on the offer.

Keywords: Diploma thesis, Domus, Salon furniture, Furniture, Android Studio, Java, HostGator, MySQL, phpMyAdmin, PHP, JavaScript, Mobile application, Web application