Izrada responzivne web aplikacije za e-trgovinu koristeći React i Spring Boot

Vukasović, Milan

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Milan Vukasović

Izrada responzivne web aplikacije za e-trgovinu koristeći React i Spring Boot

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Milan Vukasović

JMBAG: 0016142919

Studij: Informacijsko i programsko inženjerstvo

Izrada responzivne web aplikacije za e-trgovinu koristeći React i Spring Boot

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, rujan 2024.

Milan Vukasović

Izjava o izvornosti

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

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U teorijskom dijelu rada bit će opisano što je to e-trgovina i što je responzivan dizajn. Nakon toga biti će opisan plan izrade aplikacije, što su to REST servisi, koje tehnologije su potrebne za izradu web aplikacije jedne e-trgovine: React za klijentsku aplikaciju, Spring Boot za poslužiteljsku aplikaciju i PostgreSQL za bazu podataka. Nakon toga kreće praktični dio rada u kojemu je opisana sama izrada aplikacije. Prvo je opisana izrada poslužiteljske aplikacije, a nakon toga i klijentske aplikacije, s naglaskom na responzivan dizajn.

Ključne riječi: React, Spring Boot, sql, e-commerce, REST, responzivan dizajn

Sadržaj

1. Uvod

„E-trgovina (eng. e-commerce) predstavlja elektroničko poslovanje i odnosi se na trgovinu robom i uslugama putem elektroničkog medija.“ [1] E-trgovine su revolucionirale poslovanja, tako što su korisnicima omogućili kupnju raznih dobara i usluga putem mreže. Kao korisniku koji često koristi e-trgovine, mogu vidjeti puno benefita za korisnike, npr. naručivanje u bilo kojem mjestu i u bilo koje vrijeme, usporedbe cijena istog proizvoda na različitim e-trgovinama i sl.

E-trgovine imaju sve veću važnost, te iz tog razloga cilj ovog diplomskog rada će biti istražiti važnost e-trgovina, način na koji se one razvija web aplikacija za njih, te i sam razviti web aplikacije za e-trgovinu. Pri izradi aplikacije, velika važnost će se staviti na responzivan dizajn web aplikacije. Responzivan dizajn omogućuje programerima izradu web stranica koje se dinamički prilagođavaju veličini uređaja. Ova filozofija razvoja omogućuje brzo i optimizirano prikazivanje web stranica, osiguravajući dobro korisničko iskustvo na mobilnim uređajima, tabletima i računalima. [2]

Smatram kako je kod e-trgovina bitan responzivan dizajn, jer osigurava da e-trgovina radi na bilo kojem uređaju koji posjeduje web preglednik. Također, isti korisnik može koristiti aplikaciju s različitih uređaja, te će uvijek imati slično iskustvo. Tako će pri izboru češće koristiti tu e-trgovinu, jer će se uvijek lakše snaći, te će mu iskustvo biti prilagođeno. Iz toga razloga, kao primjer svoje e-trgovine izabrao sam izradu e-trgovine za vezanu uz prodaju knjiga.

U ovom radu ću najprije objasnit teorijske pojmove vezane za e-trgovine, opisati responzivan dizajn detaljnije, kao i arhitekturu aplikacije i korištene tehnologije. Za praktičan dio rada, tj. za izradu e-trgovine koristit ću nekoliko tehnologija potrebnih za izradu web aplikacije. Za izradu klijentskog dijela aplikacije, tj. korisničkog sučelja koristit ću React tehnologiju. Za izradu poslužiteljskog dijela aplikacije koristit ću Spring Boot. Za bazu podataka, u kojoj ću spremati podatke o knjigama i korisnicima, koristit ću PostgreSQL.

2. E-trgovina

Elektroničke trgovine ili e-trgovine predstavljaju različite vrste online biznisa koje pružaju proizvode i usluge. Odnosi se na sve vrste poslovanja gdje se transakcije odvijaju elektronički. Ljudi često danas povezuju pojam e-trgovina sa kupnjom proizvoda putem Interneta, ali ta definicija nije potpuna. Potpuna definicija prema Anjali Gupti glasi: „E-trgovina je upotreba elektroničkih komunikacija i digitalne obrade informacija u poslovnim transakcijama radi stvaranja, transformacije i predefiniranja odnosa za stvaranje vrijednosti između ili među organizacijama te između organizacija i pojedinaca“. [1]

Važno je razlikovati e-trgovinu i e-biznis (eng. e-business). Kod e-trgovina, informacijska i komunikacijska tehnologija (skr. IKT) se koristi za transakcije. E-biznis koristi IKT kako bi poboljšalo svoj postojeći biznis. Postoji nekoliko glavnih tipova e-trgovina [1]:

- Business-to-business (B2B)
- Business-to-consumer (B2C)
- Business-to-government (B2G)
- Consumer-to-consumer (C2C)
- Mobile commerce (m-commerce)

Konkretno u ovom radu radit ću e-trgovina tipa B2C, tj. trgovinu gdje će potrošači moći kupovati proizvode nekog biznisa. Proizvod može biti fizičko dobro, npr. kućanski aparati; digitalno dobro, npr. video igra ili e-usluga, npr. izrada web stranice. [3]

B2C e-trgovina omogućuje potrošačima prikupiti informacije, kupiti neki proizvod ili uslugu i platiti ga putem Interneta. Prednosti B2C e-trgovina za kupce je u tome što im omogućuje lagani i brz pregled proizvoda kao i mogućnost kupnje istih. Uz to, proizvod je moguće kupiti u bilo koje vrijeme iz bilo kojeg mjesta. Za organizaciju e-trgovina nudi jeftin pristup da se probiju tržište, jer ne trebaju otvarati mnoštvo dućana, već je potrebno imati samo web stranicu. Kupci mogu biti iz drugih država te mogu, gledajući njihove prijašnje kupnje, preporučiti druge slične proizvode za njihove želje ili potrebe, što je benefit i za kupca, i za organizaciju. [3]

3. Responzivan dizajn

Svaki put kada korisnik otvori web stranicu, prva i najvažnija stvar koju treba je to lakoća pristupa svim informacijama uz što manje napora. Kako danas imamo uređaje različitih veličina ekrana, tradicionalne web stranice morale su se prilagoditi takvim uređajima, kako bi korisničko iskustvo ostalo dobro. Iz tog razloga, pojavljuje se potreba za responzivnim dizajnom web stranica, koji ovisno o veličini ekrana, prilagođava svoj dizajn, od velikih TV ekrana s visokom rezolucijom, do malih mobilnih ekrana s niskom rezolucijom. [4]

Izraz “Responzivan web dizajn” prvi put koristi Ethan Marcotte 2010. na svojoj “A book apart” web stranici. Taj izraz se još može povezati sa nekoliko izraza koji otprilike imaju isto značenje: fluidni dizajn (eng. Fluid design), elastičan raspored (eng. Elastic layout), prilagodljiv raspored (eng. Adaptive layout), dizajn za različite uređaje (eng. Cross device design), fleksibilan dizajn (eng. Flexible design). [4] Responzivan web dizajn omogućuje korisnicima najbolje praksa dizajna za sve uređaje, kao što su pametni mobiteli, tableti, laptopi i stolna računala. Na idućoj slici prikazano je kako se isti elementi unutar web aplikacije mogu ponašati drugačije, ovisno o veličini ekrana:

Slika 1: primjer prikaza web stranica na različitim uređajima [5]

Današnji ekrani, uz nekoliko izuzetaka, mogu se podijeliti u tri glavne kategorije prema svojoj veličini: mali, srednji i veliki. Ove kategorije odgovaraju ekranima na različitim uređajima, poput mobitela, tableta i računala. Pri izradi responzivnih web stranica ključnu ulogu igraju prijelomne točke (eng. breakpoints), koje određuju kako će se dizajn prilagoditi različitim širinama ekrana. Prijelomne točke odnose se na broj piksela u širini ekrana. Na temelju broja piksela u širini ekrana, aplikacija se može optimizirati za prikaz na svakom od ovih uređaja, osiguravajući ugodno korisničko iskustvo. Prijelomne točke nisu striktno definirane za izradu web aplikacija, te je moguće definirati proizvoljan broj piksela, ovisno o vlastitim potrebama za

aplikaciju. Međutim, postoje dobre prakse oko koliko piksela bi se trebale postavljati prijelomne točke, pa prema Microsoftu [6], to izgleda kao u tablici 1.

Tablica 1: opis prijelomnih točaka

Veličina	Prijelomne točke	Uređaj	Tipične veličine prozora
Mala	Do 640 px	Mobiteli	320x596 360x640 480x854
Srednja	641 – 1007 px	Tableti	960x540
Velika	1008p i više	Računala	1366x768 1920x1080

Kako bi se postigao responzivan web dizajn, koristi se jezici HTML, za izradu elemenata web stranice; i CSS za stiliziranje web stranice. U HTML je potrebno postaviti tag: "*<meta name="viewport" content="width=device-width,initial-scale=1" />*". Ovaj tag govori web preglednicima da postavi širinu prikaznog prostora (eng. Viewport) na širinu uređaja i skalira dokument na 100% njegove namjeravane veličine. [7]

Postoje tri ključne značajke kod razvoja responzivne web stranice [7]:

- Medijski upiti
- Tehnologije responzivnog rasporeda
- Responzivne slike/mediji

3.1. Medijski upiti

Medijski upiti omogućuju provođenje testova i primjenu CSS-a selektivno kako bi se stranica stilizirala za potrebe korisnikovog uređaja. U jedan stilski list (eng. Stylesheet) moguće je dodati više upita, npr. ovisno o rezolucijama ekrana, i tako promijeniti stil za svaki od tih ekrana. Točke u kojima se uvodi medijski upit i mijenja raspored nazivaju se već spomenute prijelomne točke. Jedan od pristupa korištenja medijskih upita je kreiranje jednostavnog rasporeda s jednim stupcem za uređaje s uskim zaslonom (npr. za mobitele), a ako je zaslon širi, onda je moguće prikazati više stupaca odjednom. Tako će npr. sljedeći upit prikazivati elemente samo ako je veličina ekrana veća od 1025px: „*@media (min-width:1025px)*.

3.2. Tehnologije responzivnog rasporeda

U responzivnom dizajnu, postoji nekoliko CSS elemenata, koji su sami po sebi responzivni. Ti elementi su: više stupčani raspored (eng. Multiple-column layout), *Flexbox* i rešetka (eng. Grid).

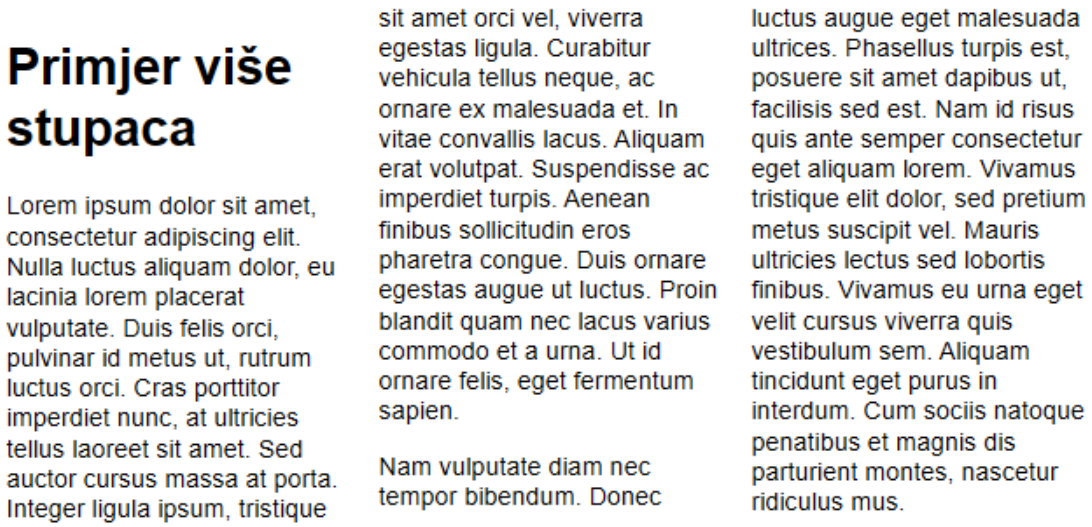
Kod višestupčanog rasporeda postoje dva načina definiranja stupaca. Prvi način je specificirati broj stupaca na koji će se sadržaj podijeliti. Preglednik tada izračunava veličinu stupaca i mijenja ih prema veličini zaslona [7]:

```
.spremnik {
  column-count: 3;
}
```

Također je moguće napraviti obratno, tj. specificirati širinu stupca, pa će preglednik kreirati broj stupaca ovisno o sveukupnoj širini, te ih tako rasporediti:

```
.spremnik {
  column-width: 200px;
}
```

Na sljedećoj slici moguće je vidjeti kako izgleda kada se 2 paragrafa postave na 3 stupca.



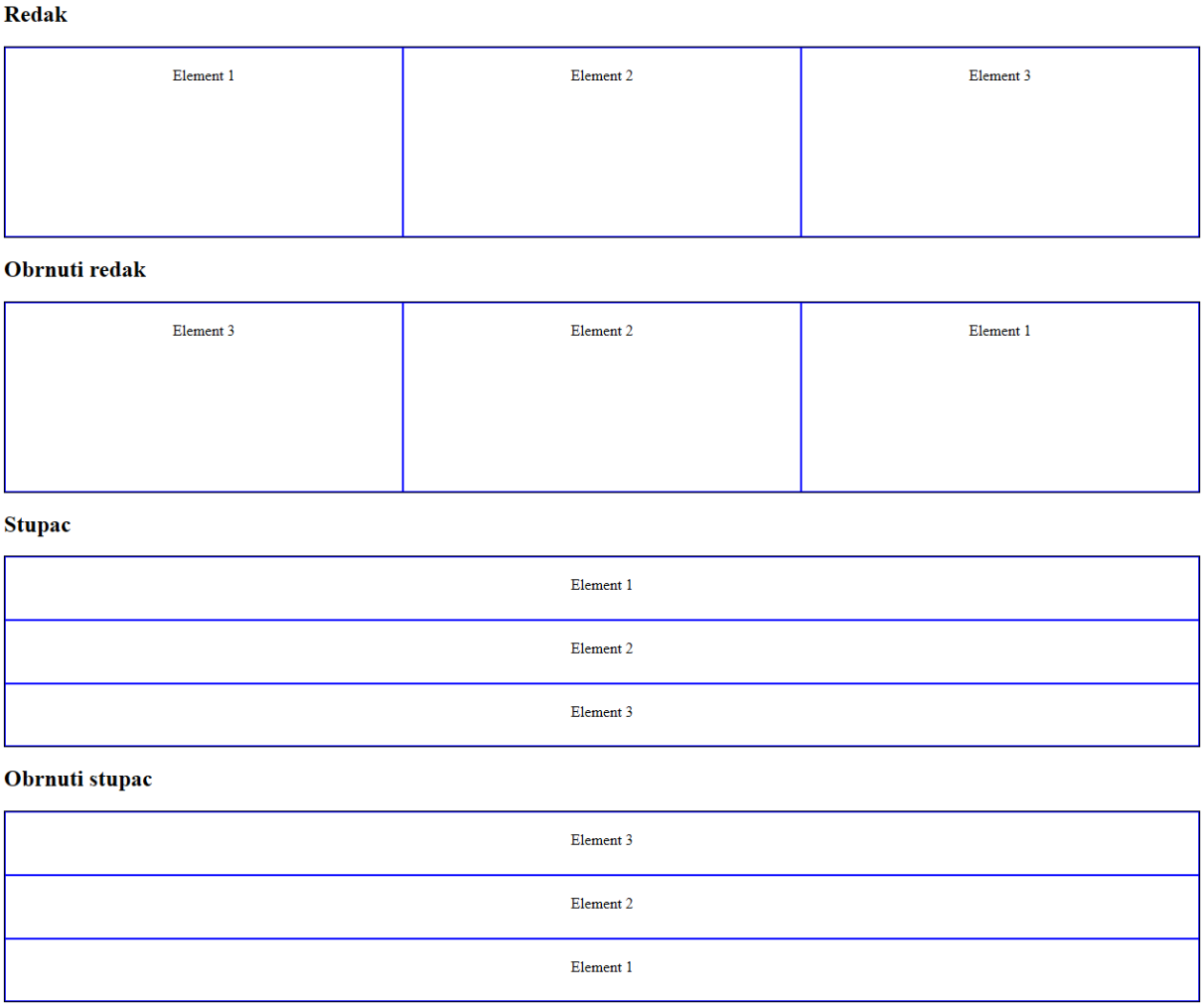
Primjer više stupaca

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet. Sed auctor cursus massa at porta. Integer ligula ipsum, tristique sit amet orci vel, viverra egestas ligula. Curabitur vehicula tellus neque, ac ornare ex malesuada et. In vitae convallis lacus. Aliquam erat volutpat. Suspendisse ac imperdiet turpis. Aenean finibus sollicitudin eros pharetra congue. Duis ornare egestas augue ut luctus. Proin blandit quam nec lacus varius commodo et a urna. Ut id ornare felis, eget fermentum sapien.

Nam vulputate diam nec tempor bibendum. Donec luctus augue eget malesuada ultrices. Phasellus turpis est, posuere sit amet dapibus ut, facilisis sed est. Nam id risus quis ante semper consectetur eget aliquam lorem. Vivamus tristique elit dolor, sed pretium metus suscipit vel. Mauris ultricies lectus sed lobortis finibus. Vivamus eu urna eget velit cursus viverra quis vestibulum sem. Aliquam tincidunt eget purus in interdum. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Slika 2: primjer više stupčanog raspored [7]

U *Flexbox*-u elementi postaju fleksibilni, te se smanjuju ili rastu, raspoređujući prostor između elemenata prema prostoru u njihovom spremniku [7]. Elementima se može odrediti kako da se ponašaju kada imaju više ili manje prostora između sebe. Kako bi postavili *FlexBox* nad spremnikom, koristi se svojstvo "*display: flex". Flexbox*-u je moguće odrediti smjer kojim će elementi ići, pomoću svojstva *flex-direction*. Elementi tako mogu ići u redak ili stupac, kao što je prikazano na sljedećoj slici.

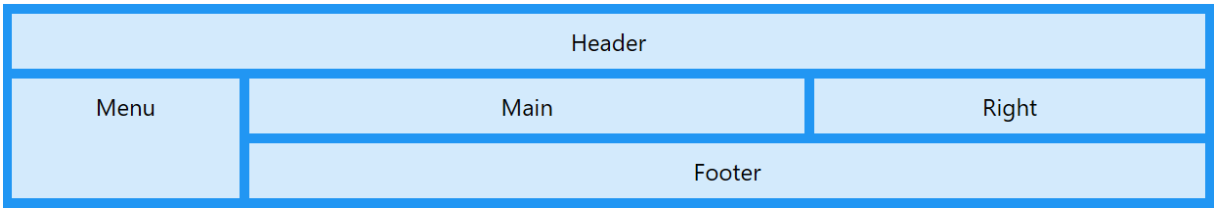


Slika 3: primjer Flexbox-a (vlastita izrada)

Kada raspoređujemo elemente pomoću rešetke, jedinica 'fr' omogućuje raspodjelu dostupnog prostora pomoću traka rešetaka. Pomoću rešetke moguće je napraviti raspored stranice, bez da se koristi pozicioniranje elemenata. Primjer korištenja rešetke [7]:

```
.spremnik {
     display :grid;
     grid-template-columns:1fr 1fr 1fr;
}
```

Na sljedećoj slici moguće je vidjeti raspored stranice koja koristi rešetke:



Slika 4: primjer rešetke [7]

3.3. Responzivne slike/mediji

Kada na web stranicu postavimo neku sliku ili ostali mediji, on nikada ne bi trebao biti veći od njegovog spremnika. Kako bi se to spriječilo koristi se maksimalna dužina elementa za medij:

```
img,
picture,
video {
  max-width: 100%;
}
```

Na ovaj način je medij skaliran i nikada neće prijeći granicu spremnika. Za slike se potrebno pobrinuti da koriste odgovarajući format, npr. PNG ili JPG, i optimizira veličina datoteke prije nego što se postave na web stanicu. Moguće je koristiti i CSS značajke, poput gradijenta i sjena, i tako izbjeći slike [7]. Moguće je koristiti medijske upite nad medijskim elementima, te tako prikazati sliku ili video za različite veličine ekrana.

4. Planiranje web-aplikacije

Planiranje web aplikacije je ključan korak u razvoju web aplikacije. Svrha je što jasnije objasniti kako će aplikacija izgledati, koja arhitektura i tehnologije će se koristiti za izradu. U idućim potpoglavljima ću opisati plan za prema kojem ću izraditi aplikaciju.

4.1. Arhitektura web-aplikacije

Ova web aplikacije koristi troslojnu arhitekturu. Troslojna arhitektura (eng. Three-Tier Architecture) je dobro uspostavljena arhitektura softverskih aplikacija koja organizira aplikacije u tri logička i fizička sloja računalstva: prezentacijski sloj ili korisničko sučelje; aplikacijski sloj, gdje se podaci obrađuju; i sloj podataka, gdje se podaci aplikacije pohranjuju i upravljaju. [8] Glavna prednost troslojne arhitekture je to što svaki sloj radi na vlastitoj infrastrukturi, te se može razvijati, ažurirati i skalirati zasebno.

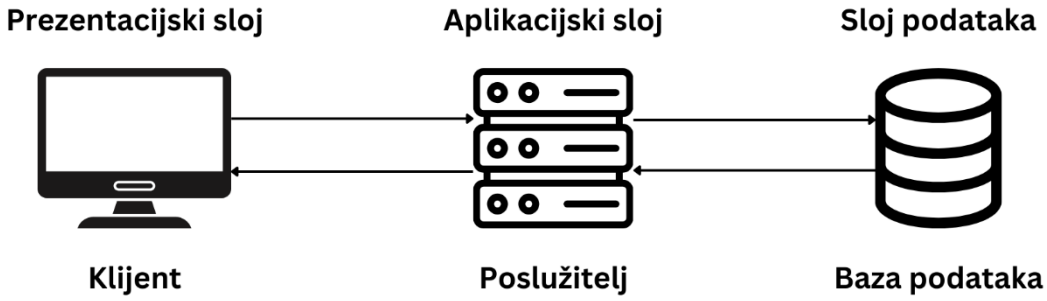
Prezentacijski sloj predstavlja korisničko sučelje i komunikacijski sloj aplikacije, gdje krajnji korisnik komunicira s aplikacijom. [8] Glavna svrha je prikazivanje informacija korisniku i prikupljanje informacija od korisnika. Ovaj sloj se u mojoj aplikaciji izvršava u pregledniku. Prezentacijski sloj može biti napisan u raznim jezicima, a moja je napisan u React programskom okviru, koristeći HTML, CSS i Javascript programske jezike.

Aplikacijski sloj, još poznat kao logički ili srednji sloj, obrađuje podatke prikupljene iz prezentacijskog sloja, korištenjem poslovne logike i poslovnih pravila. [8] Aplikacijski sloj može također dodavati, brisati ili mijenjati podatke u sloju podataka. Ovaj sloj u mojoj aplikaciji predstavlja poslužitelj koji komunicira s web aplikacijom. Razvijen je koristeći Spring Boot programski okvir, koristeći Java programski jezik, ali je moguće i koristiti druge jezike poput: Javascripta, Pythona, PHP-a.

Sloj podataka ili sloj baze podataka je mjesto gdje se pohranjuje i upravlja s informacijama koje obrađuje aplikacija. Aplikacijski sloj iz nje čita podatke i upisuje u nju. [8] Moja aplikacija koristi PostgreSQL relacijsku bazu podataka za spremanje podataka. Još neke od mogućih baza podataka su: MySQL, Microsoft SQL Server, MongoDB.

Troslojna arhitektura je jedna od najčešće korištenih arhitektura u razvoju web aplikacija, te sa sobom nosi nekoliko benefita. Razvoj troslojne arhitekture je brži, jer svaki sloj može razvijati zaseban tim. Svaki sloj se može skalirati neovisno o ostalima. Osim toga, sigurnost aplikacije je poboljšana, jer prije rada s podacima, aplikacijski sloj može verificirati korisnikov unos, i tako osigurati da korisnik ne radi nedozvoljene radnje, koje bi mogao

napraviti ukoliko bi se sva logika prebacila na prezentacijski sloj. Kod na aplikacijskom sloju nije moguće izmjenjivati, jer krajnji korisnik nema pristup do njega.



Slika 5: Troslojna arhitektura (vlastita izrada)

4.2. REST servisi

Kako bi mogli razumjeti što su REST servisi, potrebno je poznavati dva pojma: API i REST. Sučelje za programiranje aplikacija (eng. Application programming interface) definira pravila koja se moraju slijediti kako bi se komuniciralo s drugim softverskim sustavima. Programeri kreiraju API-je kako bi druge aplikacije mogle komunicirati s njihovim aplikacijama. [9] Npr. kada kupac kupi knjigu, napravi se poziv u drugu aplikaciju koja potvrđuje i zapisuje tu kupnju. Reprezentacijski prijenos stanja (eng. Representational State Transfer) je softverska arhitektura koja govori kako API treba raditi. Laka je za implementiranje i modificiranje. [9]

REST API predstavljaju sučelje koje dva računalna sustava koriste za sigurnu razmjenu informacija putem Interneta. Većina aplikacija na neki način mora komunicirati sa svojim drugim aplikacija, bile one interne ili eksterne, kako bi obavile zadatke.

REST API radi na sljedeći način [9]:

1. Klijent šalje zahtjev poslužitelju. Klijent slijedi API dokumentaciju kako bi formatirao zahtjev na način koji poslužitelj razumije
2. Poslužitelj autentificira klijenta i potvrđuje da klijent ima pravo uputiti zahtjev
3. Poslužitelj prima zahtjev i obrađuje ga
4. Poslužitelj vraća odgovor klijentu koji sadrži status odgovora (npr. Je li uspješan) i informacije koje je klijent tražio

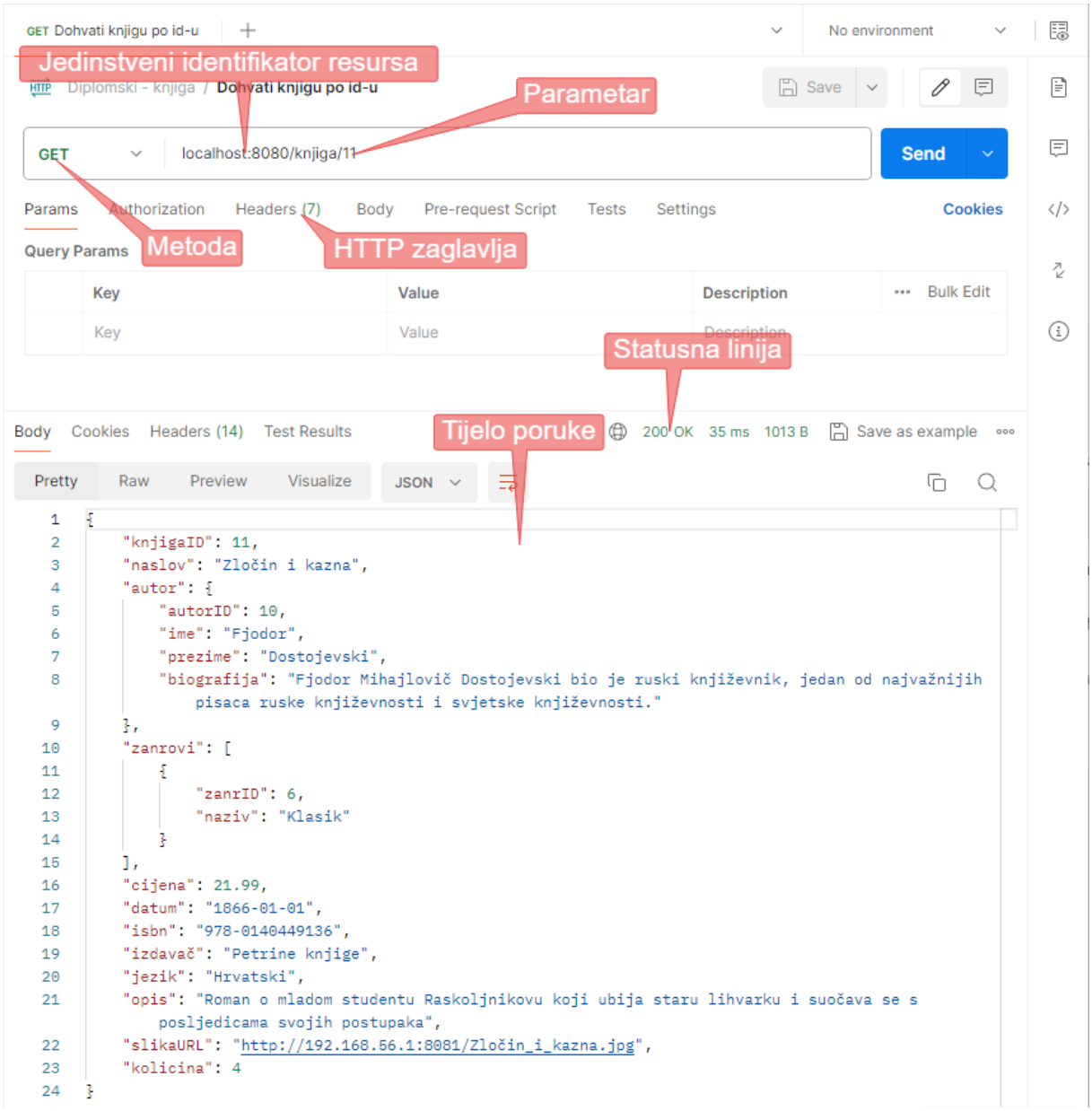
Kada klijent šalje zahtjev (eng. Request), potrebno su sljedeće komponente kako bi poslužitelj znao što napraviti s njim [9]:

1. Jedinstveni identifikator resursa (eng. Uniform Resource Locator) - predstavlja putanju do resursa.
2. Metoda – koriste se HTTP metode za određivanje radnje koju je potrebno izvršiti nad resursom:
   a. GET: dohvaća resurse s poslužitelja
   b. POST: šalje podatke na poslužitelj, često stvara novi resurs
   c. PUT: ažurira postojeći resurse
   d. DELETE: briše resurse
3. HTTP zaglavlja – predstavljaju metapodatke koji se razmjenjuju između klijenta i poslužitelja
4. Podaci – dodatni podatci koji su najčešće potrebni za POST i PUT metode
5. Parametri - dodaju dodatne detalje o onome što treba uraditi

Nakon što poslužitelj dobije zahtjev i obradi ga, šalje odgovor (eng. Response). Odgovor je također standardiziran, i potrebno je sljedeće [9]:

1. Statusna linija – troznamenkasti statusni kod koji označava uspjeh ili neuspjeh (npr. 200 označava uspjeh, 400 označava pogrešan zahtjev)
2. Tijelo poruke - sadrži poruku poslužitelja, najčešće vrijednosti nekog resursa
3. HTTP zaglavlja - predstavljaju metapodatke koji se razmjenjuju između klijenta i poslužitelja

Na slici 6 moguće je vidjeti primjer zahtjeva i odgovora u mojoj aplikaciji. U primjeru se koristi lokalni identifikator resursa, sa metodom GET i osnovnim HTTP zaglavljima. U parametar je postavljen broj 11, jer se traži resurse knjige s id-om 11. Kao odgovor dobiva se statusna linija 200 OK, što znači da je zahtjev uspješno obrađen i dobivaju se podaci o knjizi u JSON formatu. Kako se koristila metoda GET, dodatni podaci nisu bili potrebni u zahtjevu, no npr. u POST metodi dodatni podaci bi strukturno izgledali slično kao i tijelo poruke odgovora.



Slika 6: primjer HTTP zahtjeva i odgovora (vlastita izrada)

Kako bi se zadovoljio REST arhitekturalni pristup, postoje određeni principi koji se moraju poštivati [9]:

- Uniformno sučelje
- *Statelessness*
- Slojni sustav
- Keširanje

Uniformno sučelje govori kako primatelj i poslužitelj moraju prenositi informacije u standardiziranom formatu. *Statelessness* govori kako je svaki zahtjev klijenta neovisan o prethodnim zahtjevima, što znači da poslužitelj može obraditi svaki zahtjev bez da mu je

potrebna prijašnja interakcija. Slojni sustav znači da klijent može komunicirati s poslužiteljem preko više slojeva, a svaki sloj je njemu nevidljiv. Poslužitelj klijentov zahtjev može proslijediti i nekim drugim poslužiteljima, a da to klijent ne zna. Klijenti mogu keširati neke odgovore koji su uvijek isti, kako bi se smanjilo vrijeme odaziva poslužitelja.

4.3. Baza podataka

Kako bi aplikacija znala podatke o knjigama, korisnicima i narudžbama, potrebna je baza podataka u koju će se ti podaci pohranjivati. Baza podataka je organizirana kolekcija strukturalnih informacija ili podataka, koji se obično pohranjuju elektronički na računalu. Bazom podataka upravlja sustav za upravljanje bazama podataka (eng. Database Management System). [10] Postoji više tipova baza podataka, koji se izabiru ovisno o potrebnoj svrsi, poput: relacijski, objektno-orijentiranih, distribucijski, grafičkih. Ja ću u ovom radu koristiti relacijsku bazu podataka (eng. Relational database).

Relacijske baze podataka su najpopularnije baze podataka. U relacijskoj bazi, podaci su strukturirani u tablice. Tablice predstavljaju neki entitet iz stvarnoga života, te se sastoje od stupaca u kojima su opisana svojstva tog entiteta. Svaki redak u tablici je instanca nekog entiteta. Entiteti, tj. tablice su međusobno povezane kroz primarni ili vanjski ključ. Ključevi se ponašaju kao jedinstveni identifikatori koji prikazuju različite odnose između tablica, a ti odnosi su obično ilustrirani kroz različite tipove modela podataka.

Na primjeru knjižare, moguće je imati tablicu knjiga, koja će imati primarni ključ *KnjigaId* i vanjski ključ *AutorId*. Kroz vanjski ključ, tablica knjiga će znati o kojem autoru se radi. Isto tako će druge tablice kroz primarni ključ *KnjigaId* znati o kojoj knjizi se radi. tako se kreiraju relacije između tablica, tj. podataka.

Postoji nekoliko vrsta veza između tablica u relacijskoj bazi podataka, koje je važno razumjeti. Veze ću objasniti kroz primjer u svojoj bazi podataka, a one su [11]:

- Jedan naprema jedan (eng. One-to-one)
- Jedan naprema više (eng. One-to-many)
- Više naprema više (eng. Many-to-many)

Baza u mojoj aplikaciji sastoji se od nekoliko povezanih tablica koje čine osnovnu struktura knjižare. Glavni fokus je na tablici knjiga, koja sadrži informacije o svim knjigama koje su u bazi. Knjiga je povezana sa žanrom koristeći vezu više naprema više, jer svaka knjiga može imati više žanrova, a svaki žanr je moguće pridružiti više knjiga. Zbog toga je također umetnuta pomoćna tablica koja povezuje više žanrova s više knjiga. Žanr je odvojen u posebnu tablicu kako bi se smanjila redundancija i lakše pretraživali žanrovi u aplikaciji. Osim toga,

knjiga je povezana i sa tablicom autor. Autor je povezan s tablicom knjiga koristeći vezu jedan naprema više, jer knjiga može imati samo jednog autora, a autor može imati više knjiga.

U bazi postoje korisnici koji mogu biti kupci ili administratori, ovisno o njihovoj roli koja je zapisana. Kada kupac kupi knjigu, kreira se nova narudžba. U narudžbi je zapisan taj kupac, datum narudžbe i detalji narudžbe. Detalji narudžbe predstavljaju koja knjiga je kupljena i u kojoj količini, te je ona spojena sa narudžbom. Narudžba tako može imati više detalja narudžbi, a detalj narudžbi uvijek pripada određenoj narudžbi. Na slici je prikazan dijagram odnosa entiteta (eng. Entity Relationship Diagram) opisane baze podataka koja će se koristiti u aplikaciji.

Slika 7: Baza podataka web aplikacije (vlastita izrada)

4.4. Korištene tehnologije

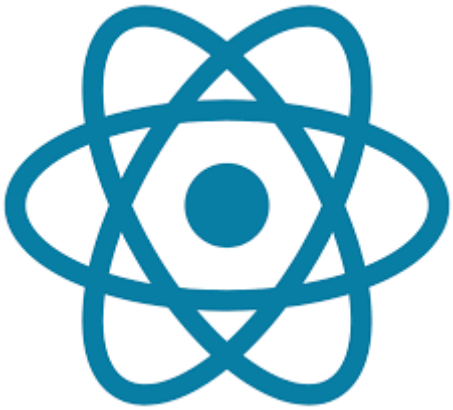
U ovom potpoglavlju bit će opisane tehnologije koje će biti potrebne za izradu web aplikacije. Uz opis svake od tehnologija, bit će opisano i zašto su baš one odabrane za izradu ove web aplikacije.

4.4.1. React

React je Javascript biblioteka za izradu klijentskog dijela aplikacije, točnije za kreiranje korisničkog sučelja. Razvijaju i održavaju ju tvrtka Meta od 2013. godine. Koristi se za razvoj web aplikacija koje koriste jednu stranicu (eng. single page), mobilnih aplikacija ili aplikacija koje se renderiraju na serveru [12].

React za kreiranje sučelja koristi komponente. Te komponente mogu biti mali djelići aplikacije koje je moguće kombinirati u cijele ekrane, stranice ili aplikacije. Komponente se mogu ugnijezditi jedna u drugu. Npr. Komponenta može biti kontejner za knjigu, koji će prikazivati sliku knjige, autora, naslov i cijenu. Ta komponenta se može zvati na više mjesta i ponovno iskoristiti. Komponente su interaktivne, što znači da im je moguće poslati podatke koje će one obraditi, i prikazati na ekranu [12]. Uz komponente, lagano je i korištenje dodatnih vanjski biblioteka, tj. modula, što omogućava lakši razvoj aplikacije.

Stranica se pomoću React-a razvija kao jedna stranica, tj. stranica koja se, kada komunicira s njom, dinamički prepisuje s novim podacima s web poslužitelja, umjesto da se učitava cijela nova stranica u web pregledniku [12]. To može tako što koristi koncept virtualnog DOM-a (eng. Document Object Model) što omogućuje da se promjene prvo primjene na virtualnom DOM-u, a zatim selektivno na stvarnom, s čim se smanjuje broj operacija na stvarnom DOM-u i poboljšava performanse aplikacije. Uz brže prijelaze u aplikacije, aplikacije imaju više osjećaja kao da su nativne aplikacija, a ne standardna web aplikacija.



Slika 8: React logo [12]

Kod odabira biblioteke koja će se koristit za prikaz korisničkog sučelja, kao najbolja opcija činila mi se upravo biblioteka React. React je jedna od najčešće korištenih biblioteka za izradu sučelja aplikacije. Zato što je popularan, sadrži mnoštvo sadržaja na Internetu iz kojega se lako naučiti, npr. dokumentacija i primjeri web aplikacija. Uz to, sadrži i veliki broj dodatnih biblioteka, što može uvelike olakšati razvoj sučelja.

4.4.2. Spring Boot

Spring Boot je alat otvorenog koda koji olakšava korištenje okvira baziranih na Java programsku jeziku za kreiranje mikroservisa i web aplikacija. Koristi se za razvoj nezavisnih, proizvodno kvalitetnih aplikacija. Temelji se na Spring-u, koji je također okvir za izradu mikroservisa u Javi. Spring Boot je u tom pogledu proširenje konvencija nad konfiguracijom za

Spring, kojemu je cilj smanjiti brige oko konfiguracije prilikom stvaranja aplikacija temeljenih na Spring-u. Koristi se pristup mišljenja (eng. Opinionated approach), koji samostalno, koristeći vlastitu prosudbu, odabire koje će pakete instalirati i koje će zadane vrijednosti koristiti. Pristup mišljenja je pristup koji uzima poziciju da postoji jedan način, koji je lakši od ostalih. [13]

Spring Boot dolazi s već ugrađenim web serverima, poput Tomcat-a ili Jetty-a, koji omogućuje pokretanje aplikacije bez potrebe za instaliranjem i konfiguriranje vanjskih servera. Uz to, pruža podršku za kreiranje REST servisa, sigurnost aplikacija, upravljanjem podacima u bazi podataka i integraciju s različitim gotovim bibliotekama.

Slika 9: Spring Boot logo [13]

Kod izrade backenda, glavni razlog odabira Spring Boot-a je to što je jedan od najpopularnijih alata za izradu poslužiteljske aplikacije. Na Internetu postoji veliki broj primjera i dokumentacija, što olakšava učenje i implementaciju. Također, Spring Boot omogućava brzo postavljanje i pokretanje poslužitelja s minimalnom konfiguraciju, što značajno ubrzava razvojni proces. [14]

4.4.3.PostgreSQL

„PostgreSQL je snažan objektno relacijski sustav baza podataka otvorenog koda s više od 35 godina aktivnog razvoja, što mu je donijelo snažnu reputaciju za pouzdanost, robusnost značajki i performansi“. [15] PostgreSQL je zapravo sustav za upravljanje relacijskim bazama podataka (eng. Relationship Database Management System) koju razvija PostgreSQL Global. Verzija 1.0 izašla je 5. rujna 1995. PostgreSQL je besplatan i otvorenog koda, što znači da ga bilo koja osoba ili organizacija može izmjenjivati prema svojim potrebama, te dodavati na njega dodatne funkcionalnosti. Moguće ga je koristiti na svim popularnim operacijskim sustavima poput Windows-a, Linux-a i macOS-a.

Slika 10: PostgreSQL logo [15]

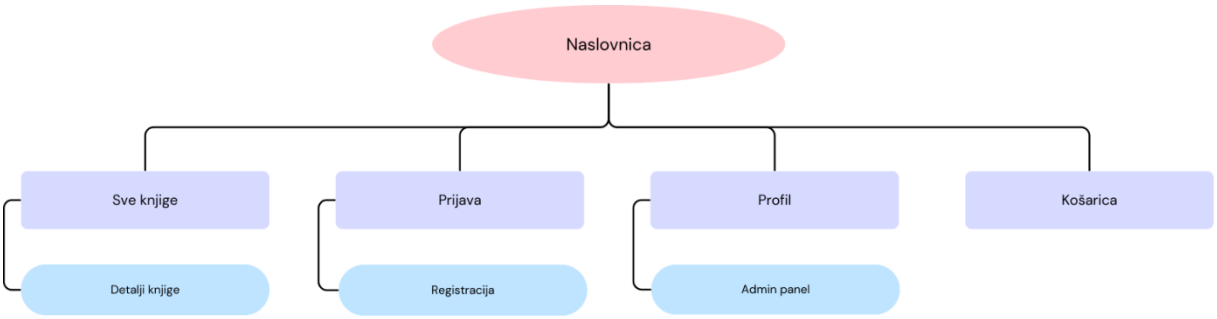
Kod izabira sustava za upravljanje bazom podataka izabrao sam PostgreSQL iz nekoliko razloga. PostgreSQL je najviše korišten sustav za upravljanje bazama podataka, te kao takav ima mnogo dokumentacije i primjera korištenja. Uz to koristi SQL, koji je najpopularniji jezik za rad s bazama podataka.

PostgreSQL podržava razne programske jezike, uključujući Java i Javascript koji su potrebni za klijenta i poslužitelja aplikacije. Kako ću raditi s klasama na poslužiteljskom dijelu, moguće je koristiti tehniku objektno-relacijskog mapiranja (eng. Object-relational mapping). tako će se klasa ispisati izravno iz baze, što je moguće zato što je PostgreSQL relacijska baza. Također je besplatan za korištenje u bilo koju svrhu. Moguće ga je instalirati na bilo kojem sustavu, te će raditi na isti način.

4.5. Struktura web stranice

Prije same izrade web stranice važno je znati kako će se korisnik moći kretati kroz aplikaciju. Potrebno je napraviti neke osnovne kretnje na samom početku planiranja izrade web stranice. Kroz izradu aplikacije, često se dešavaju promjene, pa je potrebno izmijeniti neke kretnje koje se predviđene ili zaboravljene. Zbog toga je to često iterativan proces.

Na slici je prikazana finalna karta web aplikacije, tj. tok kojim će se korisnik moći kretati po aplikaciji.



Slika 11: karta web aplikacije

Na karti možemo vidjeti kako sve kreće od naslovne stranice. Na naslovnici će se prikazivati nekoliko knjiga. Svaka stranica će imati navigacijsku traku, kroz koju će se moći ići na jednu od sljedećih stranica: sve knjige, prijavu, profil ili košaricu. Važno je naglasiti kako će se gumb za prijavu prikazivati samo ako korisnik nije prijavljen, inače će se prikazivati gumb za profil.

Na stranici sve knjige prikazivat će se sve knjige. Za knjigu će pisati osnovne informacije: naziv, autor i cijena; i bit će prikazana slika knjige. Uz to, korisnik će na toj stranici moći filtrirati knjige prema autoru, žanru, cijeni; te ih sortirati prema nazivu ili cijeni. Pritiskom na neku od knjiga otvorit će se stranica s detaljima o knjizi. Na njoj će biti prikazani detalji o knjizi, koji na prethodnim stranicama nisu bili prikazani. Knjiga će se moći staviti u košaricu iz bilo kojeg ekrana gdje su knjige prikazane.

Na stranici prijava, korisnik će moći obaviti prijavu. Ukoliko nije registriran, iz stranice prijava će moći pristupiti stranici za registraciju te se tamo registrirati.

Na stranici profil, koja je vidljiva samo nakon što se korisnik prijava, korisnik će moći vidjeti i promijeniti svoje osobne podatke i lozinku. Osim toga moći će vidjeti i svoje narudžbe. Ako je korisnik administrator, prikazat će se gumb koji vodi na Admin panel. Iz njega, administrator će moći uređivati knjige i dodavati ih u bazu podataka.

U košarici će korisnik moći vidjeti sve proizvode koje je dodao te ih izmjenjivati.

5. Implementacija aplikacije

U idućim poglavljima opisati ću razvoj web aplikacije. Izrada višeslojne aplikacije je kompliciranа, te jedna promjena na poslužiteljskom dijelu može uzrokovati promjenu da drugom dijelu aplikacije. Iz toga razloga je proces izrade sveukupno aplikacije iterativan proces. U trenutku pisanja rada aplikacija je izrađena, tako da će biti opisano finalno stanje aplikacije.

Prvo ću opisati izradu poslužiteljskog dijela aplikacije, jer klijentski dio ne može funkcionirati bez njega.

5.1. Implementacija poslužitelja

Koristeći Visual Studio Code (skraćeno VSCode) IDE, moguće je kreirati Spring Boot projekt koristeći *addon*. Najlakši način za kreiranje novog Spring Boot projekta moguć je kroz Spring Initializr, mini program koji generira projekt ovisno o korisnikovim željama i potrebama. Spring Initializr moguće je koristiti u pregledniku ili kroz VSCode. Inicijalizacija se sastoji od 7 koraka, koje sam zapisao zajedno sa svojim odabranim mogućnostima:

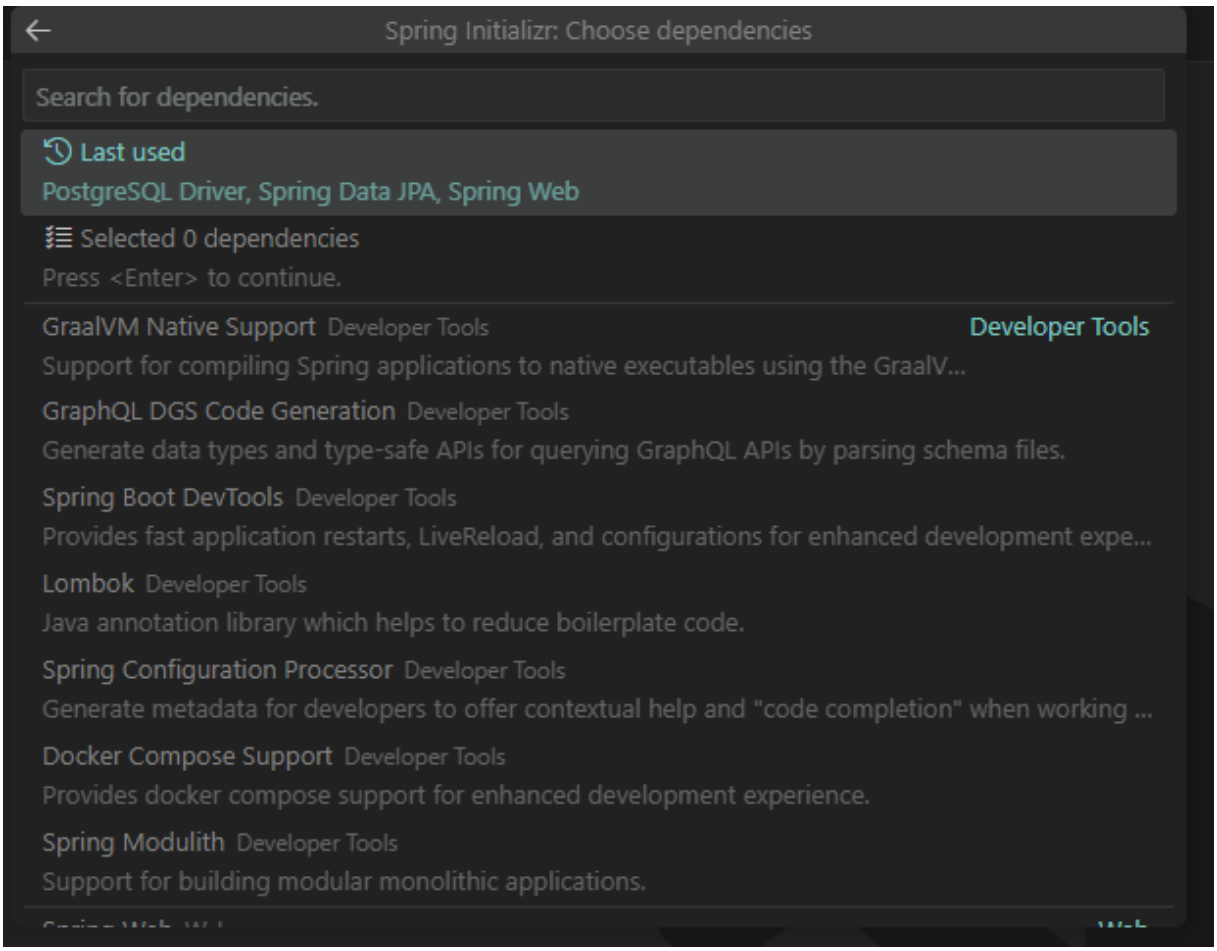
1. Specificirati Spring Boot verziju – verzija 3.2.5
2. Specificirati jezik projekta – Java
3. Postaviti id grupe – org.mvukasovi
4. Postaviti id artifakta – knjizara
5. Specificirati tip paketa – jar
6. Specificirati verziju Jave – 17
7. Dodati vanjske biblioteke

Kako bi mogli napraviti REST API pozive, potrebne su sljedeće vanjske biblioteke pomoću kojih je moguće definirati REST pozive i komunicirati s bazom:

- spring-boot-starter-data-jpa
- spring-boot-starter-web
- postgresql
- lombok

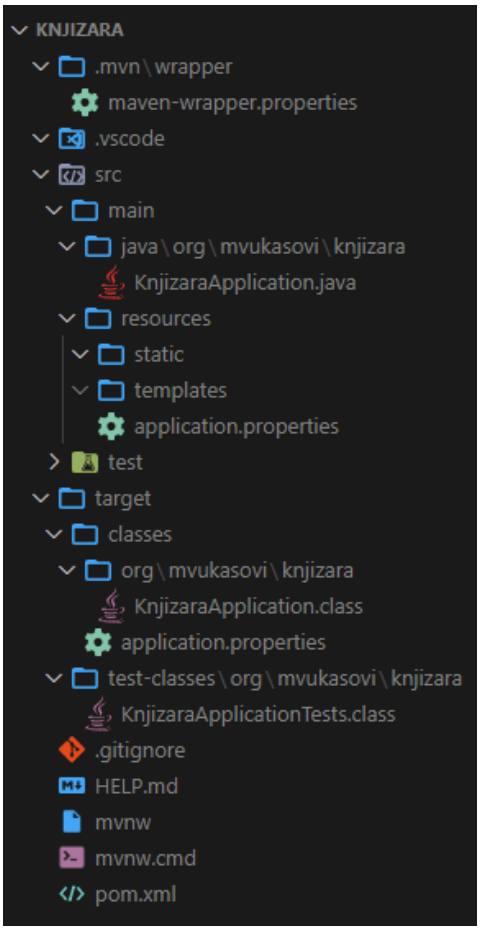
Od svih biblioteka, posebno bih istaknuo *spring-boot-starter-data-jpa*. To je biblioteka u kojoj se nalazi „Jakarta Persistance API“ , okvir koji pruža skup specifikacija i sučelja za upravljanje relacijskim podaci u Java aplikaciji. Omogućuje programerima mapiranje Java objekata na relacijske baze podataka koristeći objektno-relacijsko mapiranje (skraćeno ORM).

ORM pruža jednostavan način za interakciju s bazom podataka bez potrebe za pisanje SQL koda. [16]



Slika 12: korak inicijalizacije Spring Boot-a (vlastita izrada)

Nakon što je projekt inicijaliziran dobivamo strukturu na slici 12. Imamo nekoliko važnih datoteka i mapa. Neke od tih datoteka nisu važne za naš razvoj, već služe kako bi Spring Boot znao kako izgraditi aplikaciju, poput datoteka u mapi .mvn, ili datoteka mvnw i mvnw.cmd. Za developere je bitna datoteka *pom.xml*. U njoj se nalaze sve ranije postavke koje su se postavile kroz Spring Initializr. Iduća važna datoteka je *application.properties*, u koju možemo postaviti svojstvo koje će se koristiti na razini cijele aplikacije, npr. ime aplikacije ili podaci o povezivanje na bazu podataka. Još su nam ostale dvije važne datoteke. Obje od njih se u ovom slučaju zovu „KnjizaraApplication“, jedna završava sa *.java*, a druga sa *.class*. Datoteke s ekstenzijom *.java* koriste se za pisanje koda, a datoteke s ekstenzijom *.class* su rezultat bytecode koji je dobiven kompiliranjem *.java* datoteke. „KnjizaraApplication.java“ tako sadrži *main* funkciju, iz koje kreće izvršavanje aplikacije.



Slika 13: Spring Boot inicijalan direktorij (vlastita izrada)

Postoji konvencija prilikom izrade Spring Boot projekata, a to je način na koji su mape i datoteke organizirane. Dobra praksa je mape unutar projekta trebale bi se podijeliti na četiri dijela: kontroleri, modeli, vanjske točke i repozitoriji. Kako je moja aplikacije jednostavna u pogledu rada s podacima, s malo biznis logike, u svom projektu ću spojiti kontroler zajedno s vanjskim točkama. Model opisuje podatke kojima aplikacija radi. U kontrolerima se nalazi vanjske točke, u kojima je zapisan jedinstveni identifikator resursa. Kontroler dobiva podatke iz zahtjeva i manipulira modelom. Repozitorij se koristi za upravljanje ORM-om, tj. za komuniciranje s bazom podataka. Kroz iduća poglavlja ću detaljnije opisati i prikazati svaki od tih dijelova.

5.1.1. Model

Model u Spring Boot-u predstavlja klasu. Iznad klase postavlja se anotacija *@Entity*, što znači da će se ova klasa pomoću ORM-a mapirati na istoimenu tablicu u bazi podataka. Kako koristim *update* na *hibernate*, svaki put kada se model ažurira, ažurirat će se istoimena tablica u bazi podataka. Svaka varijabla u klasi povezana je sa stupcem u bazi podataka. Nad varijable moguće je stavljati dodatne anotacije koje pobliže opisuju tu varijablu npr. anotacija

@Id prema kojoj ORM zna da ta varijabla u bazi predstavlja *id* stupac. Pomoću anotacija moguće je i postavljati zabrane, poput toga *@NonNull* koji će zabraniti da varijabla bude prazna prilikom upisivanja u bazu.

Sljedeći primjer prikazuje jedan od modela u mojoj aplikaciji:

```
@Entity
public class Knjiga {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int knjigaID;
    private String naslov;
    @ManyToOne
    @JoinColumn(name = "autorID", referencedColumnName = "autorID")
    private Autor autor;
    @ManyToMany
    @JoinTable(
        name = "knjiga_zanr",
        joinColumns = @JoinColumn(name = "knjigaID"),
        inverseJoinColumns = @JoinColumn(name = "zanrID")
    )
    private Set<Zanr> zanrovi;
    private float cijena;
    private Date datum;
    private String isbn;
    private String izdavač;
    private String jezik;
    @Column(length = 1000)
    private String opis;
    private String slikaURL;
    private int kolicina = 0;
}
```

Proći ću kroz anotacije koje se ovdje spominju. *@Id* konkretno označava primarni ključ u tablice. Ispod njega nalazi se *@GeneratedValue* i govori kako će se vrijednost primarnog ključa automatski generirati, pomoću strategije *IDENTITY*, što znači da će automatski generirati od strane baze. *@Column length* specificira maksimalnu duljinu teksta koji je moguće pohraniti za taj stupac u bazi podataka.

Ovdje se anotacije koriste i za već prije spomenute veze između tablica. Prva takva anotacija je *@ManyToOne*, koja označava da postoji veza više naprema jedan između tablica *Knjiga* i *Autor*, konkretno da jedna knjiga može imati jednog autora, a jedan autor može imati

više knjiga. *@JoinColumn* specificira ovu vezu, gdje govori kako se u tablici Knjiga referencira strani ključ *AutorID* koji povezuje ovu tablicu s tablicom *Autor*.

Koristi se i anotacije *@ManyToMany*, koja označava vezu više naprema više, između tablica *Knjiga* i *Žanr*. Svaka knjiga može imati više žanrova, i svaki žanr može pripadati više knjiga. *@JoinTable* specificira međutablicu, koja povezuje knjige i žanrove. Kao parametre prima ime međutablice, i dva stupca koja su povezana, a to su id knjige i id žanra.

Kroz aplikaciju su svi modeli napravljeni na ovaj način, tako da ih neću ponovno objašnjavati, te ću ih samo nabrojati: *Knjiga, Autor, Zanr, Korisnik, Narudzba i DetaljiNarudzbe*. Uz varijable koje ti modeli koriste, još sadrže i funkcije za dohvaćanje i postavljanje podataka za svaku varijablu (eng. Getter i Setter), te konstruktor za inicijalizaciju objekta

Uz modele, postoji i jedan sličan koncept koji se zove objekt prijenosa podataka (eng. Data Transfer Object). Ti objekti se koriste kako bi se strukturirali podaci koji se šalju s klijenta i lakše obradili na poslužitelju. Oni se također definiraju kao klase sa varijablama, ali ne sadrže anotacije jer ne komuniciraju s bazom, te mogu imati varijable samo one koje su potrebne da se izvrši neka akcija. U ovoj aplikaciji se ti objekti koriste.

5.1.2. Repozitorij

Repozitorij u Spring Boot-u je sučelje koje služi kao most između poslovne logike aplikacije i baze podataka. Ono opisuje kako aplikacija komunicira s bazom podataka. Omogućuje više objektno orijentiran pristup u radu s bazom podataka. Postoje dva tipa repozitorija u Spring Boot-u, a to su: *CrudRepository* i *JpaRepository*. *CrudRepository* se koristi za vrlo jednostavne operacije za kreiranje, dohvaćanje, ažuriranje i brisanje podataka, dok se *JpaRepository* koristi za kompleksnije upite. Ja ću u aplikaciji koristiti *JpaRepository*. Za kreiranje repozitorija moguće je iznad sučelja postaviti anotacije *@Repository*, no ona nije potrebna ako sučelje proširuje *JpaRepository* sučelje. Idući kod prikazuje djelić repozitorija Knjiga u aplikaciji:

```
public interface KnjigaRepositorij extends JpaRepository<Knjiga,
Integer> {
  List<Knjiga> findByAutor_AutorID(Integer autorID);


  @Query("SELECT k FROM Knjiga k WHERE LOWER(k.autor.ime) LIKE
LOWER(CONCAT('%', :name, '%')) OR LOWER(k.autor.prezime) LIKE
LOWER(CONCAT('%', :surname, '%'))")


    List<Knjiga> findByAutorNameOrSurname(@Param("name") String name,
@Param("surname") String surname);
}
```

Prvo možemo vidjeti kako sučelje „KnjigaRepozitorij“ proširuje sučelje *JpaRepository*. Njemu se definira da koristi tablicu *Knjiga* i da je primarni ključ tipa integer. Sučelje sadrži dvije metode. Prva metoda, *findByAutor_AutorID()* prima kao parametar id autora, a vraća listu knjiga tog autora. Ova metoda koristi JPA funkcionalnost “Query Derivation” koja automatski generira upit na temelju naziva metoda. JPA zna kako je potrebno tražiti u tablici autor njegov id.

Ukoliko je potreban kompleksniji upit, može se napisati kao u drugoj metodi. Druga metoda traži autora prema imenu ili prezimenu, te vraća listu knjiga koje je autora napisao. Kako bi postavili svoj upit, potrebno je koristiti anotaciju *@Query*. U nju se upisuje SQL upit. Koristi se još i anotacija *@Param*, koja mapira argumente metode s odgovarajućim parametrima u upitu. Parametri u upitu će poprimiti vrijednosti argumenata dok se upit izvrši.

Ostali repozitoriji u aplikaciji se ponašaju na sličan način. U aplikaciji se nalaze sljedeći repozitoriji: *KnjigaRepozitorij, AutorRepozitorij, KorisnikRepozitorij, NarudzbeRepozitorij*.

5.1.3. Kontroler

Kontroleri u Spring Boot-u su klase koju su zadužene za upravljanje dolaznih HTTP zahtjeva i vraćanja odgovora na iste. Ponašaju se kao posrednici između klijenta (u mom slučaju web aplikacije) i poslovne logike aplikacije. Jednostavno rečeno, kontroleri primaju zahtjev, obrađuju ga i vraćaju odgovor. Na klasu je potrebno dodati anotaciju *@RestController* kako bi je označili kao kontroler. Uz to, potrebno je postaviti anotaciju *@RequestMapping*, koja se koristi za mapiranje web zahtjeva na metode u kontroler klasama koje obrađuju zahtjevima. Na primjeru klase kontrolera knjige ću u potpunosti objasniti kontrolere. [17]

```
@RestController
@RequestMapping("/knjiga")
public class KnjigaKontroler {
    @Autowired
    private KnjigaRepositorij knjigaRepozitorij;

@GetMapping("/")
    public List<Knjiga> dohvatiSveKnjige(
            @RequestParam(required = false) String autor,
            @RequestParam(required = false) String zanr,
            @RequestParam(required = false) String jezik,
            @RequestParam(required = false) Float minCijena,
            @RequestParam(required = false) Float maxCijena,
            @RequestParam(required = false) String sort) {
        List<Knjiga> knjige = new ArrayList<>();
```

```
        knjigaRepozitorij.findByFilters(autor, zanr, jezik, minCijena,
maxCijena).forEach(knjige::add);
        return knjige;
    }
```

Prije imena klase potrebno je postaviti već spomenute anotacije *@RestController* i *@RequestMapping*. *@RequestMapping* anotaciju postavljamo na jedinstveni identifikator na kojem želimo da nam se resurs nalazi. U ovoj klasi je to postavljeno na */knjige*, sukladno tome, puna domena za pristup klasi knjiga glasila bi: *adresa_poslužitelja/knjiga*. U klasu se dodaju potrebni repozitoriji koristeći anotaciju *@Autowired*, koja se koristi za automatsko umetanje ovisnosti (eng. Dependency injection). Omogućuje da se ovisnost umetne automatski, bez da je ručno potrebno stvarati objekte.

Nakon toga, dodana je prva funkcija *dohvatiSveKnjige()*. Iznad nje je korištena anotacije *@GetMapping*, koja mapira na sličan način kao i *@RequestMapping*, mapira metodu na HTTP GET zahtjev, na rutu "/". Dovoljno je koristiti poziv *adresa_poslužitelja/knjiga/*, kako bi se dohvatile sve knjige. Funkcija vraća listu knjiga u JSON formatu. Kao parametri funkcije traže se određene String varijable, nad kojima je zadana anotacija *@RequestParam*.

Dohvaćanje svih resursa jedna je od osnovnih operacija u REST servisima. Osim toga, često se implementiraju pozivi za kreiranje, čitanje, ažuriranje i brisanje resursa (skraćeno CRUD). Ovdje je već napravljen jedan od tih poziva, a to je čitanje. Sljedeći kod prikazuje dodavanje, ažuriranje i brisanje resursa:

```
@PostMapping("/")
    public ResponseEntity<Knjiga> dodajKnjigu(@RequestBody KnjigaDTO
knjigaDTO) {
        Optional<Autor> autorOptional =
autorRepozitorij.findByImeAndPrezime(knjigaDTO.getAutorIme(),
                knjigaDTO.getAutorPrezime());
        Autor autor = autorOptional.orElseGet(() -> new
Autor(knjigaDTO.getAutorIme(), knjigaDTO.getAutorPrezime()));
        Knjiga knjiga = new Knjiga(
                knjigaDTO.getNaslov(),
                autor,
                knjigaDTO.getCijena(),
                knjigaDTO.getDatum(),
                knjigaDTO.getIsbn(),
                knjigaDTO.getIzdavac(),
                knjigaDTO.getJezik(),
                knjigaDTO.getOpis(),
                knjigaDTO.getSlikaURL(),
                knjigaDTO.getKolicina());
```

```
        if (!autorOptional.isPresent()) {
            autorRepozitorij.save(autor);
        }
        knjigaRepozitorij.save(knjiga);
        return ResponseEntity.ok(knjiga);
    }

@PutMapping("/{id}")
    public String azurirajKnjigu(@PathVariable int id, @RequestBody
Knjiga novaKnjiga) {
        return knjigaRepozitorij.findById(id)
                .map(knjiga -> {
                    knjiga.setNaslov(novaKnjiga.getNaslov());
                    knjiga.setCijena(novaKnjiga.getCijena());
                    knjiga.setDatum(novaKnjiga.getDatum());
                    knjiga.setIsbn(novaKnjiga.getIsbn());
                    knjiga.setIzdavač(novaKnjiga.getIzdavač());
                    knjiga.setJezik(novaKnjiga.getJezik());
                    knjiga.setOpis(novaKnjiga.getOpis());
                    knjiga.setKolicina(novaKnjiga.getKolicina());
                    knjigaRepozitorij.save(knjiga);
                    return "Knjiga uspješno ažurirana";
                }).orElse("Knjiga nije pronađena");
    }

@DeleteMapping("/{id}")
    public String obrisiKnjigu(@PathVariable Integer id) {
        knjigaRepozitorij.deleteById(id);
        return "Knjiga uspješno obrisana";
    }
```

Metoda za dodavanje knjige koristi anotaciju *@PostMapping* koja mapira metodu na HTTP POST zahtjev. Funkcija vraća *ResponseEntity*, tip podataka koji enkapsulira HTTP odgovor, dajući statusni kod, zaglavlje i tijelo, a u tijelu se nalaze podaci o dodanoj knjizi. U parametrima funkcije dodana je jedna nova anotacija *@RequestBody*. *@RequestBody* mapira tijelo poruke u varijablu. Ta varijabla mora imati identičnu strukturu kao i tijelo poruke, te se iz tog razloga koristi već ranije spomenuti DTO. U funkciji se prvo dohvaća autor knjige koje se želi upisati u bazu. Ako autor ne postoji, on se onda kreira. Nakon toga se kreira knjiga koja

se sprema u bazu koristeći metodu *save()* i vraća se odgovor statusnog koda „200 Ok“, zajedno s podacima o knjizi.

Metoda za ažuriranje knjige koristi anotaciju *@PutMapp*ing koja mapira metodu na HTTP PUT zahtjev. Funkcija vraća String, koji govori je li knjiga uspješno ažurirana ili daje poruku greške. Ona sadrži jednu novu anotaciju u parametrima, a to je *@PathVariable*. Pomoću te anotacija, mapira se vrijednost iz putanje URL-a na istoimenu varijablu. Metoda pokušava pronaći knjigu prema id-u, te ako je nađe, ažurira podatke koji su postavljeni u tijelo poruke.

Metoda za brisanje knjige koristi anotaciju *@DeleteMapping* koja mapira metodu na HTTP DELETE zahtjev. Metoda vraća String s opisom da je knjiga uspješno obrisana. Metoda ne koristi niti jednu novu anotaciju. Metoda pokušava obrisati knjigu prema id-u, te ako je pronađe, obrisat će je iz baze podataka.

Većina kontrolera u web aplikaciji napisana je na sličan način, koristeći samo metode koje će biti potrebne u web aplikaciji. Ovdje ću popisati sve putanje u aplikaciji. U tablici 2 moguće je vidjeti sve vanjske točke u aplikaciji i kako ih pozvati.

Tablica 2. Popis ruta u aplikaciji (vlastita izrada)

Kontroler	Metoda	Ruta	Opis
/knjiga	GET	/	Dohvaća sve knjige
	GET	/{id}	Dohvaća knjigu prema id-u
	POST	/	Dodaje knjigu
	PUT	/{id}	Ažurira knjigu prema id-u
	PUT	/{id}/kvantitet	Ažurira kvantitet knjige prema id-u
	PUT	/{id}/slika	Ažurira sliku knjige prema id.u
	DELETE	/{id}	Briše knjigu prema id-u
	GET	/trazi	Traži knjigu prema naslovu
	GET	/slicne/{id}	Dohvaća slične knjige prema id-u
/autor	GET	/	Dohvaća sve autore
/zanr	GET	/	Dohvaća sve žanrove
/korisnik	POST	/autentifikacija	Autentifcira korisnika
	POST	/kreiraj	Registrira korisnika

	GET	/{korIme}	
	PUT	/{korisnikId}	
	PUT	/lozinka/{korisnikId}	

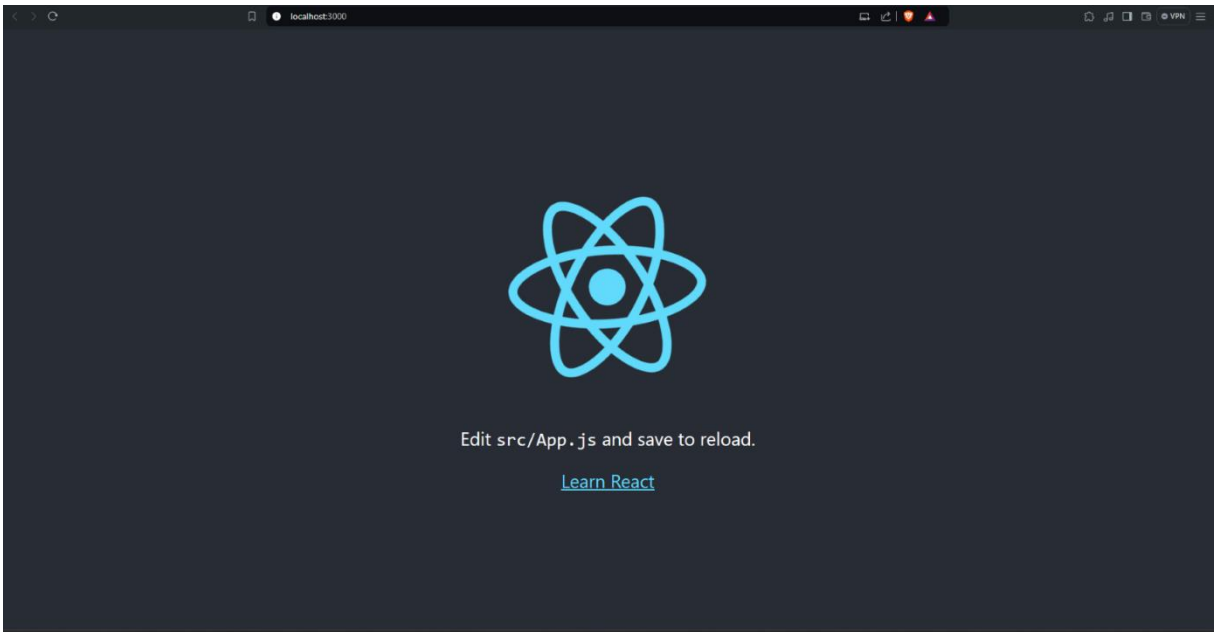
5.2. Implementacija klijenta

U ovom poglavlju ću opisati izradu klijentskog dijela aplikacije. Kao pomoć pri izradi korisničko sučelja koristio sam „*MDN Web Docs*“ dokumentaciju za HTML, CSS i Javascript, te službenu React dokumentaciju. Prvo ću opisati inicijalizaciju React projekta i njegovu strukturu, zatim što su komponente te opisat sve web stranice i način na koji je postignut responzivan dizajn.. Kroz svaku stranicu ću objasniti neke od koncepata rada s React-om i koncepata responzivnog dizajna. Responzivan dizajn ću testirati kroz preglednik koristeći emulirane ekrane:

- Računalo – rezolucija 1920x1080
- Tablet (iPad) – rezolucija 768x1024
- Mobitel (iPhone 14 Pro Max) – rezolucija 430x932

5.2.1. Inicijalizacija React-a

Kako bi napravili frontend, tj. kreirali sučelje, potrebno je kreirati React projekt. Nakon instalacije Node.Js programskog alata na računalo, potrebno je otvoriti terminal i u njega upisati: *npx create-react-app ime_aplikacije.* Nakon što se projekt kreira, možemo ga pokrenuti koristeći komandu *npm run start*. Projekt se pokreće lokalno, na adresi *localhost:3000.* Početni ekran aplikacije izgleda kao na slici.

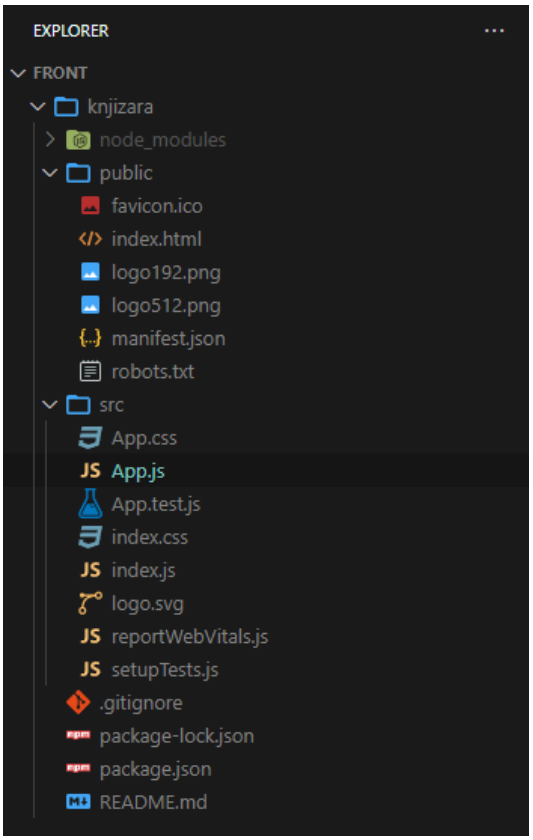


Slika 14: Početni ekran React web aplikacije

Nakon kreiranja projekta, dobivamo početnu strukturu kao na slici. U mapi *src* nalazi se kod aplikacije. Dobra praksa je u toj mapi napraviti dvije mape, *assets* i *components*. U *assets* folder se dodaju sve datoteke koje moraju na neki način biti u aplikaciji, npr. slike. U mapu *components* dodaju se komponente aplikacije. Još su važne datoteke *package.json* i *package-lock.json* u koje se zapisuju preuzeti moduli. U mapi *node_modules* nalaze se ti preuzeti moduli. Neki od modula su automatski dodani prilikom inicijalizacije aplikacije. Programeri sami mogu dodavati module pomoću komande *npm i ime_modula*. Ja ću dodati još nekoliko potrebnih modula:

- Axios
- React-responsive-carousel
- React-select
- React-tabs

Axios je HTTP klijent baziran na obećanjima (eng. Promises) za web preglednike. Njega ću koristiti kako bih slao HTTP zahtjeve na poslužitelja. *React responsive carousel* je modul kojim se vrlo može napraviti karusel u pregledniku. U mojoj aplikaciji, na početnoj stranici koristit će se karusel za prikaz nekoliko knjiga koje će se okretati i tako izmjenjivati. Ostali moduli olakšavaju responzivnost, a bit će opisani prilikom njihovog korištenja.



Slika 15: React inicijalan direktorij

5.2.2.Komponente

React web aplikacija se sastoji od komponenti. Većina komponenti predstavlja neku od web stranica kojima je moguće pristupiti kroz rutu. Neke komponente služe kako bi se u njih zapisali podaci koji ostaju zapisani tijekom rada aplikacija, a to su kontekst komponente. Također postoje komponente koje su sadržane u jednoj ili više komponenti jer ih je moguće koristiti više puta. Svaka komponenta, koja služi za prikaz neke od stranica koristi .js datoteku za Javascript programski kod, i .css datoteku za stiliziranje HTML elemenata. HTML elementi se nalaze unutar Javascript datoteke. Sljedeći kod prikazuju osnovnu strukturu komponente:

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import "./Knjige.css";
const Knjige = ({ filteri, sort }) => {
  const [knjige, postaviKnjige] = useState([]);
  return (
    <div className="knjige-grid">
      {knjige.map((knjiga, index) => (
            // Kod za HTML elementa
      ))}
```

```
      </div>
    );
};
export default Knjige;
```

Strukturu komponente možemo podijeli na sljedeće dijelove: uvozi (eng. imports), deklaracija komponente, deklaracija varijabla i metoda komponente, renderiranje HTML-a i izvoz komponente. U uvozima možemo uvesti vanjske funkcionalnosti biblioteka, drugih komponenta, te možemo uvesti datoteke poput slika, ili css stilova. U deklaraciji komponente deklariramo sam naziv komponente koji uvijek mora početi velikim slovom. U njoj se također mogu nalaziti parametri, tj. *props*, koje komponenta prima iz neke druge komponente, te tako dobiva informacije. Npr. ovdje parametri govore koje filtere i sortiranje komponenta treba koristiti. Nakon toga unutar komponente deklariraju se varijable i metode koje ta komponenta treba koristiti. Npr.*const [knjige, postaviKnjige] = useState([]);* deklarira niz knjiga, čije je početno stanje prazno, i metodu koja će postaviti knjige u niz. Zatim dolazi *return* dio komponente, koja vraća JSX, koji definira kako će izgledati korisničko sučelje. JSX je sintaksna ekstenzija za Javascript, koja koristi HTML za strukturiranje, ali joj je moguće dodavati varijable i metode koje su definirane unutar komponente. Na primjer nakon što se niz knjige napuni, metoda *knjige.map()* će za svaku od knjiga kreirati HTML strukturu koja je unutra. Na kraju dolazi *export default* koji omogućuje da se komponenta Knjige koristi u ostalim komponentama. U sljedećim poglavljima opisati ću svaku izrađenu komponentu unutar aplikacije.

5.2.3. Navigacijska traka

Navigacijska traka je komponenta koja se koristi na svim stranicama u aplikaciji. Iz nje je moguće otići na sve proizvode, pretraživati proizvode, otići u košaricu, prijaviti se i ući u svoj profil. Na slici je prikazana izrađena navigacijska traka za računalo, tablet i mobitel

Slika 16: navigacijska traka

U navigacijskoj traci uvezena je nativna biblioteka React-a, r*eact-router-dom,* te njezina komponenta Link i njezina kuka *useNavigate*. *Link* je komponenta koja se koristi za navigaciju unutar aplikacije, bez potrebe za ponovnim učitavanjem aplikacije. Na taj način se postiže web stranica koja koristi jednu stranicu. U kodu se postavlja kao <*Link to="/proizvodi"*>, koji vodi korisnika na stranicu s proizvodima. U tu komponentu moguće je postaviti sliku, gumb ili proizvoljan element koji će na klik voditi do određene stranice.

Na navigaciji postoji i tražilica, u koju je moguće upisati ime knjige. Nakon što se u element upiše ime knjige, i pritisne *Enter* ili gumb pretraži, izvršava se funkcija *rukujPretrazi*, koja koristi kuku *useNavigate* kako bi se stranica preusmjerila na rutu s proizvodima koja u sebi ima upit iz tražilice. Kod za *rukujPretrazi* je sljedeći:

```
const rukujPretrazi = async (e) => {
    e.preventDefault();
    if (trazilicaUpit.trim() !== "") {
      navigate(`/proizvodi?trazi=${trazilicaUpit}`);
    }
  };
```

Kako bi se postigao responzivan dizajn, navigacijska traka je podijeljena na dva dijela, lijevi i desni. U lijevom dijelu nalazi se logo, gumb i tražilica; a u desnom košarica i korisničke opcije. Cijela navigacijska traka je flex-box, koja za veće ekrane, računalo i tablet, ima raspored elemenata u retku, te za manje ekrane, mobitel, raspored elemenata u stupcu. Za poravnanje elemenata na većim ekranima, koriste se *justify-content*: *space-between* tako da su lijevi i desni element na suprotnim stranama trake; a *align-items: center* kako bi elementi bili centrirani vertikalno.

Koristi se jedan medijski upit, koji određuje veličinu elemenata do maksimalne širine 684 piksela, kako bi se definirali elementi za veličine mobitela i manjih tableta, kako bi raspored elemenata ostao jednak pri većim ekranima. Cilj je kod oba rasporeda elemenata bio da elementi uvijek budu imaju fiksnu veličinu, iz tog razloga se koristi fiksni *width* kod elemenata. Koriste se razne margine i ispunjavanja kako bi se dobio odgovarajući razmak između elemenata.

5.2.4. Proizvodi i knjiga

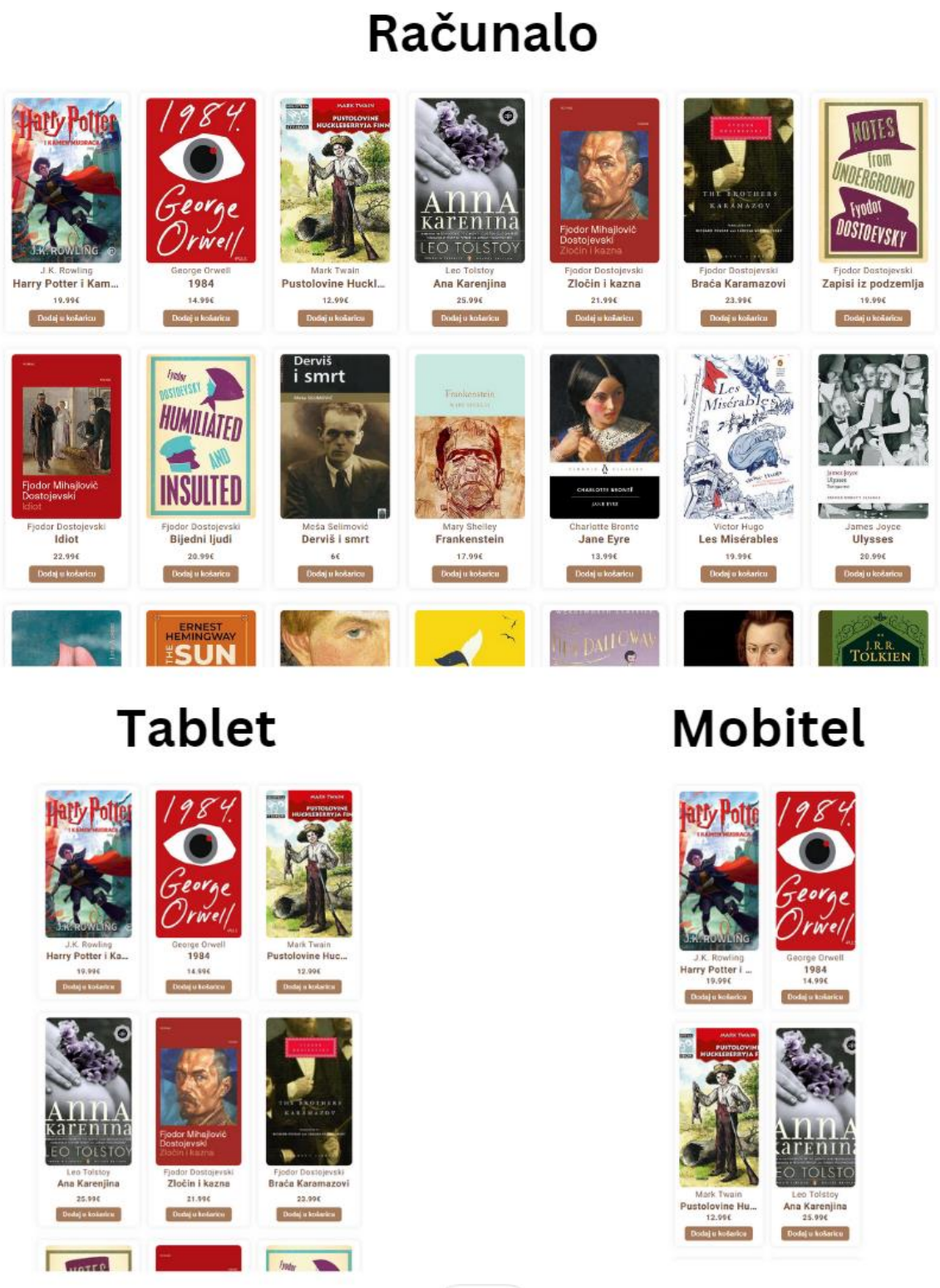
Proizvodi i knjige su dvije različite komponente. Komponenta proizvodi u sebi sadrži komponentu knjiga, ali uz to još ima elemente u koje je moguće unijeti filtre i sortiranje. Komponenta knjige povlači knjige iz baze podataka, te ih prikazuje. Kako su te dvije komponente zavisne jedna o drugoj, u ovom poglavlju ću ih opisati obje.

Komponenta knjiga koristi se na tri mjesta: na naslovnici, u svim proizvodima (kroz komponentu proizvodi). Komponentu knjige moguće je pozvati sa opcijom filtera i sortiranja, ili bez njih, te je moguće dodati parametar traži. U slučaju da se koristi parametar traži, na link web stranice dodaje se *?trazi=ime_knjige*, Kako bi se ti parametar traži dohvatio, koristi se kuka iz *react-router-dom useSearchParams* koji pronalazi putanje parametre uz upita. Uz to koriste se kuke *useState* i *useEffect* iz react biblioteke. *UseState* kuka čuva dohvaćene knjige, a *useEffect* kuka koristi se kada se komponenta prvi put učita i kada joj se promjene filteri, sort ili se napravi pretraga, bez potrebe za ponovnim učitavanjem stranice. Sljedeći kod prikazuje korištenje *useEffect* kuke za dohvat knjiga s poslužitelja:

```
useEffect(() => {
    const dohvatiKnjige = async () => {
      try {
        const upitParametri = new URLSearchParams();
          if(filteri !== null && filteri !== undefined) {
            if (filteri.autor) upitParametri.append('autor',
filteri.autor);
            if (filteri.zanr) upitParametri.append('zanr',
filteri.zanr);
            if (filteri.jezik) upitParametri.append('jezik',
filteri.jezik);
            if (filteri.minCijena) upitParametri.append('minCijena',
filteri.minCijena);
            if (filteri.maxCijena) upitParametri.append('maxCijena',
filteri.maxCijena);
          }

          const url = trazi === null
            ? "http://localhost:8080/knjiga/"
            : `http://localhost:8080/knjiga/trazi?trazi=${trazi}`;
          const zahtjev = (upitParametri.size > 0) ?
`${url}?${upitParametri.toString()}` : url;
          const odgovor = await axios.get(zahtjev);
          let knjige = odgovor.data;

          if (sort === 'naslov_az') {
            knjige = knjige.sort((a, b) =>
a.naslov.localeCompare(b.naslov));
          } else if (sort === 'naslov_za') {
            knjige = knjige.sort((a, b) =>
b.naslov.localeCompare(a.naslov));
          } else if (sort === 'cijena_nize') {
            knjige = knjige.sort((a, b) => a.cijena - b.cijena);
```

```
        } else if (sort === 'cijena_vise') {
          knjige = knjige.sort((a, b) => b.cijena - a.cijena);
        }
        postaviKnjige(knjige);
      } catch (error) {
        setError(error);
      } finally {
        setUcitavanje(false);
      }
    };

    dohvatiKnjige();
  }, [trazi, filteri, sort]);
```

Kako bi se napravio zahtjev za dohvatom knjiga, potrebno ga je ispravno formulirati. Iz tog razloga se prvo gleda jesu li postavljeni određeni filteri, kako bi se dodali u upitne parametre za zahtjev na poslužitelj. Uz to, gleda se postoji li dolazi li zahtjev za pretragu knjiga. Ako dolazi zahtjev za pretragu knjiga, on se izvršava, bez filtera. Kako bi se zahtjev izvršio, koristi se biblioteka axios. Kako bi napravili GET HTTP zahtjev, potrebno je napisati *axios.get(putanja),* pri tome moramo paziti da je putanja ispravna. Nakon što se zahtjev gotov, dobivamo knjige, te ako postoji sortiranje, one se sortiraju prema određenom sortu, te se na kraju postavljaju u niz knjige. Ako se desi pogreška, ona se postavlja, te se na kraju učitavanje postavlja kao gotovo.

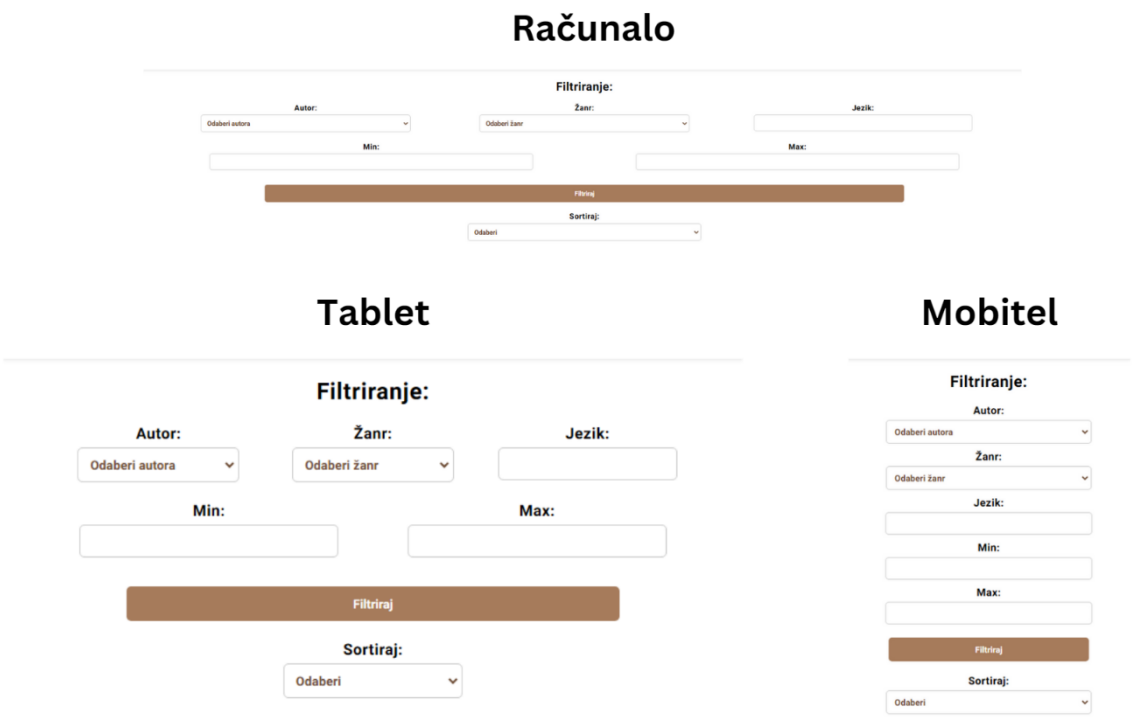


Slika 17: prikaz liste knjiga

U komponenti knjiga, responzivan dizajn je postignut na tako što su knjige raspoređene u mrežu koristeći *display: grid*. Svaka knjiga je poseban element unutar te mreže, te se ti elementi ponavljaju onoliko puta koliko je knjiga. Korišten je CSS property *grid-template-*

columns: repeat(auto-fit, minmax(200px, 1fr));, koji definira kako će stupci unutar rešetke biti raspoređeni. *Repeat* označava da želimo da se stupci ponavljaju, a *auto-fit* automatski određuje koliko stupaca treba stati u red. *Minmax* definira minimalnu i maksimalnu veličinu stupca, a ovdje je to postavljeno na minimalno 200px i maksimalno 1fr, koji označava jedan dio prostora, gdje se dostupni prostor dijeli među stupcima prema ovom ovom omjeru. Ako kontejner ima više prostora, stupci će se proširiti kako bi ga popunili. Razmak između knjiga je postavljen fiksno, koristeći *grid-gap* od 20 piksela. Na mrežu je postavljena maksimalna dužina od 90%, pa kako god se ekran smanjuje, knjige koje neće stat će otići ispod. Kroz smanjivanje ekrana, knjige će otići u nove retke, pa tako kod malih veličina ekrana, mogu biti prikazane dvije knjige u jednom retku, dok na većim ekranima može biti 8 i više, te je ovdje postignut responzivan dizajn. Kod manjih rezolucija zaslona, minimalni dio minmax je smanjen na 150 piksela, kako bi bolje izgledalo kod manjih rezolucija.

Komponenta za sve proizvode sadrži filtere i sortiranja, koja se izvršavaju nad knjigama. Koristi već objašnjeni *useEffect* i *axios* kako bi dohvatila sve autore i žanrove iz baze, koji se postavljaju na padajuće izbornike, pa korisnik može filtrirati prema postojećim autorima i žanrovima iz baze. Kako bi filtrirao, mora pritisnuti na gumb, jer se za svaki filter šalje poziv na poslužitelja koji dohvaća filtrirane knjige, dok je za sortiranje dovoljno izabrati koji tip sortiranja želi, jer se ono izvršava na klijentu.



Slika 18: filteri

Za mobilni prikaz filtera korišten je već spomenuti *flex-box*, koristeći raspored prema elemenata prema stupcu. Za prikaz na računalo ili tablet, iako izgleda da je korištena mreža, zapravo je također korišten *flex-box* u kombinaciji sa *flex-basis* svojstvom. *Flex-basis* definira početnu veličinu elementa unutar flex-box kontejnera prije nego što se uzme u obzir slobodan prostor. U ovom slučaju, koristi se da se postavi početna širina elemenata unutar kontejnera za filtriranja. Kontejner koji sadrži filtere ima postavljen flex-basis na 100%. Na svaki element je onda moguće staviti koliki postotak želimo staviti da on zauzme prostora u tom flex-boxu. Svaki element u prvom retku zauzima 32%, što je sveukupno 96%, i ostatak je za razmake. Kako znam da imam 5 filter elemenata, moguće je koristiti CSS svojstvo *:nth-child(broj_djeteta)*, i u njega definirati 4 i 5 element, za filtere cijene, te će za njih vrijediti druga pravila. Na njih je postavljen *flex-basis* 48%, što će ih postaviti u novi red ispod prva tri filtra. Elementi su centrirani, te se margine i elementi za unos povećavaju ili smanjivanju prema širina ekrana, čime se postigao responzivan dizajn.

5.2.5. Detalji o knjizi

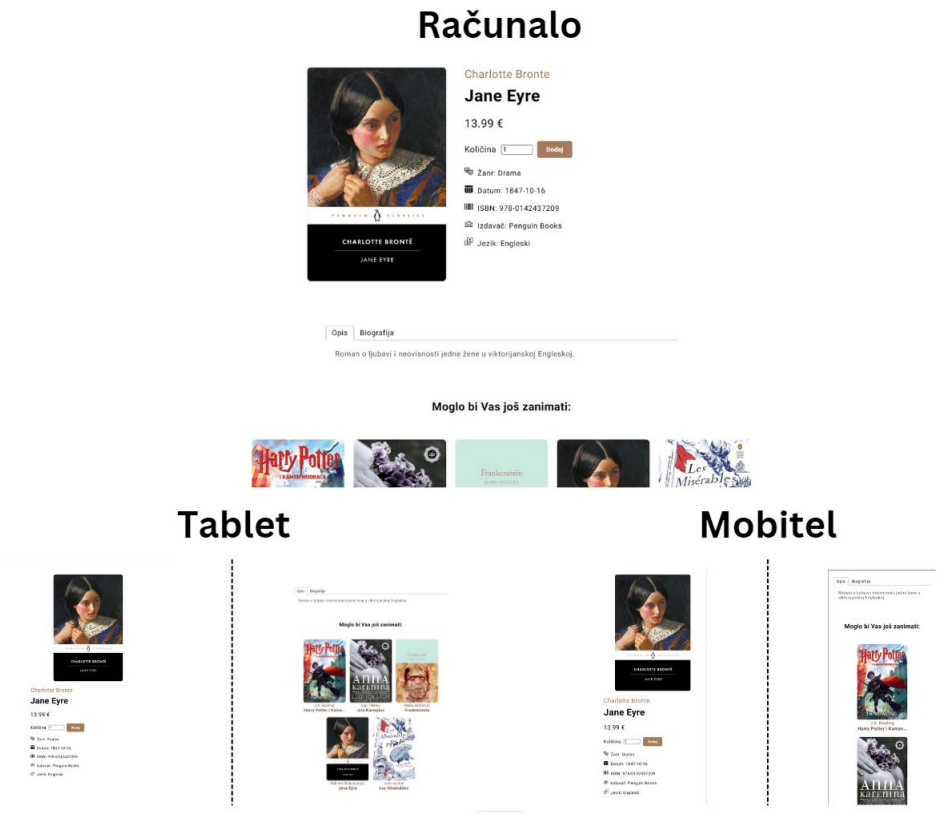
Kada iz komponente knjiga pritisnemo na neku knjigu, otvorit će nam se detalji o toj knjizi. Za to je zaslužna komponenta *KnjigaDetalji*. Komponenti se šalje id knjige, te će ona proslijediti HTTP zahtjev poslužitelju, kako bi dohvatila podatke o knjizi. Uz to šalje se i dodatan HTTP zahtjev koji će dohvatiti knjige koje su slične ovoj knjizi. Detalji o knjizi proširuju informacije koje se na ostalim komponentama vide, poput žanra, jezika, izdavača; a uz to prikazuje i opis knjige te biografiju autora.

Jedna novina u ovoj komponenti je korištenje biblioteke *react-tabs*, koja dodaje kartice (eng. Tabs) koje omogućuju korisniku da pregleda različite sekcije unutar stranice, tako da pritisne na nju. Kako su te sekcije na istom mjestu, ali se na klik izmjenjuju, to isto pomaže u kreiranju responzivnog dizajna. Tabs je zapravo komponenta, koja se sastoji od 4 dijela: *Tabs, TabList, Tab, TabPanel.* Tabs je glavni kontejner koji sadrži strukturu kartica. *TabList* sadrži liste kartice, tj. naslove, pomoću koje možemo izmijenit sekciju. *Tab* predstavlja pojedinačnu karticu unutar *TabList*, a *TabPanel*¸ predstavlja sadržaj koji je povezan s odabranom karticom. Struktura koda za sekcije je sljedeća:

```
<Tabs className="knjiga-opis-kontejner">
     <TabList>
            <Tab>Opis</Tab>
            <Tab>Biografija</Tab>
     </TabList>
     <TabPanel>
            <p className="knjiga-opis-paragraph">{knjiga.opis}</p>
    </TabPanel>
```

```
    <TabPanel>
            <p className="knjiga-opis-paragraph">
              {knjiga.autor.biografija}
            </p>
    </TabPanel>
</Tabs>
```

Kako bi se postignula responzivnost, urađene su već slične tehnike kao i ranije. Za uže ekrane, napravljen je *flex-box* s elementima u stupcu. Kod većeg ekrana, slika knjige i njezini detalji su postavljeni na *flex-box* i podijeljeni na lijevu i desnu stranu. Elementi su centrirani u oba slučaja, pa kako se ekran smanjuje, tako će elementi uvijek biti u centru.



Slika 19: detalji knjige

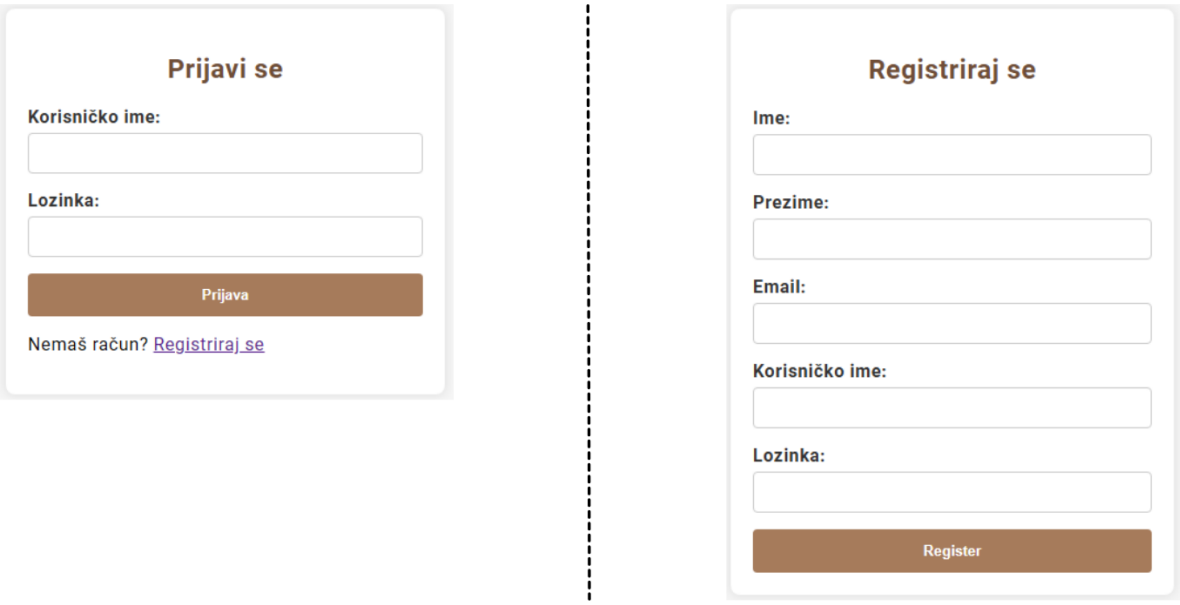
5.2.6. Prijava i registracija

Prijava i registracija su slične komponente, u kojima je cilj da se korisnik prijavu ukoliko ima račun, ili da se registrira ukoliko nema. Prijava nakon upisa podataka šalje zahtjev na backend za autentifikaciju, te ako ona uspješno prođe, navigira korisnika natrag na naslovnicu. Registracija radi na isti način, samo šalje podatke za registraciju na drugi backend poziv. Uz to, obje komponente koriste i kontekstnu komponentu *AuthContext,* kako bi spremile podatke

o korisniku dok se aplikacije koristi. Sljedeći kod prikazuje korisnikovu prijavu i korištenje konteksta:

```
const { prijava } = useContext(AuthContext) || {};
const handlePrijava = async (event) => {
        event.preventDefault();
        try {
            const response = await
axios.post('http://localhost:8080/korisnik/autentifikacija', {
                korIme: korIme,
                lozinka: password,
            });
            const korisnickoIme = response.data;
            prijava(korisnickoIme);
            if(korisnickoIme!== 'Korisnik nije pronađen' ||
korisnickoIme!== 'Neispravna lozinka') {
                navigate('/')
            }
        } catch (error) {
            postaviError('Nevažeća adresa e-pošte ili lozinka!');
        }
    };
```

Prvo se šalje HTTP POST zahtjev koristeći *axios*. Kao parametre je potrebno definirati adresu i JSON objekt, koji sadrži korisničko ime i lozinku, kao tijelo poruke. Ako je poslužitelj vratio korisničko ime, a ne pogrešku, izvršava se prijava koja poziva komponentu konteksta. Komponenta *AuthContext* koristi *createContext* iz *react* biblioteke, s kojim na početku kreira praznog korisnika. Kada se zove metoda prijava, ona postavlja korisnika u kontekstu, koji je moguće koristiti kroz cijelu aplikaciju, gdje god se *AuthContext* zove.

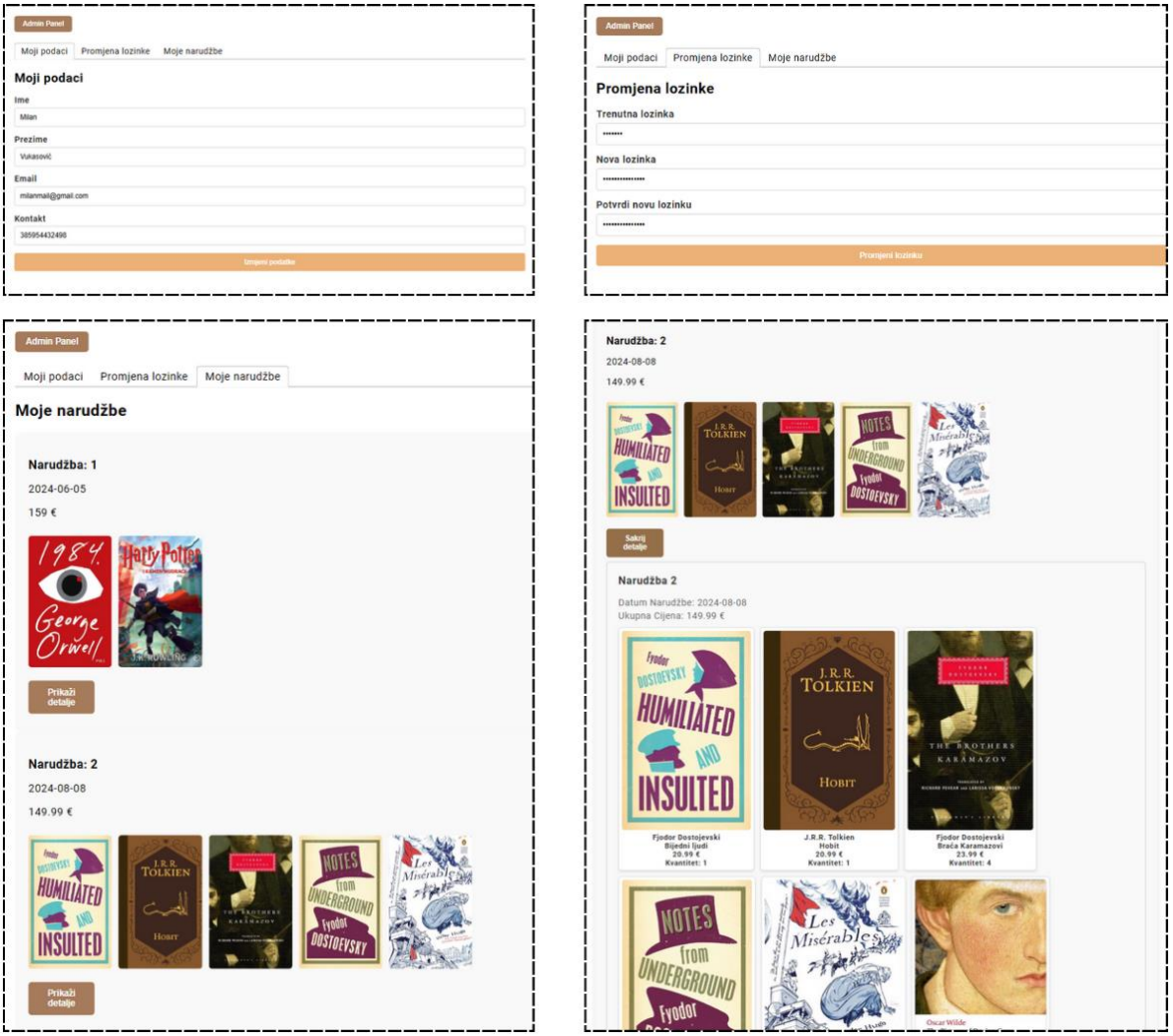


Slika 20: prijava i registracija

Responzivnost je ovdje postignuta na način da su kontejneri za prijavu i registraciju uvijek u sredini i iste veličine. Iza njih je postavljena nešto tamnija pozadina, s visinom od 75 vh. Vh označava mjernu jedinicu za visinu prozora za prikaz (eng. viewport height), što znači da će pozadina uvijek biti visoka 75% visine ekrana. Na manjim ekranima, elementi za unos podataka će se još smanjiti prema postotku.

5.2.7. Profil

Kada je korisnik prijavljen, on može pristupiti stranici sa svojim profilom. Na svom profilu ima tri kartice: kartica s osobnim podacima, kartica s promjenom lozinke i kartica s narudžbama. Te kartice su izrađene s već spomenutom bibliotekom *react-tabs*. Komponenta šalje zahtjev na poslužitelj za dohvaćanje podacima o profilu i o narudžbama. Uz to, koristi se *axios.put*, kako bi se napravio HTTP PUT poziv za ažuriranje podataka ili lozinke. Sljedeća slika prikazuje svaki od tri kartice od kojih je komponenta profil sačinjena.

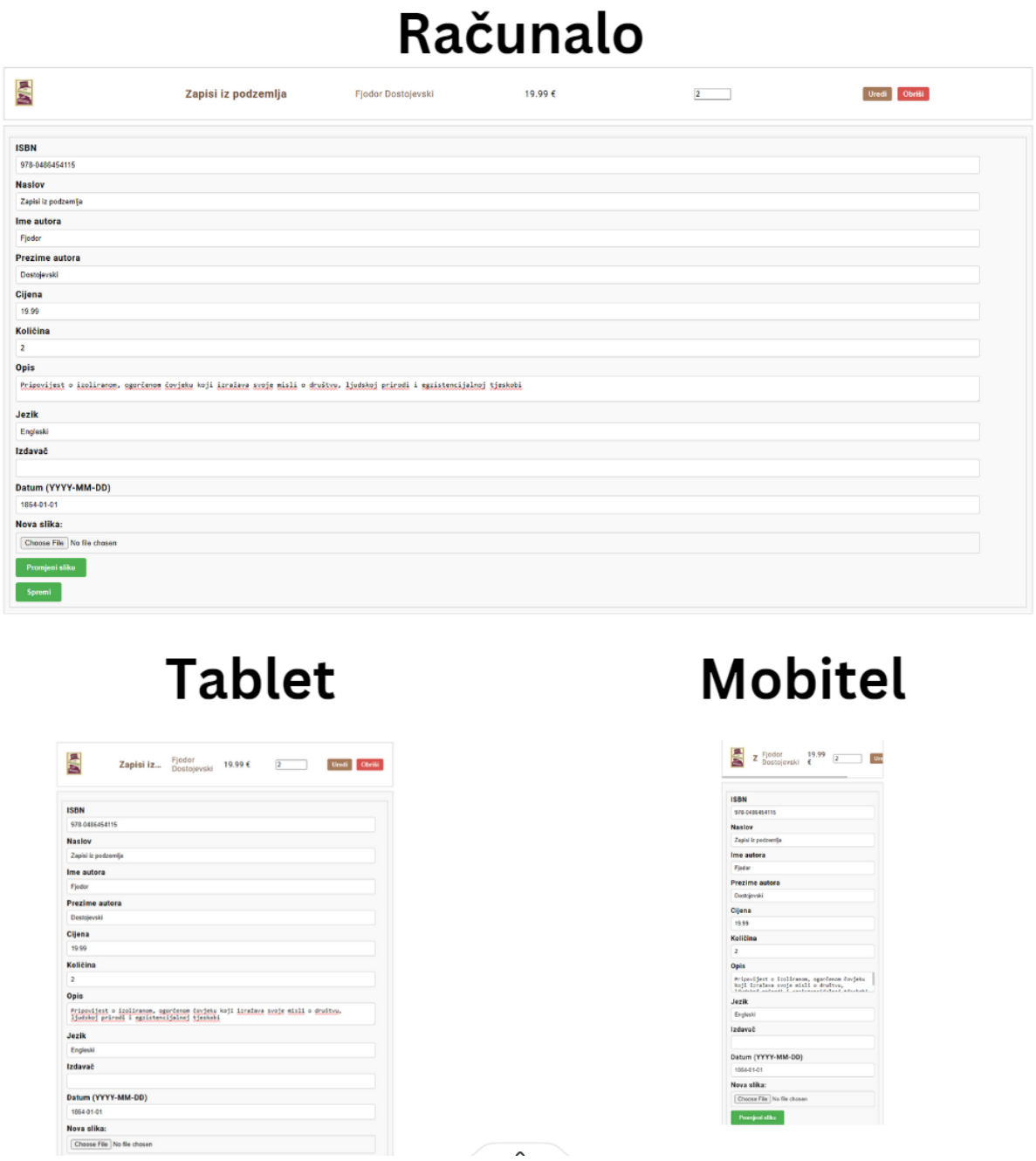


Slika 21: profil

Na kartici „Moje narudžbe“ korisnik može vidjeti sve svoje narudžbe. Svaka narudžba se sastoji od id-a, datuma i ukupne cijene. Ovdje sam postigao responzivnost, tako da se neovisno o ekranu uvijek prikazuje maksimalno pet knjiga. Ako ekran smanji dužinu, npr. na manje od 768 piksela, prikazivat će se još manji broj knjiga, u ovom slučaju 3 knjige. Ako korisnik želi, može ispod svake narudžbe pritisnuti „Prikaži detalje“ koji otvara novu komponentu „NarudzbaDetalji“ u kojoj je moguće vidjeti sve knjige, i detalje o njoj. Ta komponenta postiže responzivnost koristeći *display: block*. Kako će se elementi smanjivati, tako će se i elementi knjiga smanjivati i spuštati se u novi redak.

5.2.8. Admin panel

Kada korisnik dođe u svoj profil, u bazi se također provjerava je li korisnik admin. Ako je korisnik admin aplikacija mu prikazuje dodatan gumb koji vodi na „Admin panel“. Administrator aplikacije ovdje može dodavati nove knjige u bazu podataka i ažurirati postojeće. Administrator ovdje može vidjeti sve knjige koje su u bazi podataka. Klikom na uredi vidi i njezine sve podatke, te ih može uređivati. Sljedeća slika prikazuje uređivanje detalja knjige.



Slika 22: admin panel

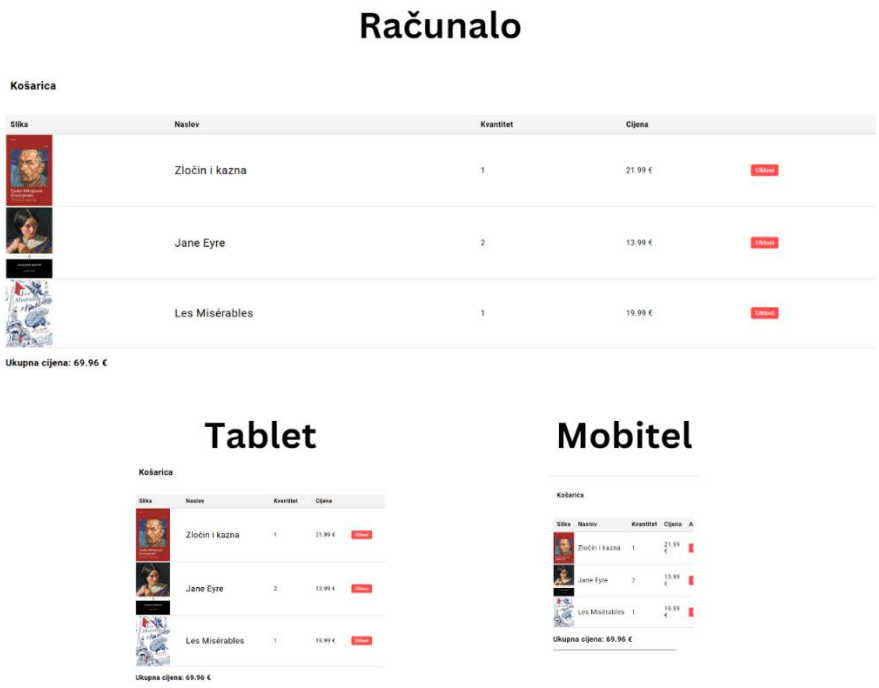
Svaka knjiga iz baze podataka je postavljena u poseban redak sa svojim osnovnim podacima, te su dodani gumbi za uređivanje i brisanje. Na manjim ekranima se događao problem gdje se svi osnovni podaci ne bi mogli prikazivati. Kako bi postigao responzivnost, za svaki redak knjige koristio sam svojstvo *overflow-x: scroll*, koje govori što se događa kada sadržaj u nekom kontejneru izlazi izvan granica. U ovom slučaju do ostalog sadržaja moguće je pomicati se (eng. Scroll) horizontalno, kako bi se vidjeli ostali podaci i gumbi za uređivanje i brisanje.

5.2.9.Košarica

Kada korisnik stavi neku od knjigu u košaricu, na navigacijskoj traci, na slici košarica dodat će se broj koji označuje broj knjiga u košarici. Klikom na ikonu košarice, korisnik dolazi na komponentu košarica. Komponenta košarica prikazuje osnovne podatke o svakoj knjizi koja se nalazi u košarici, kao i ukupnu cijenu svih knjiga. Svaku od knjiga moguće je ukloniti.

Svaki put dok se u aplikaciji kroz neku od komponenta doda knjiga, ona se upisuje u dodatnu komponentu „KosaricaKontekst“. „KosaricaKontekst“ radi na sličan princip kao i već spomenuti kontekst za prijavu korisnika. Kontekstu se prosljeđuje tip akcije: dodavanje ili brisanje iz košarice, i knjiga za koju je akcija potrebna.

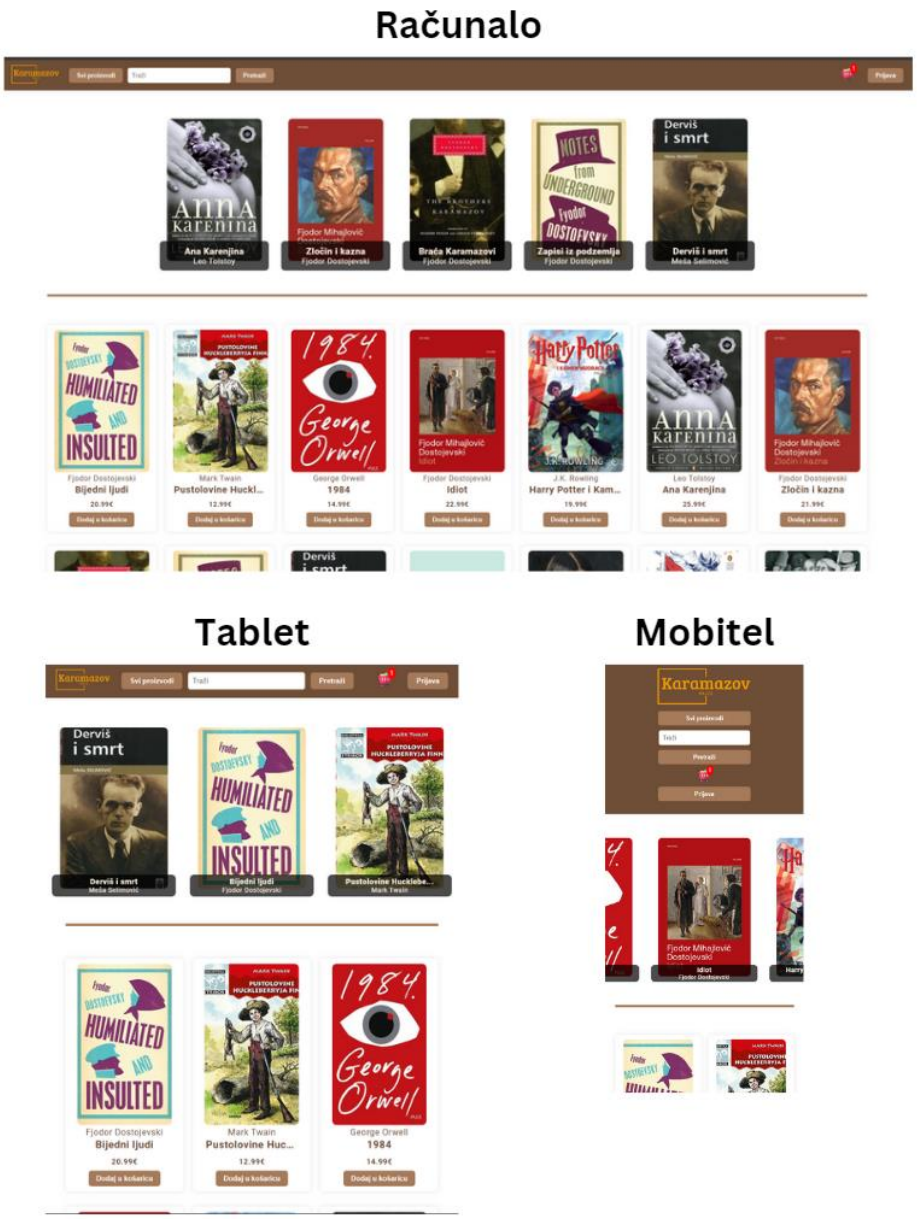
Responzivnost je u ovoj komponenti postignuta na sličan način kao i u admin panelu. Svaka knjiga postavljena je u poseban redak. Ako se ekran smanji, moguće se pomicati horizontalno kako bi se vidio ostatak podataka i obrisala knjiga iz košarice.



Slika 23: košarica

5.2.10. Naslovnica

Naslovnica je prva stranica koja se otvara kada se aplikacija upali. No, ona je ovdje opisana zadnja jer ne koristi neki novi koncept responzivnosti. Naslovnica se sastoji od dvije komponente, već opisanih knjiga i *react-responsive-carousel*. Na slici je prikazana naslovnica.



Slika 24: naslovnica

„React responsive carousel“ je biblioteka koja omogućuje jednostavno kreiranje responzivnih slajdova. Često se koristi kada se želi implementirati dinamičan prikaz sadržaja na web stranicama. Ova biblioteka je popularna zbog svoje fleksibilnosti i jednostavnosti. Uz nju se automatski postiže i responzivnost. Karusel se automatski prilagođava veličini prozora. Slajdovi se sami pomiču, ovisno o tome kako programer složi. Korisnik može sam mijenjati

slajdove, koristeći gumbiće, ili pomoću povlačenjem ekrana. U mom slučaju, ovdje se karusel koristi kako bi se prikazivalo nekoliko knjiga u krug. Nakon što se knjige povuku iz baze podataka, uzima se 10 knjiga, koje se prikazuju u krug. Korisnik može pritisnuti na knjigu što ga vodi na stranicu o detaljima knjige. Izabrao sam ovu komponentu kako bi pridonio estetici naslovne stranice.

6. Zaključak

U ovom diplomskom radu obrađena je izrada responzivne web aplikacije za e-trgovinu koristeći moderne tehnologije, React za web aplikaciju i Spring Boot za poslužiteljski dio aplikacije. Prvi dio rada fokusirao se na to definiciju e-trgovine i potrebu responzivnog dizajna za e-trgovinu. Objašnjen je i sami responzivan dizajn i njegovi koncepti poput medijskih upita, tehnologija za prilagodbu rasporeda i korištenja responzivnih slika i medija.

Nakon teorijskog dijela, u drugom dijelu rada opisan je proces planiranja aplikacije. Prvotno je objašnjena arhitektura sustava. Nakon toga detaljno je opisan način na koji REST servisi funkcioniraju, kako bi se postigla komunikacija između klijenta i poslužitelja. Opisana je osnovna teorija relacijskih baza podataka, te je detaljno opisana struktura korištene baze podataka. Nakon toga, opisane su ključne tehnologije u izradi web aplikacije: React, Spring Boot i PostgreSQL; te zašto sam izabrao baš njih. U završnom dijelu planiranja isplanirao sam i opisao kako će se sam korisnik moći kretati kroz web aplikaciju.

Treći dio rada sastoji se prikaza implementacije aplikacije. Prvo je prikazana izrada poslužiteljske aplikacije. U njoj su obrađene osnovne teme potrebne za izradu poslužitelja koristeći Spring Boot, tj. način na koji se kreira projekt U Spring Boot-u, što je model, repozitorij i kontroler, te kreiranje osnovnih CRUD operacija koje koriste REST servisi. Nakon toga opisana je detaljna izrada web aplikacije. Opisano je kako se kreira projekt u React-u, što su komponente, te je prikazana svaka komponenta u web aplikaciji i opisan način na koji se postignuta responzivnost.

Kroz izradu ove aplikacije, najviše fokusa je bilo na postizanju responzivnog dizajna. Izazov je bio u tome da aplikaciju bude moguće koristiti na ekranima bilo koje veličine. Često se dizajn morao promijeniti ili dodatno promisliti kako bi svaka komponenta mogla funkcionirati onako kako je zamišljeno. Kako sam moram prilagođavati ekran da vidim kako stranica izgleda na drugačijim ekranima, shvatio sam koliko je zapravo bitno da je stranica dostupna za svaki uređaj. npr. na manjim uređajima neki gumbi nisu bili dostupni odmah, što bi korisniku učinilo aplikaciju neiskoristivom. Uz to, aplikacija sada na svakom uređaju izgleda estetski lijepo, uz minimalna učitavanja, što bi kod korisnika moglo poboljšati ugođaj, te bi se mogli na stranicu češće vraćati kako bi kupovali, što je i cilj e-trgovine.

Popis literature

[1] A. Gupta, „E-Commerce: Role of E-Commerce in today's business“. [Na Internetu]. Dostupno na: https://www.ijccr.com/January2014/10.pdf [Pristupljeno: 15-svi-2024]

[2] Almeida, F. (2017). „The role of responsive design in web development.„ Na Internetu] Dostupno na: http://www.webology.org/2017/v14n2/a157.pdf [Pristupljeno: 15-svi-2024]

[3] J.V. Zande (2024). „Ecommerce: Electronic Commerce Definition, Benefits, Types, Examples“ [Na Internetu]. Dostupno na: https://www.forbes.com/sites/sap/2024/05/24/ecommerce-electronic-commerce-definition-benefits-types-examples/ [Pristupljeno: 11-lip-2024]

[4] Baturay, M. H., & Birtane, M. (2013). „Responsive Web Design: A New Type of Design for Web-based Instructional Content. *Procedia - Social and Behavioral Sciences“*, *106*, 2275–2279. [Na Internetu]. Dostupno na: https://doi.org/10.1016/j.sbspro.2013.12.259 [Pristupljeno: 15-svi-2024]

[5] „Responsive Web Design“ [Slika] (bez dat.) Dostupno: https://www.interaction-design.org/literature/article/responsive-design-let-the-device-do-the-work [Pristupljeno: 10-kol-2024]

[6] Microsoft (2023.) „Screen sizes and breakpoints“ [Na Internetu]. Dostupno na: https://learn.microsoft.com/en-us/windows/apps/design/layout/screen-sizes-and-breakpoints-for-responsive-design [Pristupljeno: 10-kol-2024]

[7] MDN (2023.) „Responsive design“ [Na Internetu]. Dostupno na: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design#responsive_layout_technologies [Pristupljeno: 10-kol-2024]

[8] IBM (2023.) „What is Three-Tier Architecture“ [Na Internetu]. Dostupno na: https://www.ibm.com/topics/three-tier-architecture [Pristupljeno: 10-kol-2024]

[9] Amazon Web Services (2024.) „What is RESTful API? - RESTful API Beginner’s Guide – AWS“ [Na Internetu]. Dostupno na: https://aws.amazon.com/what-is/restful-api/ [Pristupljeno: 10-kol-2024]

[10] Oracle (2022.), “What is a database?,” [Na Internetu]. Dostupno na: https://www.oracle.com/database/what-is-database/ [Pristupljeno: 11-kol-2024]

[11] Oracle (bez dat.), “Define a relationship ” [Na Internetu]. Dostupno na: https://docs.oracle.com/html/E79061_01/Content/Data%20model/Define_a_relationship.htm [Pristupljeno: 11-kol-2024]

[12] Meta Open Source (bez dat.), “React,” [Na Internetu]. Dostupno na: https://react.dev/ [Pristupljeno: 12-kol-2024]

[13] IBM (bez dat.) “What is Java Spring Boot? ,” [Na Internetu]. Dostupno na: https://www.ibm.com/topics/java-spring-boot [Pristupljeno: 12-kol-2024]

[14] Microsoft Azure (bez dat.) “What is Java Spring Boot?—Intro to Spring Boot ,” [Na Internetu]. Dostupno na: https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-java-spring-boot [Pristupljeno: 12-kol-2024]

[15] The PostgreSQL Global Development Group, (bez dat.) “PostgreSQL: The world’s most advanced open source database,” [Na Internetu]. Dostupno na: https://www.postgresql.org/ [Pristupljeno: 12-kol-2024]

[16] I. Jacob, (2024.) “Java Persistence API (JPA) For Database Access,” [Na Internetu]. Dostupno na: https://www.turing.com/kb/jpa-for-database-access [Pristupljeno: 14-kol-2024]

[17] S. Mhatre, (stu. 2023.) “Understanding Controllers in Spring Boot with Examples,” ,” [Na Internetu]. Dostupno na: https://saurabhnativeblog.medium.com/understanding-controllers-in-spring-boot-with-examples-312cb1879ec1 [Pristupljeno: 18-kol-2024]

Popis slika

Popis tablica