Razvoj web aplikacije primjenom okvira React i Next.js sa sustavom za upravljanje bazama podataka MySQL

Dumić, Antonio

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:112011

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Antonio Dumić

RAZVOJ WEB APLIKACIJE PRIMJENOM OKVIRA REACT I NEXT.JS SA SUSTAVOM ZA UPRAVLJANJE BAZAMA PODATAKA MYSQL

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Dumić

Matični broj: 0016156157

Studij: Informacijske tehnologije i digitalizacija poslovanja

RAZVOJ WEB APLIKACIJE PRIMJENOM OKVIRA REACT I NEXT.JS SA SUSTAVOM ZA UPRAVLJANJE BAZAMA PODATAKA MYSQL

ZAVRŠNI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, rujan 2024.

Antonio Dumić

Izjava o izvornosti

Izjavljujem da je moj završni 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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada bavi se razvojem web aplikacije korištenjem React biblioteke, Next.js okvira i MySQL sustava za upravljanje bazama podataka. Cilj rada je demonstrirati proces izgradnje moderne, skalabilne i responzivne web aplikacije korištenjem navedenih tehnologija. U teorijskom dijelu rada, detaljno su opisane korištene tehnologije: React kao biblioteka za izgradnju korisničkih sučelja, Next.js kao kompletan okvir za razvoj poslužiteljske strane, te MySQL kao relacijska baza podataka. Rad istražuje glavne aspekte razvoja klijentske i poslužiteljske strane aplikacije, kao i integraciju baze podataka. Kroz primjer razvijene web aplikacije, prikazane su neke od glavnih funkcionalnosti web aplikacije na temu pregleda ponuda videoigara na raznim internetskim platformama korištenjem navedenih tehnologija. Zaključak rada ističe važnost korištenja modernih tehnologija kao što su React i Next.js za razvoj web aplikacija koje znatno mogu unaprijediti i poboljšati razvoj.

Ključne riječi: Web aplikacija; React; Next.js; MySQL; JavaScript; Biblioteke; Razvojni okvir

Sadržaj

1. Uvod

Razvoj modernih web aplikacija važan je za današnje digitalno okruženje jer omogućuje pristup informacijama i uslugama s bilo kojeg mjesta i uređaja. Korištenje raznih tehnologija, kao što su biblioteke (eng. *libraries*), okviri (eng. *frameworks*) i full-stack tehnologije, omogućuju izradu skalabilnih, responzivnih i interaktivnih aplikacija. Integracija klijentskih i poslužiteljskih tehnologija unutar full-stack okruženja omogućuju učinkovit razvoj aplikacija koje mogu brzo reagirati na potrebe korisnika, osiguravajući bolje korisničko iskustvo (eng. *user experience - UX*) i veću funkcionalnost.

Tehnologije poput Reacta i Next.js omogućuju stvaranje interaktivnih i visoko učinkovitih korisničkih sučelja (eng. *user interface - UI*). React, kao biblioteka za izgradnju korisničkih sučelja, pruža komponentni pristup koji omogućuje ponovnu upotrebu koda i bržu izradu kompleksnih aplikacija. Next.js, kao full-stack okvir, nudi renderiranje na poslužiteljskoj strani (eng. *Server-Side Rendering - SSR*), statičko generiranje stranica (eng. *Static Site Generation - SSG*) i optimizaciju performansi, čime poboljšava korisničko iskustvo i SEO (eng. *Search Engine Optimization - SEO*). Upotreba ovih tehnologija omogućuje izradu skalabilnih i održivih aplikacija koje mogu odgovoriti na dinamične zahtjeve tržišta.

U ovom radu opisane su tehnologije React, Next.js i sustav za upravljanje bazom podataka MySQL u kontekstu izrade modernih web aplikacija. Prikazan je cjelokupni proces razvoja web aplikacije korištenjem navedenih tehnologija. Kroz rad neće biti opisane Web tehnologije HTML, CSS i JavaScript već se cjelokupni rad bazira isključivo na navedenim tehnologijama.

Ostatak rada strukturiran je tako da poglavlje 2 opisuje evoluciju web aplikacija te njihove karakteristike i značaj tehnologija u njihovom razvoju, poglavlje 3 pruža pregled tehnologija za razvoj web aplikacije na klijentskoj i poslužiteljskoj strani te pregled baza podataka, poglavlje 4 opisuje biblioteku React, poglavlje 5 opisuje okvir Next.js, poglavlje 6 opisuje sustav za upravljanje bazama podataka MySQL, dok poglavlje 7 prikazuje primjer razvijene web aplikacije u Reactu, Next.js-u i MySQL-u.

2. Web aplikacije

Prije nego se započne govoriti o web aplikacijama, važno je pojasniti razliku između Weba i interneta. Pojmovi Web (World Wide Web) i Internet često se koriste kao sinonimi iako oni to nisu. Internet je globalna mreža kroz koju se pristupa Webu ili World Wide Webu, koji je skup informacija. Drugim riječima, internet je temelj, a Web je njegova nadogradnja.

2.1. Evolucija web aplikacija

Web je prvi puta predstavljen kasnih 1980-ih od strane Tim Berners-Leea, direktora World Wide Web Consortiuma. Prema njegovoj perspektivi, Web ima tri faze kroz koje su se razvijale i same web aplikacije. Blogovi [1] i [2] opisuju faze na sljedeći način:

- Web 1.0 (read-only Web) - Dopuštao je traženje informacija i njihovo čitanje, bez mogućnosti interakcije korisnika ili generiranja sadržaja.
- Web 2.0 (read-write Web) - Uvodi se mogućnost komunikacije s drugim korisnicima i dodavanja sadržaja od strane korisnika.
- Web 3.0 (read-write-execute Web) - Omogućuje stvaranje strukturiranih podataka za bolje povezivanje i integraciju, potiče organiziranu suradnji i, što je najvažnije, povećava zadovoljstvo korisnika.

Danas možemo razlikovati dvije osnovne vrste web stranica koje čine web aplikacije. One su statičke i dinamičke web stranice. Termini se odnose na način na koji se web stranica ponaša i koliko je korisnicima omogućena interakcija s njom te koliko se brzo mijenja ponašanje i izgled web stranice.

Ako je glavni cilj web stranice prikazati podatke u obliku teksta ili grafike, te je primarna uloga korisnika samo pregled podataka, a ne interakcija s njima, tada je riječ o statičnoj stranici. Obično su napravljene u HTML-u s minimalnom funkcionalnosti korištenjem jezika JavaScript. Vjerojatno će se isti sadržaj prikazivati korisniku svaki put kada pokuša pristupiti sadržaju. Svaki put kada klijent zatraži nešto, poslužitelj obično daje isti odgovor i sadržaj se mijenja samo kada se promjene naprave na strani poslužitelja.

Web stranice koje su dinamične omogućuju korisnicima brojne načine interakcije i mogu mijenjati prikazane podatke ili kontinuirano ažurirati aplikaciju. Takve stranice će često imati animacije, promjenu prikaza grafika, gumbe, klizače i opcije za korisnike da unose izmjene, ali i općenito složeniju funkcionalnost od statične stranice. Promjene u sadržaju mogu se redovito događati putem skripti koje se izvršavaju na serveru.

2.2. Karakteristike web aplikacija

Web aplikacije su programi koje se pokreću na web pregledniku i omogućuju korisnicima da interaktivno koriste aplikaciju preko interneta. Web aplikacije se ne instaliraju na lokalna računala kao tradicionalne aplikacije, već im se pristupa putem Uniform Resource Locatora (URL) bez potrebe za instalacijom.

Mogu biti analizirane iz različitih perspektiva. Svaka dimenzija ima svoje karakteristike koje razlikuju web aplikacije od tradicionalnih aplikacija.

Knjiga [3] navodi da ISO/IEC 9126-1 standard definira sljedeće dimenzije: "proizvod (eng. *product*)", "korištenje (eng. *usage*)", "razvoj (eng. *development*)" s aspektom "unapređenje (eng. *evolution*)" kao sveobuhvatnom dimenzijom. Svaka dimenzija ima svoje karakteristike koje je čine jedinstvenom unutar svoje dimenzije.

Karakteristike povezane s dimenzijom proizvoda čine osnovne elemente web aplikacije, koji se sastoje od sljedećih elemenata [3]:

- Sadržaj (eng. *content*) - kreiranje, održavanje, spajanje i ažuriranje sadržaja su važni za korisnost web aplikacija.
- Navigacija (eng. *hypertext*) - nelinearna priroda hipertekstualnih dokumenata omogućuje organizaciju i prikazivanje informacija putem čvorova (eng. *nodes*), veza (eng. *links*) i sidra (eng. *anchors*).
- Prezentacija (eng. *presentation*) - estetika i intuitivnost korisničkog sučelja su presudni za izgled i dojam web aplikacija.

U skladu s objektno orijentiranom paradigmom, svaki od tih elemenata posjeduje kako statički tako i dinamički aspekt.

Korištenje web aplikacija zahtijeva neprestano prilagođavanje posebnim situacijama korištenja, poznatim kao konteksti. Prilagođavanje tim situacijama može biti podjednako važno za sve elemente web aplikacije, kao što su sadržaj, navigacija i prezentacija. Radi važnosti prilagođavanja kontekstu, karakteristike povezane s dimenzijom korištenja podijeljene su u tri kategorije [3]:

- Društveni (eng. *social*) - društveni kontekst odnosi se na aspekte specifične za korisnika, uključujući multikulturalnost i spontano korištenje.
- Tehnički (eng. *technical*) - tehnički kontekst uključuje kvalitetu mrežne veze te hardverske i softverske uređaje korištene za pristup web aplikaciji.
- Prirodni (eng. *natural*) kontekst - prirodni kontekst obuhvaća lokaciju i vrijeme pristupa, važnu za internacionalizaciju i dostupnost web aplikacija.

Elementi koji karakteriziraju dimenziju razvoja uključuju [3]:

- Razvojni tim (eng. *development team*) i tehnička infrastruktura (eng. *technical infrastructure*) - multidisciplinarni i mladi razvojni timovi, te nejednakost i nedostatak zrelosti korištenih komponenti utječu na razvoj web aplikacija.
- Proces razvoja (eng. *development process*) - obuhvaća sve karakteristike razvoja, naglašavajući fleksibilnost i paralelizam.
- Integraciju (eng. *integration*) postojećih rješenja - mnoge web aplikacije zahtijevaju tehničku, sadržajnu i organizacijsku integraciju s unutrašnjim i vanjskim sustavima.

Kako je već spomenuto, dimenzija unapređenja utiče na sve tri dimenzije proizvoda, korištenja i razvoja. Potreba za evolucijom može se opravdati [3]:

- Konstantnim promjenama zahtjeva i uvjeta - web aplikacije zahtijevaju kontinuirani razvoj zbog brzih promjena zahtjeva i uvjeta.
- Konkurencijskim pritiskom - velika konkurencija na internetu zahtijeva kratke životne cikluse proizvoda i brze razvojne cikluse.
- Općim ubrzanjem razvoja - brza evolucija Weba uzrokuje kratke životne vjekove web aplikacija i česta ažuriranja.

Prethodne karakteristike raspoređene su prema određenim dimenzijama. Blog [4] navodi neke općenitije karakteristike vezane uz web aplikacije:

- Pristupačnost - Web aplikacije su pristupačne na svim uređajima s web preglednikom, bez potrebe za preuzimanjem ili instalacijom.
- Neovisnost od platforme - Web aplikacije funkcionalne su na svim operacijskim sustavima, kao što su Windows, Mac ili Linux.
- Ažuriranja - Web aplikacije automatski se ažuriraju na serveru, uvijek pružajući najnoviju verziju bez potrebe za ručnim intervencijama.
- Surađivanje - Web aplikacije dopuštaju istovremeni rad više korisnika na projektima, potičući suradnju u realnom vremenu bez obzira na lokaciju korisnika.

2.3. Značaj tehnologija u razvoju web aplikacija

Prema knjizi [3] odabir pravih tehnologija je važan za uspješan razvoj web aplikacija. Da bismo bili sposobni koristiti tehnologije na smislen način, potrebno je razumijevati njihove karakteristike. Pored poznavanja relevantnih tehnologija, razvoj web aplikacija često zahtijeva razumijevanje kako različite tehnologije funkcioniraju zajedno unutar trenutne arhitekture.

Direktor i suosnivač kompanije Blurify [5] navodi da je prilikom odabira pravih tehnologija velika važnost u tome što klijenti žele, kakav je posao kojim se bave i kakve funkcionalnosti

traže u aplikaciji. Najbitnije prilikom odabira tehnologije za izradu web aplikacija je njena jednostavnost korištenja i sigurnost u stvaranju visokokvalitetne i korisne web aplikacije.

Optimizirane tehnologije mogu znatno unaprijediti performanse aplikacije, što rezultira bržim učitavanjem stranica i poboljšanim korisničkim iskustvom. Izbor tehnologija sa snažnim sigurnosnim karakteristikama može osigurati zaštitu aplikacije od mogućih napada. Zaštita podataka korisnika oslanja se na važne sigurnosne značajke poput enkripcije, autentifikacije i autorizacije. Efikasne tehnologije mogu umanjiti sveukupne troškove razvoja i održavanja aplikacije.

Prema [5] izbor neadekvatne tehnologije može dugoročno štetno utjecati na cjelokupni projekt, uključujući kvalitetu web aplikacije i financijske zahtjeve za njezin razvoj. Tijekom procesa razvijanja web aplikacije, postoji rizik od susreta s različitim problemima i neočekivanim situacijama koje kasnije mogu prouzročiti ozbiljne posljedice.

Razvoj tehnologije je brz, pa trenutno popularne tehnologije brzo postaju zastarjele. Izbor tehnologija koje su redovno ažurirane i podržane od strane zajednice ili kompanije može umanjiti opasnost od zastarjelosti. Nove i složene tehnologije mogu zahtijevati dugotrajan proces učenja, što može usporiti napredak.

Za olakšanje razvoja aplikacije, poboljšanje performansi, omogućavanje skalabilnosti i održavanja te općenito utjecanje na uspjeh projekta, koriste se različiti paketi tehnologija (eng. *technology stack*). Paketi tehnologija obuhvaćaju grupu alata, okvira i tehnologija koje se primjenjuju za razvoj web aplikacija. Tablica 1 prikazuje neke od paketa zajedno s tehnologijama koje koriste.

Tablica 1: Prikaz tehnologija korištenih u paketima tehnologija

MERN	MEAN	MEVN	LAMP	PERN
MongoDB	MongoDB	MongoDB	Linux	PostgreSQL
Express.js	Express.js	Express.js	Apache	Express.js
React	Angular	Vue.js	MySQL	React
Node.js	Node.js	Node.js	PHP	Node.js

(Izvor: Vlastita izrada)

3. Tehnologije i jezici za razvoj web aplikacije

Postoje mnoge tehnologije i programski jezici za razvoj web aplikacija. U ovom poglavlju bit će navedene i uspoređene neke tehnologije koje se koriste za razvoj web aplikacija. Specifičan fokus bit će stavljen na stranu klijenta (eng. *front-end*) i stranu poslužitelja (eng. *back-end*), kao i na baze podataka koje su važne za čuvanje i kontrolu podataka. Svako poglavlje će sadržavati analizu popularnih tehnologija, uključujući i njihove prednosti i nedostatke.

3.1. Pregled tehnologija za klijentsku stranu

Strana klijenta, poznatija kao front-end obuhvaća sve ono što je vidljivo krajnjem korisniku i s čime korisnik može imati interakciju preko svog preglednika. Glavni cilj je kreirati privlačno, jednostavno i interaktivno sučelje te omogućiti responzivnost sučelja na svim vrstama uređaja. Kod razvoja klijentske strane sve se svodi na tri glavna jezika:

- HTML (HyperText Markup Language) - nije programski jezik u tradicionalnom smislu, već skup instrukcija koje govore računalu kako prikazati sadržaj te služi za razvoj strukture same web stranice [6]
- CSS (Cascading Style Sheets) - jezik koji definira stilove poput fontova, boja i pozicija, te definira kako bi se informacije na web stranici trebale prikazivati i formatirati [7]
- JavaScript - interpreterski programski jezik koji služi za definiranje ponašanja web stranica, pretvara statične web stranice u dinamične [8]

Danas se web aplikacije gotovo nikada ne razvijaju korištenjem čistog HTML, CSS i JavaScript-a. Razvoj web aplikacija olakšava se i gotovo uvijek se koriste razne biblioteke ili okviri koji olakšavaju razvoj web aplikacije i omogućuju ponovno iskorištavanje dijelova web aplikacije.

Prema istraživanju provedenom na Stackoverflowu [9], React je najkorištenija biblioteka za razvoj klijentske strane i web aplikacija, prema odgovorima profesionalnih programera. Isto tako, React je najpopularnija prema Npm trendovima [10] i najtraženija prema Google trendovima [11]. Sve te informacije pokazuju da je React biblioteka vrlo popularna i često korištena prilikom razvoja klijentske strane web aplikacija.

Na temelju bloga [12] u nastavku su spomenute neke od prednosti i nedostataka korištenja okvira Angular, Vue.js i biblioteke React za razvoj klijentske strane.

Programeri se oslanjaju na Angular jer je podržan od strane Googlea, što garantira dugoročnu održivost aplikacija. Dokumentacija Angulara je detaljna i ilustrirana primjerima što je čini korisnom čak i za početnike. Suradnja unutar tima na Angular projektima je prilagodljiva jer male promjene ne zahtijevaju modifikacije cijele strukture, dok konzistentan i čitljiv kod povećava efikasnost. S druge strane, za učenje Angulara potrebno je poznavanje TypeScripta, što može biti izazovno za neke programere.

Prilikom razvoja React aplikacija, korištenje komponenti smanjuje kompleksnost i obujam koda zbog mogućnosti ponovnog korištenja komponenti. HTML dokumenti koriste objektni model dokumenata (eng. *Document Object Model (DOM)*) koji predstavlja prikaz podataka o objektima koji čine strukturu i sadržaj dokumenta na webu te omogućuje programerima stvaranje i izgradnju dokumenata, kretanje po njihovoj strukturi i dodavanje, mijenjanje ili brisanje elemenata i sadržaja. React koristi virtualni DOM umjesto klasičnog, što povećava učinkovitost i ubrzava funkcionalnost aplikacija. Iako je React najpopularnija JavaScript biblioteka i prilično jednostavna za naučiti, programeri moraju biti u toku s njegovim redovnim ažuriranjima. Nije potrebno imati posebne preduvjete za učenje Reacta, no JavaScript Syntax Extension (JSX) sintaksa može biti teška za neke programere.

Vue.js je stvoren s ciljem da bude prihvatljiv i lagan za korištenje, bez potrebe za prethodnim iskustvom. Prednosti koje donosi uključuju SEO, ubrzanje učitavanja web stranica i jednostavno učenje. Ipak, Vue.js ima uži ekosustav i manju podršku za zastarjele preglednike i operacijske sustave. Važnost podrške zajednice je neophodna za očuvanje i usvajanje tehnologije, dok je ekosustav važan za prilagodbu aplikacija različitim preglednicima i operativnim sustavima. Angular i React imaju snažnu podršku Googlea i Facebooka, što stvara povjerenje među korisnicima, dok Vue.js nije toliko široko prihvaćen među širom publikom.

3.2. Pregled tehnologija za poslužiteljsku stranu

Kada je riječ o poslužiteljskoj strani, poznatijoj kao backend, misli se na sve procese koji se odvijaju na serveru, uključujući poslovnu logiku, autentifikaciju, komunikaciju s bazama podataka i drugo. Nasuprot klijentskoj strani, poslužiteljska strana se sastoji od skrivenog dijela koji nije vidljiv krajnjem korisniku.

Za razvoj poslužiteljske strane također je potrebno korištenje određenog programskog jezika i okvira. Mogu se koristiti mnogi jezici, a neki od njih zajedno s pripadajućim okvirima su sljedeći:

- JavaScript (Node.js)
- C# (ASP.NET CORE)
- Python (Flask)
- PHP (Laravel)
- Java (Spring Boot)

Prema istraživanju provedenom na Stackoverflowu [9], Node.js je najkorišteniji za razvoj poslužiteljske strane te drugi najkorišteniji odmah iza biblioteke React za razvoj web aplikacija, prema odgovorima profesionalnih programera. prema Google trendovima [13] može se primijetiti kako je Node.js i dalje najtraženiji u odnosu na prethodno spomenute okvire.

Na temelju blogova [14] i [15] u nastavku su spomenute neke od prednosti i nedostataka okvira Node.js, Flask i Laravel za razvoj poslužiteljske strane.

Kada se usporede Node.js i PHP, može se primijetiti da Node.js brže izvršava kod zahvaljujući svom asinkronom, neblokirajućem pristupu vođenom događajima, što ga čini idealnim za višekorisničke aplikacije. I dok PHP koristi sinkrono programiranje, Node.js koristi samo JavaScript i za klijentsku i za poslužiteljsku stranu što olakšava proces razvoja. Ipak, strukturiranje koda je važno u asinkronom modelu Node.js-a, dok ugniježđene asinkrone funkcije otežavaju ispravljanje pogrešaka. Novim programerima mogu predstavljati problem kompleksni koncepti poput povratnih poziva (eng. *callback*) i asinkronih funkcija (eng. *asynchronous function*).

Flask je poznat po svojoj skalabilnosti i brzom razvoju web aplikacija. Svojom sposobnošću prilagodbe i podrškom za modularno programiranje, olakšava premještanje i testiranje modula, dok minimalna ovisnost omogućuje da kod i dalje besprijekorno funkcionira kod povećanja obujma projekta. Ipak, nedostatak integriranih alata zahtijeva od programera dodavanje biblioteka ručno, što može rezultirati sporijom aplikacijom. Isto tako, podrška različitim tehnologijama može rezultirati većim troškovima, dok korištenje zastarjele tehnologije može zahtijevati nova rješenja, što dovodi do povećanja troškova održavanja.

Laravel pruža brojne korisne opcije za brz i jednostavan razvoj web aplikacija, uključujući fleksibilan sustav usmjeravanja i jednostavnu sintaksu aplikacijskog programskog sučelja Application Programming Interface - API. Snažan ekosustav i dobro napravljena dokumentacija, uključujući seminare i obrazovne materijale, povećavaju interes kod programera. Dodatni ugrađeni mehanizmi olakšavaju razvoj kroz podršku obradi pogrešaka, provjeru autentičnosti i autorizaciju. Međutim, jednostavna struktura može uzrokovati zaboravljanje složenijih koncepata, dok nelogičan raspored direktorija i nedostatak kompatibilnosti između verzija također mogu biti problematični.

Isto tako, ta jednostavna sintaksa može predstavljati i nedostatak. Postoji rizik od prilagodbe na pojednostavljenu sintaksi i zaboravljanja kako su čisti zahtjevi i funkcije napisani. Također, nelogičan raspored direktorija i datoteka predstavlja jedan od nedostataka, te postoje povrede povratne kompatibilnosti između verzija okvira.

Razvijeni su i okviri za poslužiteljsku stranu bazirani na temelju okvira i biblioteka klijentske strane. Time se spaja razvoj na obje strane i tako olakšava razvoj web aplikacija. Izgrađeni okviri temeljeni na postojećim klijentskim okvirima i bibliotekama su sljedeći:

- Next.js - okvir baziran na biblioteci React
- NestJS - okvir baziran na okviru Angular
- Nuxt.js - okvir baziran na okviru Vue.js

3.3. Pregled baza podataka

Porast informacija daleko premašuje napredak u pronalaženju informacija. Zato je izbor odgovarajućeg modela baze podataka važan za razvoj sustava za pretraživanje informacija. Postoje različite kategorije baza podataka koje se koriste za skladištenje raznolikih tipova podataka.

Najčešće korištene su relacijske baze podataka (npr. MySQL) u kojima se koristi strukturalni jezik za upite (eng. *Structured Query Language (SQL)*) i NoSQL baze podataka (npr. MongoDB). Drugi tipovi baza podataka uključuju [16]:

- Centralizirane baze podataka (eng. *Centralized*)
- Distribuirane baze podataka (eng. *Distributed*)
- Objektno orijentirane baze podataka (eng. *Object-oriented*) i drugi.

Istraživanje provedeno od strane Deineko, Sotnik, Vovk i Lyashenko 2021. godine navodi neke od razlika između relacijskih i NoSQL baza podataka. [16]

Relacijske baze podataka se sastoje od više tablica koje su povezane međusobno i sadrže podatke o konkretnim objektima. Svaki redak tablice sadrži podatke o jednom objektu, dok su stupci karakteristike koje opisuju te objekte. Prednosti relacijskih baza podataka uključuju jednostavnost zahvaljujući jedinstvenoj strukturi informacija koja služi kao formalizacija tabličnog prikaza podataka, što je već poznato korisnicima. Kada je potrebno promijeniti strukturu relacijske baze podataka radi neovisnosti podataka, obično dolazi do malih promjena u aplikaciji. Relacijske baze podataka osiguravaju sigurnost pružanjem podrške za autentifikaciju.

Suprotno tome, postoje određeni nedostaci u upotrebi relacijskih baza podataka. Veza između podataka se uspostavlja sporo u relacijskim bazama podataka. Velika potrošnja memorije za prikazivanje baze podataka predstavlja jedan od većih nedostataka.

NoSQL baze podataka koriste se za čuvanje velikog broja skupova podataka, koji nisu samo pohranjeni u tablicama, već i na različite načine. Različite vrste pohrane uključuju ključ-vrijednost pohranu (eng. *key-value storage*), pohranu grafova (eng. *graph storage*), pohranu orijentiranu na dokumente (eng. *document-oriented*), i druge.

NoSQL baze podataka omogućuju brz i fleksibilan razvoj aplikacija te pružaju visoku skalabilnost. Također, dopuštaju vrlo efikasnu obradu nestrukturiranih podataka, često pri vrlo velikim brzinama. Dopuštaju obradu velikih količina podataka ili čuvanje jedne velike baze podataka u distribuiranim server klasterima servera. Osim toga, dopuštaju jednostavno prebacivanje baze podataka u oblak za postojeća NoSQL radna opterećenja.

Ograničenja u NoSQL baza podataka proizlaze iz ograničenja integriranog upitnog jezika (eng. *query language*), jer se većina upita i API metoda temelje na SQL funkcijama s manje funkcionalnosti. Pojavljuju se poteškoće prilikom brzog prebacivanja podataka iz jedne NoSQL baze u drugu. Velika mana je potreba za stvaranjem vlastitih alata za upravljanje bazom podataka. Većini poduzeća nedostaje dovoljno podataka i posebnih radnih uvjeta za korištenje NoSQL rješenja kao glavne baze podataka.

Prema istraživanju provedenom na Stackoverflowu [9], prema odgovorima profesionalnih programera, najkorištenija relacijska baza podataka je PostgreSQL, dok je na drugom mjestu MySQL, nakon koje slijedi SQLite. Od NoSQL baza podataka najkorištenija je MongoDB koja se nalazi na petom mjestu. Prema trenutnom rangiranju DB-Enginesa [17], Oracle je trenutno najpopularniji sustav za upravljanje bazom podataka (SUBP), s MySQL-om na dru-

gom mjestu i PostgreSQL-om na četvrtom. Ovime se jasno vidi koliko je MySQL popularna među programerima i uvijek se nalazi na vrhu.

Na temelju bloga [18] u nastavku su spomenute dodatne razlike između najkorištenijih relacijskih baza podataka, odnosno PostgreSQL, MySQL i SQLite baza podataka.

PostgreSQL se ističe po sposobnosti proširivanja i pridržavanja standarda, uključujući nasljeđivanje tablica i preopterećenje funkcija. Primjena Multiversion Concurrency Controla (MVCC) omogućuje obavljanje više zadataka istovremeno, što je karakteristika poznata kao konkurentnost (eng. *concurrency*), dok napredne SQL funkcionalnosti i veća usklađenost sa standardima privlače programere. Iako zahtijeva više truda za instalaciju i upravljanje te ima manje vanjskih alata, PostgreSQL se ističe zbog svoje jake zajednice i mogućnosti proširenja baza podataka.

Postavljanje i upravljanje su vrlo složeni u usporedbi s MySQL-om, što predstavlja veliku komplikaciju. Usporedbe s drugim sustavima za upravljanje bazom podataka pokazuju da je sporija za jednostavne operacije koje zahtijevaju puno čitanja te postoji manje alata trećih strana za pomoć u upravljanju bazom podataka.

MySQL baza podataka je osmišljena kako bi se omogućile brze i pouzdane funkcionalnosti, pridržavajući se u potpunosti standarda SQL-a. U suprotnosti s aplikacijama koje koriste SQLite, aplikacije koje koriste MySQL pristupaju bazi putem zasebnog procesa. S obzirom na to da se proces poslužitelja nalazi između baze podataka i ostalih aplikacija, pruža veću kontrolu nad odobravanjem pristupa bazi podataka.

Nasuprot tome, zbog svoje brzine i jednostavnosti korištenja, MySQL ima određena funkcionalna ograničenja. Posjeduje dvostruku licencu, uključujući besplatnu verziju otvorenog koda zajednice i više plaćenih komercijalnih izdanja koja se objavljuju pod vlasničkim licencama.

SQLite se ističe po svojoj prenosivosti, pouzdanosti i snažnim performansama u situacijama gdje je memorija ograničena. Omogućuje direktno čitanje i pisanje u datoteku baze podataka na disku bez potrebe konfiguriranja serverskog procesa, što pojednostavljuje proces instalacije i uporabe. Sve informacije se čuvaju u jednoj datoteci, koja zauzima prostor manji od 600 KB.

Iako omogućuje više procesa istovremeno, samo jedan proces može modificirati bazu podataka u isto vrijeme, što ograničava upotrebu za aplikacije s više korisnika. Nedostaci u sigurnosti i ograničene dozvole također otežavaju korištenje za zahtjevnije aplikacije.

4. React

React je JavaScript biblioteka razvijena od strane Facebooka 2011. godine koja služi za izradu korisničkih sučelja. Cilj razvoja Reacta je olakšanje i poboljšanje razvoja korisničkog sučelja. U pogledu web aplikacija i web stranica, korisničko sučelje predstavlja sve dijelove zaslona koji su prikazani korisnicima i s kojima korisnici posjeduju određene interakcije. U dijelove zaslona mogu spadati razni izbornici, navigacije, gumbi, trake za pretraživanje i slično.

4.1. Osnovni pojmovi i koncepti Reacta

4.1.1. React komponente

React aplikacije sastavljene su od takozvanih komponenata (eng. *components*) i osnovni su građevni blokovi. Komponente su dijelovi korisničkog sučelja koji posjeduju vlastitu logiku i dizajn te omogućuju ponovno korištenje. Komponente mogu biti različitih veličina, od manjeg gumba pa sve do čitave stranice. Konceptualno su kao JavaScript funkcije i vraćaju React elemente koji opisuju što bi se trebalo pojaviti na ekranu. Glavna komponenta unutar neke datoteke specificirana je ključnom riječju `export default`.

4.1.2. Virtualni DOM

Umjesto dvosmjernog povezivanja podataka, React nudi drugačije rješenje koje se zove virtualni DOM. Virtualni je DOM brža reprezentacija stvarnog DOM-a u memoriji i omogućuje reaktivno korištenje JavaScripta i DOM-a.

Prema autoru Fedosejevu [19], virtualni DOM funkcionira u tri koraka:

1. Kad god se promijeni stanje podatkovnog modela, virtualni DOM i React ponovno će renderirati korisničko sučelje u prikaz virtualnog DOM-a.
2. React zatim izračunava razliku između dva prikaza virtualnog DOM-a. Prethodni prikaz virtualnog DOM-a koji je izračunat prije promjene podataka i trenutni prikaz virtualnog DOM-a koji je izračunat nakon promjene podataka. Ova razlika između dva prikaza virtualna DOM-a je ono što se zapravo treba promijeniti u stvarnom DOM-u.
3. React ažurira samo ono što treba promijeniti u stvarnom DOM-u.

Pronalaženje razlike između dva prikaza virtualnog DOM-a i renderiranje samo ažuriranih dijelova u stvarnom DOM-u je brz proces. Glavna korist je manje brige oko potrebe za ponovnim renderiranjem. React omogućuje pisanje koda kao da se svaki put kad se promijeni stanje aplikacije, renderira cijeli DOM.

4.1.3. JSX

JSX se može promatrati kao transformacijski sloj koji pretvara XML sintaksu za kreiranje React komponenti u sintaksu koju React koristi za prikazivanje elemenata u JavaScriptu. Njegovo korištenje nije obavezno, ali je vrlo poželjno jer olakšava izradu samih aplikacija. Sintaksa može podržati obične HTML oznake, kao i prilagođene React klase.

Prevođenje oznaka u odgovarajuće React elemente obavlja se na način kao što je prikazano u sljedećem primjeru iz knjige [20]:

```
// JSX verzija

React.render(
    <div>
        <h1>Header</h1>
    </div>
);

// JSX verzija će biti prevedena u sljedeći kod

React.render(
    React.createElement('div', null,
        React.createElement('h1', null, 'Header')
    );
);
```

Iz prethodnog programskog koda može se primijetiti kako JSX verzija koristi JSX sintaksu, koja vizualno podsjeća na HTML. JSX čini programski kod čitljivijim i olakšava njegovo pisanje, naročito ako se radi o složenijim strukturama. JSX sam po sebi nije važeći JavaScript, već ga treba prevesti u ekvivalentni JavaScript korištenjem `React.createElement` funkcije, što se uglavnom obavlja s pomoću alata kao što je Babel.

Drugi dio programskog koda predstavlja ekvivalentnu JavaScript verziju prethodne JSX verzije. Koristi uobičajeni JavaScript i funkciju `React.createElement` za kreiranje React elemenata. Takav kod može se izvršavati direktno u JavaScript okruženjima bez potrebe za dodatnim kompilacijskim koracima. Svaka JSX oznaka prevodi se u poziv funkcije `React.createElement` koja ima nekoliko parametara. Prvi parametar odnosi se na tip elementa (npr. `div` ili `h1`), drugi parametar je objekt svojstava (eng. *props*) (koji je u ovom slučaju `null`), dok se treći i svi sljedeći parametri odnose na djecu tog elementa.

4.1.4. React vezne točke

Vezne točke (eng. *hooks*) dozvoljavaju korištenje raznovrsnih React karakteristika unutar komponenata. React nudi mogućnost njihovog samostalnog korištenja te mogućnost izrade vlastitih kombinirajući ugrađene.

Prema službenoj React dokumentaciji [21], vezne se točke mogu podijeliti u sljedeće kategorije:

- Vezne točke stanja (eng. *State Hooks*) - Omogućuju komponentama da zapamte informacije poput korisničkih unosa (npr. `useState`, `useReducer`).
- Vezne točke konteksta (eng. *Context Hooks*) - Omogućuju komponentama primanje informacija od udaljenih roditelja bez prosljeđivanja kao svojstva. (npr. `useContext`).
- Vezne točke referenci (eng. *Ref Hooks*) - Dopuštaju komponentama da sadrže neke informacije koje se ne koriste za renderiranje, poput DOM čvora ili ID-a vremenskog ograničenja. Za razliku od kuka stanja, ažuriranje reference ne renderira komponentu (npr. `useRef`, `useImperativeHandle`).
- Vezne točke efekata (eng. *Effect Hooks*) - Omogućuju komponentama povezivanje i sinkronizaciju s vanjskim sustavima (eng. `useEffect`).
- Vezne točke izvedbe (eng. *Performance Hooks*) - Omogućuju optimizaciju performansi ponovnog renderiranja preskakanjem nepotrebnog rada (npr. `useMemo`, `useCallback`).
- Ostale vezne točke - Nisu često korištene, nego su uglavnom korisne autorima biblioteka (npr. `useDebugValue`, `useId`, `useSyncExternalStore`, `useActionState`).
- Vlastite vezne točke - Mogućnost definiranja vlastitih veznih točaka prema potrebama web aplikacije

4.1.5. Svojstva React komponenata

Svojstva komponenata su uglavnom samo za čitanje. Koristeći svojstva, moguće je prenositi podatke između komponenti na isti način kao što se prenose argumenti u funkciji. Najčešće im se pristupa korištenjem `this.props` koji je običan JavaScript objekt u Reactu. Autor Gackenheimer [20] tvrdi da svojstva ne bi trebalo tretirati kao da nisu nepromjenjiva jer će ostati ista tijekom životnog ciklusa komponente. Ako je potrebno izvršiti promjenu na komponenti, moguće je promijeniti njezino stanje korištenjem objekta stanja.

4.1.6. Stanja React komponenata

Stanje se postavlja na svaku komponentu prilikom inicijalizacije i koristi se za zapamćivanje informacija o komponenti. Nije dopušteno pristupiti mu izvan komponente, osim ako nadređena komponenta ne dodaje ili postavlja početno stanje komponente. Stanje se mijenja tijekom životnog ciklusa komponente. Promjena stanja može se dogoditi kao odgovor na korisničku radnju ili događaje koje generira sustav i te promjene utječu na ponašanje komponente i način na koji će biti prikazana.

Prema Gackenheimeru [20], trebalo bi stvarati komponente sa što manje objekata stanja kako bi se izbjeglo povećanje složenosti komponenti s dodavanjem stanja, budući da React komponente ostaju nepromijenjene tijekom vremena ovisno o stanju. Ako je moguće izbjeći, prihvatljivo je uopće nemati nikakvo stanje u komponenti.

4.2. Instalacija Reacta

Kada se radi o razvoju nove web aplikacije ili web stranice s Reactom, preporuka je React dokumentacije [22] korištenje nekog od okvira koji podržava React. Naravno, postoji mogućnost korištenja React biblioteke bez okvira ili naknadno dodavanje okvira, no prednost korištenja okvira od početka je izbjegavanje potrebe za izgradnjom vlastitog okvira kasnije u razvoju web aplikacija ili web stranica.

Okviri preporučeni od strane Reacta su sljedeći [22]:

- Next.js - Instalacija s pomoću `npx create-next-app@latest` u terminalu.
- Remix - Instalacija s pomoću `npx create-remix` u terminalu.
- Gatsby - Instalacija s pomoću `npx create-gatsby` u terminalu.
- Expo (za nativne aplikacije) - Instalacija s pomoću `npx create-expo-app` u terminalu.

Instalacija Reacta bez okvira izvršava se putem terminala sljedećom naredbom `npx create-react-app <naziv aplikacije>`. Instalacija Reacta bez okvira omogućuje samo razvoj korisničkog sučelja, bez mogućnosti razvoja pozadinske logike ili baze podataka. U pozadini se koristi prethodno spomenut Babel alat te webpack alat koji skuplja statičke module JavaScript aplikacija te kombinira svaki modul u jedan ili više paketa, koji su statička sredstva za posluživanje sadržaja.

Tijekom postupka instalacije Reacta na navedeni način, automatski se inicijalizira GitHub repozitorij i instlira se predložak ovisnosti korištenjem `npm` alata. Također se instaliraju neophodni paketi za razvoj React aplikacija, koji obuhvaćaju sljedeće pakete:

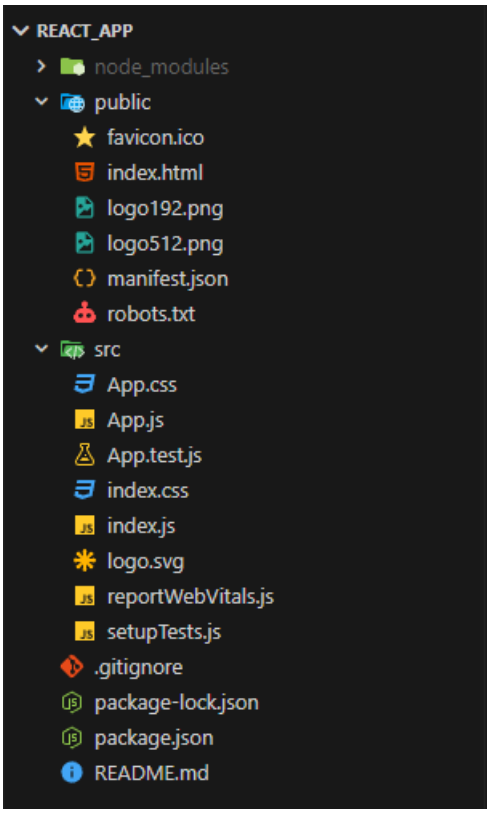
- react - sadrži samo funkcionalnosti potrebne za definiranje React komponenata
- react-dom - služi kao ulazna točka za DOM i poslužiteljske rendere za React
- react-scripts - uključuje skripte i konfiguraciju koju koriste React aplikacije

Nakon uspješno kreirane React aplikacije, moguće je pokrenuti sljedeće naredbe unutar direktorija aplikacije korištenjem terminala:

- `npm start` - pokreće razvojni server.
- `npm run build` - grupira aplikaciju u statične datoteke za produkciju.
- `npm test` - pokreće test runner.
- `npm run eject` - kopira ovisnosti o izgradnji (eng. *build*), konfiguracijske datoteke i skripte u direktorij aplikacije.

4.3. Struktura React aplikacije

Nakon što se React instalira, kreira se direktorij aplikacije s direktorijima i datotekama prikazanim na sljedećoj slici.



Slika 1: Struktura React aplikacije (Izvor: vlastita izrada)

Najvažniji direktorij je **src** koji sadrži sve dinamične komponente. Unutar direktorija nalazi se nekoliko važnih datoteka:

- **App.js** - predstavlja glavnu komponentu aplikacije.
- **App.css** - definira izgled i dizajn datoteke **App.js**.
- **App.test.js** - definira jedinične testove.
- **index.js** - predstavlja ulaznu točku aplikacije.
- **index.css** - definira izgled i dizajn datoteke **index.js** i ostale.

Direktorij **public** sadrži sve statične datoteke. Datoteke unutar ovog direktorija će zadržati isti naziv kada se prebace u produkciju. Datoteka **package.json** sadrži ukupnu konfiguraciju React aplikacije poput naziva i verzije aplikacije, svih skripti i ovisnosti.

Unutar direktorija **src** tijekom razvoj aplikacije kreiraju se novi direktoriji koji poboljšavaju organizaciju strukture aplikacije. Obično se stvara direktorij **components** u koji se dodaju sve komponente aplikacije i njihove povezane datoteke.

5. Next.js

Next.js je React okvir koji služi za razvoj full-stack web aplikacija. React komponente koriste se za razvoj korisničkih sučelja, dok se Next.js koristi za razvoj pozadinskih funkcionalnosti i optimizaciju. U pozadini, Next.js također apstrahira i automatski postavlja alate potrebne za React, kao što su povezivanje (eng. *bundling*) i kompajliranje (eng. *compiling*). Time se omogućuje programerima da se usredotoče na razvoj aplikacije umjesto da gube vrijeme na konfiguraciju.

5.1. Prednosti Next.js-a

Next.js pruža različite prednosti za izradu web aplikacija. Najveća prednost jest proširenje za React aplikacije, što omogućuje razvoj full-stack web aplikacija, što nije bilo moguće samo s Reactom.

Istraživanje prevedeno od strane Fariz, Lazuardy i Anggraini 2022. godine navodi nekoliko prednosti Next.js okvira [23]:

- Ugrađena podrška za CSS - Next.js posjeduje unaprijed ugrađenu podršku za CSS.
- Jednostavan mehanizam usmjeravanja (eng. *routing*) - Next.js posjeduje jednostavne mehanizme usmjeravanja koji uklanjaju potrebu za korištenjem trećih alata.
- Predrenderiranje (eng. *Pre-rendering*) - Next.js podržava renderiranje na strani poslužitelja čime se smanjuje opterećenje preglednika i korisnik neće vidjeti praznu stranicu kod početnog učitavanja.
- Razni mehanizmi dohvaćanja podataka (eng. *Data Fetching*) - Next.js omogućuje dohvaćanje podataka putem SSR-a, SSG-a, renderiranja na klijentskoj strani (eng. *Client-Side Rendering - CSR*) i inkrementalnom statičkom regeneracijom (eng. *Incremental Static Regeneration - ISR*).
- Dobar za SEO - Korištenje SSR-a je dobro za SEO, a ne CSR jer se HTML datoteke prikazuju na poslužiteljskoj strani.

5.2. Glavne značajke Next.js-a

5.2.1. Usmjeravanje

Prema službenoj Next.js dokumentaciji [24] usmjeravanje predstavlja usmjerivač koji se temelji na datotečnom sustavu izgrađenom na komponentama poslužitelja koji podržava rasporede (eng. *layouts*), ugniježđeno usmjeravanje (eng. *nested routing*), stanja učitavanja (eng. *loading states*), rukovanje pogreškama (eng. *error handling*) i više.

Next.js nudi dvije opcije usmjeravanja. Prva opcija usmjeravanja je takozvani *Pages Router* koja koristi direktorij **pages** unutar glavnog direktorija **app**. U ovom slučaju usmjeravanje se vrši tako da se naziv datoteke upiše izravno u direktorij **pages**, nakon čega URL adresa automatski prikazuje na ime datoteke u mapi **pages**.

Druga opcija usmjeravanja je takozvani *App Router* i dostupna je od verzije Next.js 13 te je preporučeno njegovo korištenje kako bi se maksimalno iskoristile Reactove najnovije značajke. Unutar glavnog direktorija **app** dodaju se direktoriji koji se koriste za definiranje putanja. Unutar tih direktorija koriste se datoteke poput `page.jsx`, `layout.jsx` i ostalih koje definiraju što će se prikazivati na ruti definiranoj na temelju naziva direktorija. U ovom slučaju `layout.jsx` će definirati izgled za sve stranice unutar direktorija koji predstavlja određenu rutu, dok će `page.jsx` definirati izgled stranice na definiranoj ruti.

5.2.2. Renderiranje

Prema službenoj Next.js dokumentaciji [24] renderiranje obuhvaća renderiranje na strani klijenta i poslužitelja s komponentama klijenta i poslužitelja. Dodatno optimizirano sa statičkim i dinamičkim renderiranjem na poslužitelju. Renderiranjem se programski kod pretvara u korisnička sučelja. React i Next.js omogućuju stvaranje hibridnih web aplikacija u kojima se dijelovi programskog koda mogu prikazati na poslužitelju ili klijentu.

Next.js omogućuje više načina renderiranja kako bi se prilagodio različitim slučajevima uporabe, optimizirajući performanse i korisničko iskustvo. Već su prethodno spomenute moguće metode renderiranja, a u nastavku su dodatno pojašnjene:

1. Client-Side Rendering (CSR) - Početni se HTML prikazuje na klijentskoj strani. React komponente dohvaćaju potrebne podatke nakon početnog učitavanja stranice koristeći JavaScript na klijentskoj strani.
2. Server-Side Rendering (SSR) - HTML stranice se generiraju na svaki zahtjev. Poslužitelj dohvaća potrebne podatke i prikazuje HTML za svaki dolazni zahtjev.
3. Static Site Generation (SSG) - HTML stranice se generiraju tijekom izgradnje. To znači da se sadržaj dohvaća i da se HTML unaprijed renderira kod izgradnje aplikacije.
4. Incremental Static Regeneration (ISR) - Omogućuje postupno ažuriranje statičnih stranica nakon izgradnje web mjesta. Stranice se ponovno generiraju u pozadini kako zahtjevi pristižu, na temelju razdoblja ponovne provjere.

5.2.3. Dohvaćanje podataka

Dohvaćanje podataka je važan dio svake aplikacije. Prema službenoj Next.js dokumentaciji [25] postoje četiri načina dohvaćanja podataka:

1. Na poslužitelju korištenjem `fetch` - Next.js proširuje izvorni `fetch` Web API kako bi memorirao zahtjeve za dohvaćanje tijekom renderiranja stabla React komponenata. Moguće

je korištenje `fetch` s `async/await` u komponentama poslužitelja, rukovateljima rutama i u radnjama poslužitelja.

2. Na poslužitelju korištenjem biblioteka trećih strana - Korištenjem biblioteka poput Axios. Omogućuje naprednije konfiguracije i rukovanje HTTP zahtjevima u usporedbi s izvornim API-jem za dohvaćanje.

3. Na klijentu korištenjem rukovatelja rutom (eng. *Route Handler*) - Rukovatelji rutama izvršavaju se na poslužitelju i vraćaju podatke klijentu. Ovo je korisno kada se ne želi izložiti osjetljive informacije klijentu, kao što su API tokeni.

4. Na klijentu korištenjem biblioteka trećih strana - Korištenjem biblioteka poput SWR ili TanStack Query. Ovaj se pristup koristi za rukovanje dohvaćanjem podataka, predmemorijom (eng. *caching*), sinkronizacijom (eng. *synchronization*) i više na strani klijenta nakon početnog učitavanja stranice.

5.2.4. Stiliziranje

Next.js podržava različite načine stiliziranja aplikacije, uključujući [26]:

- Globalni CSS - Dostupan je i primjenjuje se na sve komponente.
- CSS moduli - Namijenjen isključivo određenoj komponenti.
- Tailwind CSS - CSS okvir koji omogućuje brze prilagođene dizajne sastavljanjem klasa.
- Sass - CSS pretprocesor koji proširuje CSS sa značajkama kao što su varijable, ugniježđena pravila i mixins.
- CSS-in-JS - Ugrađivanjem CSS-a izravno u JavaScript komponente, omogućujući dinamičko i ograničeno stiliziranje.

5.2.5. Optimizacija

Next.js dolazi s nizom ugrađenih optimizacija koje su dizajnirane za poboljšanje brzine aplikacije.

Ugrađene komponente uklanjaju složenost implementacije uobičajenih optimizacija korisničkog sučelja. U ove komponente spadaju [27]:

- Slike - Izgrađeno na izvornom `<img>` elementu. Komponenta slike optimizira slike za izvedbu odgođenim učitavanjem i automatskom promjenom veličine slika na temelju veličine uređaja.
- Veze - Izgrađeno na izvornim `<a>` oznakama. Komponenta veze unaprijed dohvaća stranice u pozadini za brže i glađe prijelaze stranica.

- Skripte - Izgrađene na izvornim `<script>` oznakama. Komponenta skripte daje kontrolu nad učitavanjem i izvršavanjem skripti treće strane.

Next.js optimizira i metapodatke koji pomažu tražilicama da bolje razumiju sadržaj web stranice, što može rezultirati i boljim SEO-om te omogućuju prilagodbu načina na koji je sadržaj predstavljen na društvenim medijima. Omogućuju izmjenu `<head>` elementa stranice te se mogu konfigurirati na dva načina [27]:

- Metapodaci temeljeni na konfiguraciji
- Metapodaci temeljeni na datoteci

Next.js unutar direktorija aplikacije kreira direktorij **public**, a datoteke unutar tog direktorija mogu biti predmemorirane od strane pružatelja mreže za dostavljanje podataka (eng. *Content delivery network - CDN*) tako da se isporučuju učinkovitije.

5.2.6. TypeScript

Next.js pruža mogućnost korištenja TypeScripta za razvoj React aplikacija. Dolazi s ugrađenom podrškom za TypeScript za automatsku instalaciju potrebnih paketa i konfiguriranje odgovarajućih postavki te uz sam TypeScript dodatak za uređivač programskog koda.

TypeScript može pomoći u [28]:

- Pružanju upozorenja ako se proslijede nevažeće vrijednosti za opcije konfiguracije segmenata.
- Prikaz dostupnih opcija i dokumentacije u kontekstu.
- Osiguravanje da se direktiva klijenta koristi ispravno.
- Osiguravanje da se vezne točke za klijente (poput `useState`) koriste samo u komponentama klijenta.

5.3. Instalacija Next.js-a

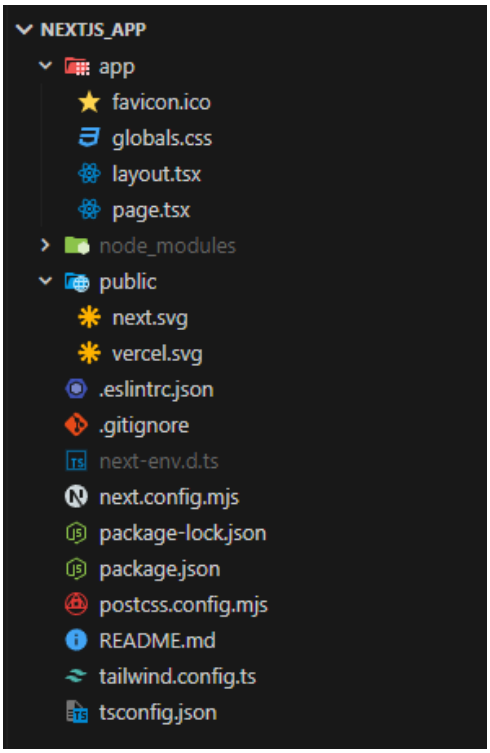
Instalaciju Next.js aplikacije moguće je sprovesti na dva načina. Postoji opcija automatske instalacije koja je i preporučena od strane Next.js dokumentacije te opcija ručne instalacija koja zahtjeva praćenje dokumentacije kako bi se pravilno podesio Next.js projekt.

U nastavku je objašnjena automatska instalacija Next.js aplikacije korištenjem terminala.

Instalacija Next.js aplikacije pokreće se pokretanjem naredbe `npx create-next-app@latest` unutar terminala. Nakon pokretanja instalacije postavlja se nekoliko upita na temelju kojih će se konfigurirati Next.js aplikacija i instalirati sve potrebne ovisnosti i dodaci. Radi se o upitima vezanih uz naziv aplikacije, korištenje TypeScripta, ESLinta, Tailwind CSS-a, **src**/ direktorija, App Routera te se pruža mogućnost promjene zadanog aliasa za uvoz (eng. *import*).

5.4. struktura Next.js aplikacije

Nakon instalacije Next.js-a, kreira se direktorij aplikacije s direktorijima i datotekama prikazanim na sljedećoj slici.



Slika 2: Struktura Next.js aplikacije (Izvor: vlastita izrada)

Direktoriji i datoteke koji će biti kreirani ovise o odgovorima na upite prilikom instalacije Next.js aplikacije. U ovom slučaju instalirani su svi dodaci i ovisnosti koje nudi instalacija Next.js projekta osim korištenja **src** direktorija.

Kreiran je **app** direktorij koji predstavlja glavni direktorij same aplikacije i unutar kojeg je omogućeno provođenje usmjeravanja. Unutar **app** direktorija kreirane su datoteke `globals.css` koja predstavlja globalni CSS, `layout.tsx` koji predstavlja korisničko sučelje koje će primjenjivati sve rute unutar direktorija **app** te `page.tsx` koji sadrži korisničko sučelje namijenjeno isključivo ruti u kojoj se nalazi. Unutar **app** direktorija kreiraju se dodatni direktoriji i datoteke poštujući pravila usmjeravanja korištenjem App Routera.

Kreiran je i direktorij **public** koji služi za posluživanje statičkih resursa poput slika, fontova i drugih datoteka. Pored tih direktorija kreirane su i druge datoteke poput `next.config.mjs` za konfiguraciju Next.js-a, `package.json` koja sadrži metapodatke o projektu, uključujući ovisnosti, skripte i druge detalje konfiguracije, `.eslintrc.json` za konfiguraciju ESLinta, `tsconfig.json` za konfiguraciju TypeScripta, `tailwind.config.ts` za konfiguraciju Tailwind CSS-a i druge.

6. MySQL

MySQL je jedan od najpopularnijih sustava za upravljanje relacijskim bazama podataka otvorenog koda koji koristi SQL za upravljanje bazama podataka. Razvijen je od strane tvrtke MySQL AB, a trenutno je u vlasništvu Oracle Corporationa.

MySQL ima široku upotrebu u web aplikacijama i nudi robusne mogućnosti za pohranu, pretraživanje i upravljanje podacima. Kombinacija s drugim tehnologijama poput Reacta, Next.js-a, Node.js-a i PHP-a omogućuje razvoj složenih i dinamičkih web stranica i aplikacija.

6.1. Karakteristike MySQL-a

MySQL je poznat po svojoj brzini, pouzdanosti i jednostavnosti korištenja. Posjeduje bogatu i opsežnu dokumentaciju o tome kako instalirati i upravljati MySQL bazom podataka, što uključuje i različite alate trećih strana. Programerima omogućuje da ne uključe određenu značajku SQL-a, čime prioritet mogu dati brzini. [18]

Prema službenoj dokumentaciji, MySQL posjeduje sljedeće karakteristike [29]:

- Otvorenog koda - MySQL je dostupan kao softver otvorenog koda, što znači da je besplatan za korištenje i prilagodbu prema potrebama korisnika.
- Prenosivost (eng. *Portability*) - MySQL je napisan u C i C++ jezicima i radi na mnogim platformama.
- Vrste podataka (eng. *Data Types*) - Podržava razne tipove podataka što omogućuje fleksibilnu pohranu različitih vrsta podataka.
- Izjave (eng. *Statements*) i funkcije - Nudi punu podršku za SQL operatore i funkcije, GROUP BY, ORDER BY, JOIN izjave i MySQL specifične SHOW izjave.
- Sigurnost - Dolazi s predinstaliranom skriptom koja pomaže u unapređenju sigurnosti baze podataka kroz implementaciju različitih sigurnosnih mehanizama.
- Skalabilnost i ograničenja - Podržava velike baze podataka s milijunima zapisa i tisućama tablica, omogućujući rad s ogromnim količinama podataka.
- Povezivost - MySQL klijenti mogu se povezati putem različitih protokola kao što su TCP/IP, imenovane cijevi (eng. *named pipes*), zajednička memorija (eng. *shared memory*) i soketi Unix domene (eng. *Unix domain sockets*).
- Lokalizacija - Podrška za različite jezične skupove i regionalne postavke, uključujući Unicode, omogućuje globalnu upotrebu.
- Klijenti i alati - Uključuje različite klijente i alate poput mysql, mysqladmin i grafički alat MySQL Workbench, koji olakšavaju upravljanje bazom podataka i administraciju.

6.2. Tipovi podataka i upiti u MySQL-u

6.2.1. Osnovni tipovi podataka

MySQL podržava razne tipove podataka koji omogućuju pohranu različitih vrsta informacija:

- Numerički tipovi
  - **INT** - Cijeli brojevi.
  - **FLOAT** - Decimalni brojevi s pokretnim zarezom.
  - **DOUBLE** - Decimalni brojevi s dvostrukom preciznošću.
  - **DECIMAL** - Decimalni brojevi s fiksnom preciznošću.
- Znakovni tipovi
  - **CHAR** - Fiksna duljina znakova.
  - **VARCHAR** - Varijabilna duljina znakova.
  - **TEXT** - Veliki tekstualni podaci.
  - **BLOB** - Binari Large Objects za pohranu binarnih podataka poput slika i datoteka.
- Datum i vrijeme
  - **DATE** - Datum (YYYY-MM-DD).
  - **TIME** - Vrijeme (HH:MM).
  - **DATETIME** - Datum i vrijeme (YYYY-MM-DD HH:MM).
  - **TIMESTAMP** - Datum i vrijeme s vremenskom zonom.
- Logički tipovi
  - **BOOLEAN** - Logička vrijednost (TRUE ili FALSE).

6.2.2. Upiti

SQL upiti omogućuju interakciju s bazom podataka za dohvaćanje, dodavanje, ažuriranje i brisanje podataka. U nastavku su prikazani i pojašnjeni SQL upiti s primjerom korištenja.

- **SELECT** - Koristi se za dohvaćanje zapisa iz baze podataka.

```
SELECT ime, prezime FROM korisnici;
```

- **INSERT** - Koristi se za dodavanje novih zapisa u tablicu.

```
INSERT INTO korisnici(ime, prezime)VALUES('Ivan', 'Horvat');
```

- **UPDATE** - Koristi se za ažuriranje postojećih zapisa.

```
UPDATE korisnici SET prezime = 'Ivic' WHERE ime = 'Ivan';
```

- **DELETE** - Koristi se za brisanje zapisa iz tablice.

```
DELETE FROM korisnici WHERE ime = 'Ivan';
```

6.3. Oblikovanje MySQL baze podataka

Proces oblikovanja baza podataka je izrazito bitan dio razvoja bilo koje aplikacije. Potrebno je posvetiti dovoljno vremena i truda za oblikovanje same baze podataka. Oblikovanje baze podataka se može promatrati kao postavljanje nacrta kuće, što bi značilo da bez dobro oblikovane baze podataka nije pametno krenuti razvijati aplikaciju jer može doći do dodatnih troškova ponovno oblikovanja i ponovne implementacije same aplikacije.

Prema knjizi [30] navodi se da postoje tri glavne faze kod oblikovanja baze podataka:

1. Analiza zahtjeva (eng. *Requirements analysis*) - Uključuje prikupljanje i definiranje poslovnih potreba i funkcionalnih zahtjeva. Cilj je razumjeti što aplikacija treba postići i koje podatke treba pohraniti.
2. Konceptualni dizajn (eng. *Conceptual design*) - Podrazumijeva izradu konceptualnog modela baze podataka, često s pomoću ER (Entity-Relationship) dijagrama. Definiraju se entiteti, njihovi atributi i odnosi među njima.
3. Logički dizajn (eng. *Logical design*) - Uključuje pretvaranje konceptualnog modela u relacijski model. Ovo podrazumijeva definiranje tablica, stupaca, primarnih ključeva i stranih ključeva.

Za pretvaranje konceptualnog modela u shemu MySQL baze podataka može se koristiti alat otvorenog koda MySQL Workbench.

Nakon prethodnih koraka potrebno je provesti i fizičko modeliranje koje podrazumijeva implementaciju logičkog modela u stvarnu MySQL bazu podataka. Definiraju se stvarne tablice i indeksi u MySQL-u, kao i postavke za pohranu podataka, particioniranje i optimizaciju performansi.

Fizičkim modeliranjem sam proces oblikovanja baze podataka još nije završio već je potrebno sve podatke dodatno organizirati kako bi se smanjila redundantnost i osigurao integritet podataka. Taj proces naziva se normalizacija (eng. *normalization*). To uključuje razbijanje tablica u manje, povezane tablice i definiranje odnosa između njih. Postoji nekoliko normalnih formi (NF), od kojih je najčešće korištena treća normalna forma (3NF).

U nekim slučajevima, kako bi se poboljšale performanse, može biti potrebno izvršiti denormalizaciju. To podrazumijeva kombiniranje tablica kako bi se smanjio broj `join` operacija u upitima. Denormalizacija se primjenjuje pažljivo kako bi se izbjegla nepotrebna redundantnost podataka.

Nad podacima u bazi potrebno je postaviti određena pravila kako bi se osigurala njihova točnost i integritet. To se postiže postavljanjem pravila koja se nazivaju ograničenja (eng

constraints). To uključuje primarne ključeve, strane ključeve, jedinstvena ograničenja i provjere. Indeksi se koriste za ubrzavanje pristupa podacima.

Bazu podataka je potrebno osigurati definiranjem korisničkih prava i pristupa te primjenom sigurnosnih mjera kao što su šifriranje podataka i izradom sigurnosnih kopija (eng. *backups*).

Nakon oblikovanja i implementacije, baza podataka se mora temeljito testirati kako bi se osiguralo da zadovoljava sve poslovne zahtjeve i da radi ispravno pod različitim opterećenjima.

Nakon što je baza uspješno oblikovana potrebno je provoditi kontinuirano održavanje i optimizaciju baze podataka kako bi se osigurale njene performanse i skalabilnost. Ovo uključuje redovite sigurnosne kopije, ažuriranja, optimizaciju upita i praćenje performansi.

6.4. Integracija MySQL-a u React i Next.js

Integracija MySQL baze podataka u React i Next.js aplikaciju zahtijeva nekoliko koraka, uključujući postavljanje baze podataka, instalaciju potrebnih paketa, konfiguraciju konekcije i definiranje API ruta. Integraciju je moguće obaviti koristeći različite pristupe kao što su mysql2, sequelize, Knex.js, Prisma i drugi.

U nastavku će biti ukratko prikazan postupak integracije baze podataka korištenjem paketa mysql2. Postupak obuhvaća postavljanje MySQL servera pa sve do izrade konfiguracijske datoteke i API ruta za dohvat podataka koje su prikazane na jednostavnom primjeru rada s korisnicima.

6.4.1. Postavljanje MySQL baze podataka

Postavljanje MySQL baze podataka potrebno je izvršiti u dva koraka:

1. Instalacija MySQL-a - Potrebno je instalirati i podesiti MySQL server te osigurati da je server pokrenut.
2. Eksportiranje EER modela - Potrebno je izvesti EER model iz MySQL Workbencha kao SQL skriptu i pokrenuti je kako bi se kreirala shema baze podataka unutar MySQL servera.

6.4.2. Modul mysql2

mysql2 je popularna biblioteka za Node.js koja se koristi za pristupanje MySQL bazama podataka, omogućujući korisnicima povezivanje, postavljanje upita i interakciju s njima. Ovo je novija verzija biblioteke mysql koja je osmišljena da pruži bolje performanse, veću sigurnost i dodatne funkcionalnosti u odnosu na staru biblioteku.

- **Instalacija paketa**

```
npm install mysql2
```

- **Konfiguracija i povezivanje**

```
// config/database.js
const mysql = require('mysql2/promise');

const pool = mysql.createPool({
    host: 'localhost',
    user: 'username',
      password: 'password',
      database: 'database_name',
});

module.exports = pool;
```

- **API ruta za dohvat podataka**

```
// korisnici/page.js
import pool from '../../config/database';

export default async function handler(req, res) {
    if (req.method === 'GET') {
        const [redovi] = await pool.query('SELECT * FROM korisnici');
        res.status(200).json(redovi);
    } else if (req.method === 'POST') {
        const { ime } = req.body;
        const [rezultat] = await pool.query('INSERT INTO korisnici (ime)
            VALUES (?)', [ime]);
        res.status(201).json({ id: rezultat.insertId, ime });
    }
}
```

Integracija MySQL baze podataka u React i Next.js aplikaciju može se ostvariti korištenjem različitih alata, ovisno o potrebama i preferencijama projekta. Odabir pravog alata ovisi o specifičnim zahtjevima aplikacije i preferencijama razvojnog tima.

7. Primjer web aplikacije

Razvijena web aplikacija omogućuje korisnicima pregled raznih ponuda videoigara s ciljem pronalaska najpovoljnije ponude u odnosu na brojne platforme na kojima se videoigra prodaje. Prilikom izrade web aplikacije korišten je CheapShark API koji sadrži mnoštvo ponuda raznih videoigara i koje se svakodnevno ažuriraju s aktualnim ponudama. Korištenje glavnih značajki web aplikacije omogućeno je svim korisnicima, dok je za određene značajke potrebna registracija, odnosno izrada korisničkog profila.

7.1. Funkcionalnosti po ulogama

7.1.1. Neprijavljeni korisnik

U nastavku je prikazan popis funkcionalnosti koje su omogućene neprijavljenom korisniku, odnosno svim korisnicima web aplikacije.

- Neprijavljeni korisnik ima pristup početnoj stranici s pregledom svih ponuda videoigara na raznim platformama.
- Neprijavljenom korisniku omogućeno je sortiranje ponuda na temelju kriterija ocjene ponude, uštede, snižene cijene te ocjene videoigre.
- Neprijavljenom korisniku omogućeno je filtriranje ponuda na temelju platformi te raspona snižene cijene videoigara.
- Neprijavljenom korisniku omogućeno je direktno prebacivanje na platformu ponude videoigre.
- Neprijavljenom korisniku omogućen je pregled detalja ponude videoigre.
- Neprijavljenom korisniku omogućen je pregled komentara i ocjena videoigre.
- Neprijavljenom korisniku omogućena je pretraga videoigra na temelju naziva.
- Neprijavljenom korisniku omogućena je promjena valute.
- Neprijavljenom korisniku omogućena je registracija i prijava u web aplikaciju.

7.1.2. Prijavljeni korisnik

Prijavljeni korisnik posjeduje iste funkcionalnosti kao i neprijavljeni korisnik uz dodatne funkcionalnosti. U nastavku je prikazan popis funkcionalnosti koje se nude prijavljenim korisnicima u odnosu na neprijavljene.

- Prijavljenom korisniku omogućeno je postavljanje ocjene i komentara na videoigru.
- Prijavljenom korisniku koji je običan korisnik web aplikacije omogućeno je brisanje isključivo vlastitih komentara i ocjena, dok je prijavljenom korisniku s ulogom administratora omogućeno brisanje svih komentara i ocjena neovisno od strane koga je komentar i ocjena postavljena.
- Prijavljenom korisniku omogućeno je dodavanje videoigre na listu želja.
- Prijavljenom korisniku omogućen je pregled liste želja koje prikazuju sve postavljene videoigre na listu želja od strane prijavljenom korisnika.
- Prijavljenom korisniku omogućeno je postavljanje i uklanjanje obavijesti za videoigru prilikom čega je potrebno odabrati najnižu cijenu za koju će se obavijesti poslati korisniku putem e-mail adrese.
- Prijavljenom korisniku omogućeno je uklanjanje videoigre s liste želja prilikom čega se uklanja i obavijest ako je postavljena.
- Prijavljenom korisniku omogućeno je prikupljanje značaka koje predstavljaju postignuća prilikom korištenja web aplikacije kao što su značke za broj objavljenih komentara i ocjena te značke za broj postavljenih videoigra na listu želja.
- Prijavljenom korisniku omogućeno je mijenjati korisničke podatke, odnosno korisničko ime, lozinku te profilnu sliku.

7.2. Korištene biblioteke i moduli

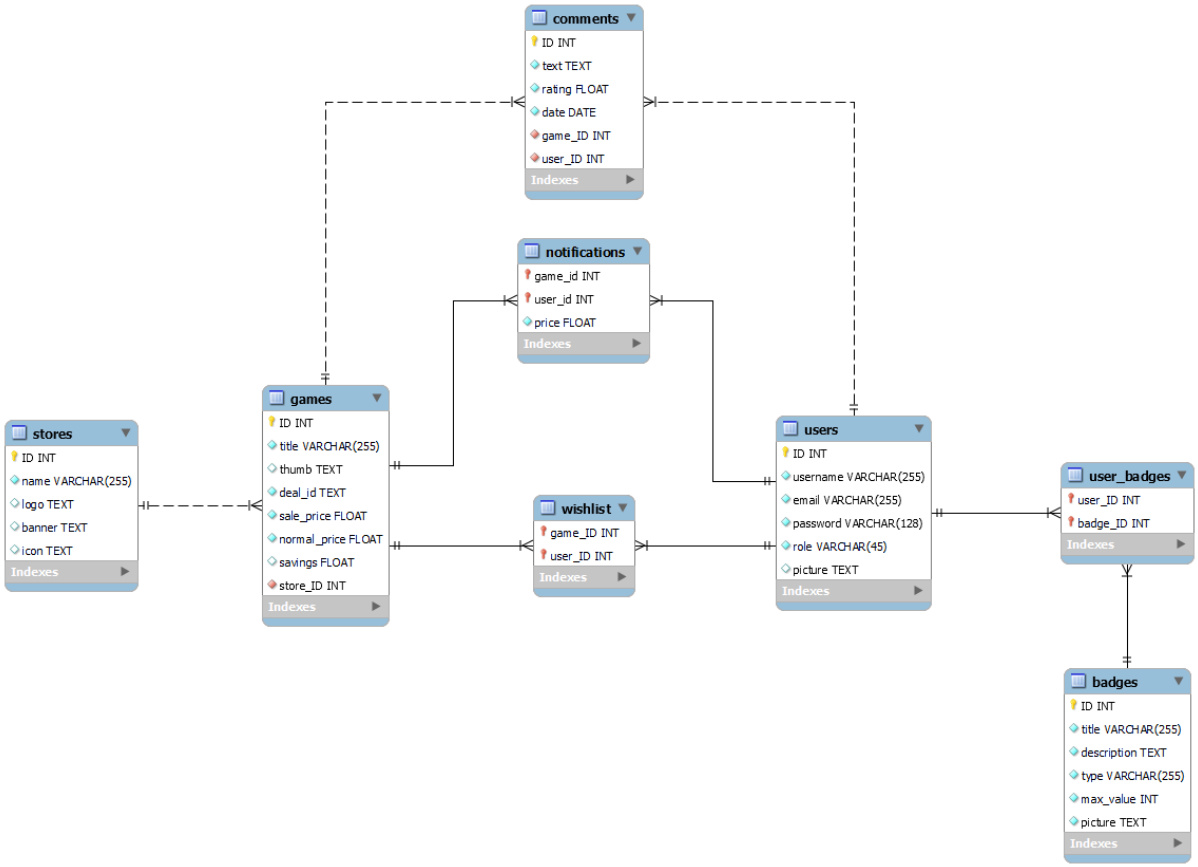
U nastavku su navedene i ukratko opisane sve biblioteke i moduli korišteni prilikom izrade web aplikacije. Korištenjem biblioteka i modula postignuta je jednostavnija i sigurnija implementacija funkcionalnosti web aplikacije.

- Tailwind CSS - CSS okvir koji omogućuje brzi razvoj responzivnih i prilagodljivih korisničkih sučelja koristeći razne unaprijed definirane klase bez potrebe pisanja vlastitog CSS-a.
- Next UI - Komponentna biblioteka za React koja nudi prilagodljive UI komponente optimizirane za Next.js aplikacije, olakšavajući brzu izradu korisničkih sučelja.
- mysql2 - Node.js modul koji omogućuje povezivanje i rad s MySQL bazama podataka, pružajući bolje performanse i podršku za Promises i `async/await`.
- bcrypt - Node.js modul koji omogućuje enkripciju i hashiranje lozinki koristeći bcrypt algoritam, što pomaže u sigurnom pohranjivanju lozinki.
- NextAuth - Biblioteka koja pruža autentifikaciju za Next.js aplikacije koja podržava različite metode autentifikacije, uključujući OAuth, e-mail s pomoću čarobnih poveznica (eng. *magic links*), i druge.

- eslint - Alat za analizu koda koji pomaže u održavanju kvalitete koda i dosljednosti kroz pronalaženje i automatsko ispravljanje problema u JavaScript i TypeScript kodu.
- path - Ugrađeni Node.js modul koji omogućuje rad s putanjama datoteka i direktorija na način neovisan o operacijskom sustavu.
- fs - Ugrađeni Node.js modul koji omogućuje interakciju s datotečnim sustavom, uključujući čitanje i pisanje datoteka, upravljanje direktorijima i druge operacije vezane uz datotečni sustav.

7.3. ERA dijagram baze podataka

Sljedeća slika prikazuje entitetsko-relacijski model baze podataka web aplikacije. Iz ERA dijagrama vidljivo je da se baza podataka sastoji od osam tablica koje su: *stores, games, notifications, comments, wishlist, users, user_badges* i *badges*.

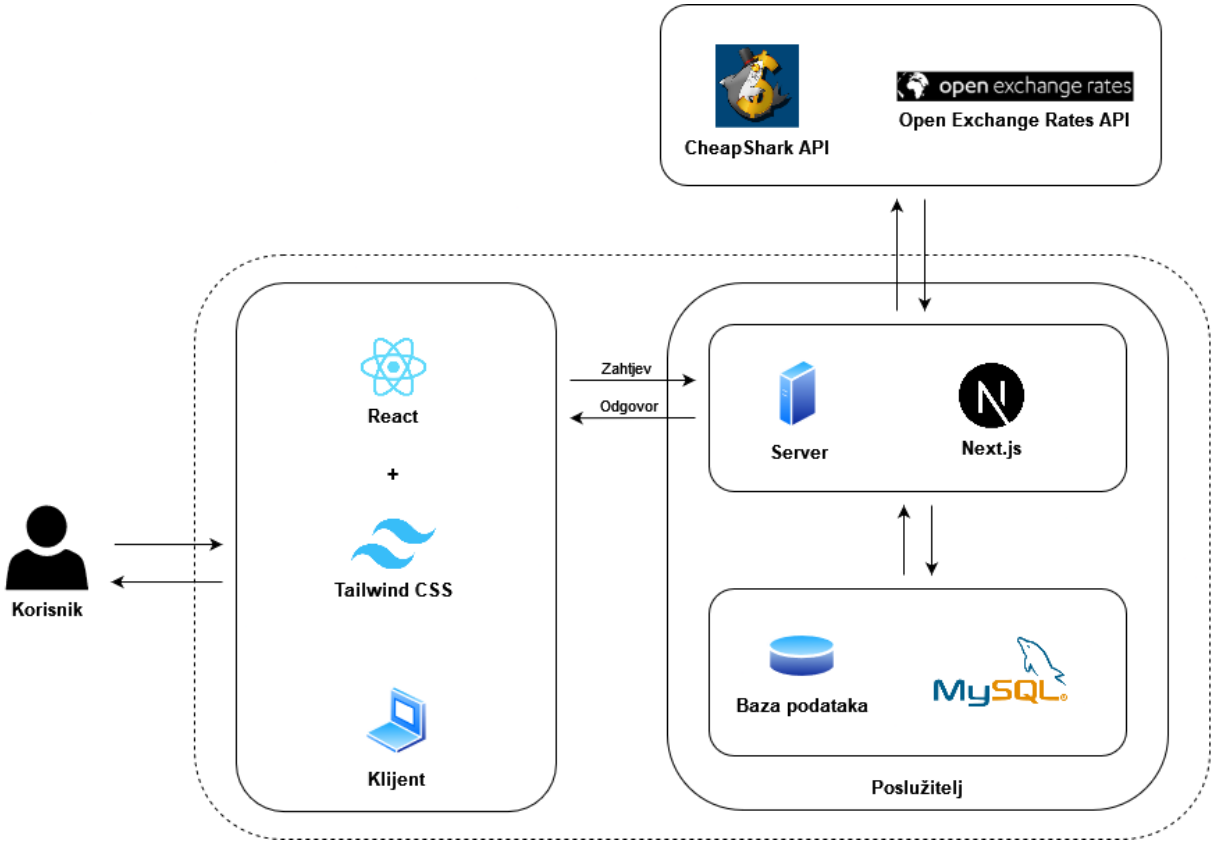


Slika 3: ERA dijagram baze podataka (Izvor: vlastita izrada)

Tablica *stores* sadrži podatke o platformama koje nude ponude videoigara, tablica *games* sadrži podatke o videoigrama, tablica *notifications* sadrži podatke o postavljenim obavijestima, tablica *comments* sadrži podatke o postavljenim komentarima od strane korisnika, tablica *wishlist* sadrži podatke o videoigrama na listi želja postavljenih od strane korisnika, tablica *users* sadrži podatke o registriranim korisnicima, tablica *user_badges* sadrži podatke o prikupljenim značkama od strane korisnika dok tablica *badges* sadrži podatke o samim značkama.

7.4. Arhitektura web aplikacije

Sljedeća slika prikazuje dijagram arhitekture web aplikacije koji služi za vizualni prikaz svih elemenata koji čine dio ili cijeli sustav te pomaže u razumijevanju izgleda sustava ili aplikacije.



Slika 4: Dijagram arhitekture web aplikacije (Izvor: vlastita izrada)

Prikazana je arhitektura web aplikacije koja koristi moderne web tehnologije, uključujući klijentsku stranu s Reactom i Tailwind CSS-om, poslužiteljsku stranu s Next.js-om, bazu podataka s MySQL-om, i integracije s vanjskim API-ima.

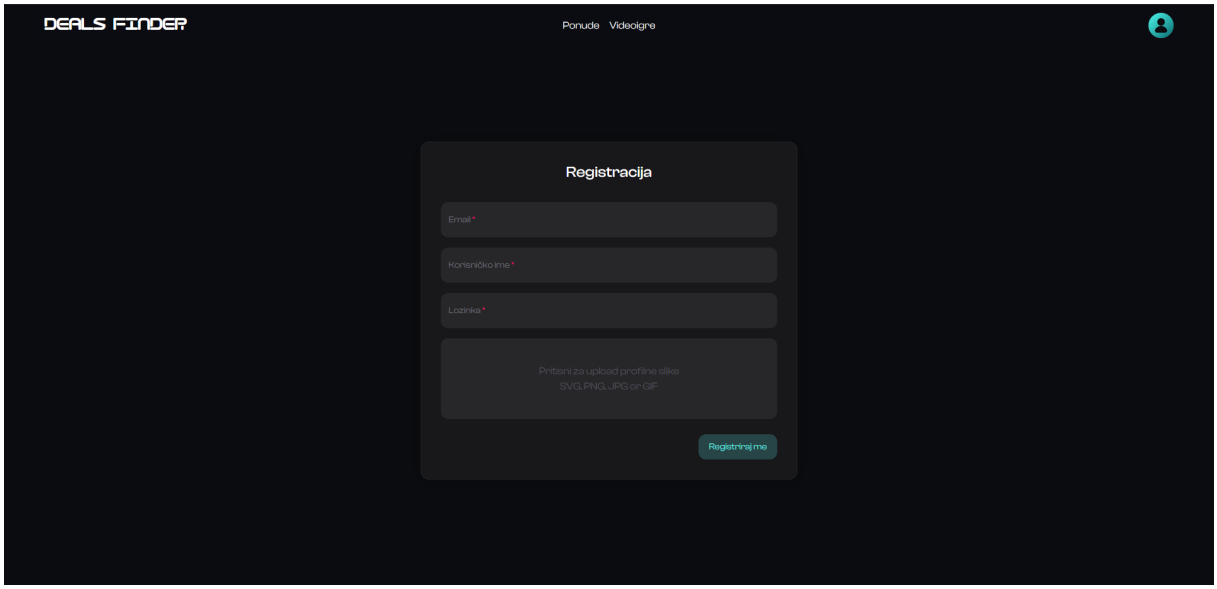
Korisnik komunicira s aplikacijom putem web preglednika. Korisničke akcije (poput klikanja na gumb ili popunjavanja forme) pokreću zahtjeve prema poslužitelju. Klijent (React aplikacija) šalje HTTP zahtjeve (kao što su GET, POST, PUT, DELETE) prema poslužitelju (Next.js). Poslužitelj obrađuje zahtjeve te u odnosu na zahtjev klijenta komunicira s bazom podataka ili vanjskim API-jem te odgovor šalje natrag klijentu.

7.5. Najvažniji dijelovi web aplikacije

U nastavku su navedeni i opisani najvažniji dijelovi razvijene web aplikacije te vizualno prikazani slikama ekrana web aplikacije uz odgovarajući programski kod.

7.5.1. Registracija korisnika

Kako bi korisnik mogao koristiti sve funkcionalnosti web aplikacije potrebno se prethodno prijaviti. Svaki korisnik prije toga mora proći proces registracije prilikom čega je potrebno unijeti neke osnovne podatke. Sljedeća slika prikazuje stranicu za registraciju korisnika.



Slika 5: Stranica registracije (Izvor: vlastita izrada)

Prilikom registracije korisnika validiraju se uneseni podaci te se provjerava jesu li uneseni svi podaci koji su obavezni. U ovisnosti na rezultat validacije i provjere unosa obaveznih podataka korisnika će se obavijestiti o pogrešci ili o uspješnoj registraciji te u tom slučaju preusmjeriti na stranicu prijave. U nastavku je prikazan programski kod funkcije na klijentskoj strani koja se poziva prilikom klika korisnika na gumb za registraciju.

```
const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");
    setSuccess("");

    const formData = new FormData();
    formData.append("username", username);
    formData.append("email", email);
    formData.append("password", password);
    formData.append("role", "user");
    if (picture !== null) {
        formData.append("picture", picture);
    }
```

```
    try {
        const { res, data } = await addUser(formData);

        if (!res.ok) {
            setError(data.error);
        } else {
            setSuccess(data.message);
            setTimeout(() => {
                router.push("/auth/login");
            }, 1000);
        }
    } catch (error) {
        throw new Error("Pogreška prilikom registracije: ", error);
    }
};
```

Funkcija prikuplja podatke forme te ih šalje pomoćnoj funkciji `addUser` koja se nalazi u zasebnoj datoteci i koja djeluje kao posrednik između klijenta i poslužitelja. U nastavku je prikazan programski kod pomoćne funkcije.

```
export const addUser = async (userData) => {
    try {
        const res = await fetch(`${BASE_URL}/auth/register`, {
            method: "POST",
            body: userData,
        });

        const data = await res.json();
        return { res, data };
    } catch (error) {
        console.error(error);
        throw error;
    }
};
```

Ova pomoćna funkcija koristi `fetch` za slanje podataka forme krajnjoj točki API-ja na strani poslužitelja. U nastavku je prikazan programski kod krajnje točke API-ja na strani poslužitelja.

```
export async function POST(req) {
    try {
        ...
        if (missingField) {
            return NextResponse.json({ error: missingField }, { status: 400 });
        }

        if (existingUser.length > 0 || existingEmail.length > 0) {
            return NextResponse.json(
                { error: "Korisnik pod navedenim korisničkim imenom ili e-mailom već
                    postoji!", },
                { status: 400 }
            );
        }
```

```
        ...
        const saltRounds = 10;
        const hashedPassword = await bcrypt.hash(password, saltRounds);

        await userDAO.createUser({
            username,
            email,
            password: hashedPassword,
            role,
            picturePath,
        });

        return NextResponse.json(
            { message: "Registracija uspješna!" },
            { status: 201 }
        );
    } catch (error) {
        console.error("Interna pogreška tijekom registracije: ", error);
        return NextResponse.json(
            { error: "Interna pogreška tijekom registracije!" },
            { status: 500 }
        );
    }
}
```

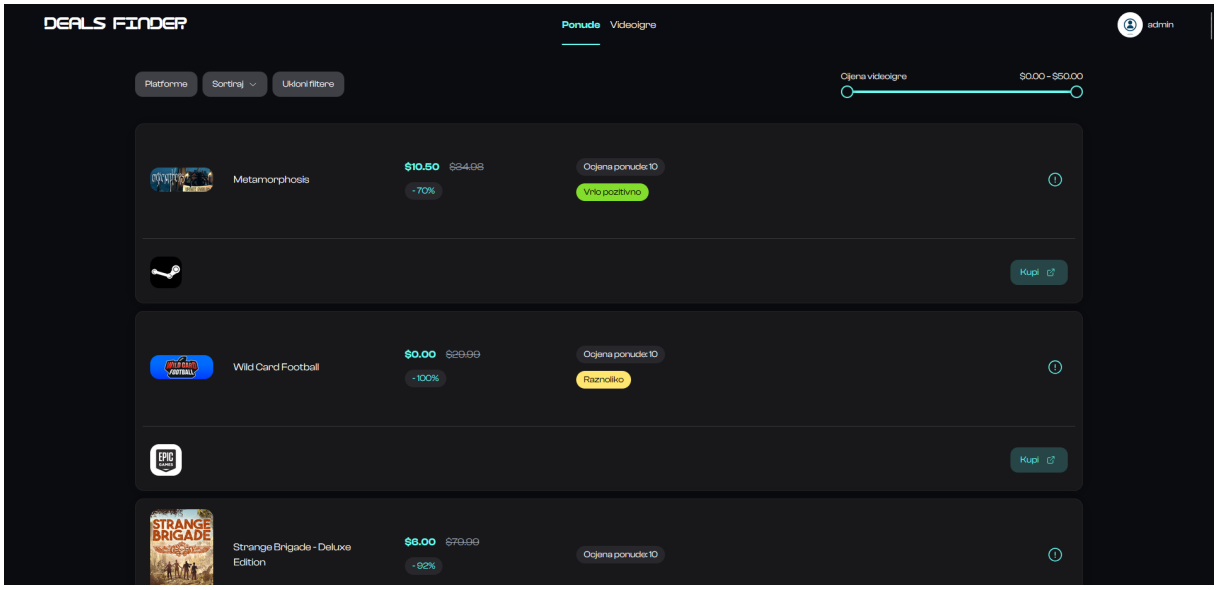
API ruta na poslužitelju prima zahtjev i izvlači podatke obrasca. Obavlja različite provjere, kao što je provjera obaveznih polja i provjera jesu li korisničko ime ili adresa e-pošte već registrirani. Poslužitelj zatim kriptira korisničku lozinku uz korištenje soli radi sigurnosti i kreira novog korisnika u bazi podataka s pomoću funkcije `createUser` iz klase `UserDAO`, koja je u interakciji s bazom podataka te šalje odgovor nazad klijentu. U nastavku je prikazan programski kod funkcije `createUser`.

```
async createUser({ username, email, password, role, picturePath }) {
    let query = "INSERT INTO users (username, email, password, role";
    let values = [username, email, password, role];

    if (picturePath) {
        query += ", picture) VALUES (?, ?, ?, ?, ?)";
        values.push(picturePath);
    } else {
        query += ") VALUES (?, ?, ?, ?)";
    }

    return await pool.query(query, values);
}
```

7.5.2. Pregled svih ponuda videoigara

Početna stranica pruža listu svih ponuda videoigara koje se trenutno nude na raznim platformama. Korisniku se nudi mogućnost sortiranja ponuda na temelju određenih kriterija te postavljanje filtera za cijenu ponude te prikaz željenih platformi kao i gumb za uklanjanje svih

filtera. Korisniku se također pruža mogućnost pregleda detalja same ponude pritiskom na gumb za detalje te direktno prebacivanje na platformu s ponudom. Sljedeća slika prikazuje stranicu pregleda svih ponuda videoigara.



Slika 6: Stranica pregleda svih ponuda videoigara (Izvor: vlastita izrada)

Pregled ponuda omogućen je vanjskim API-jem (CheapShark API) koji pruža sve podatke o trenutno aktivnim ponudama. U nastavku je prikazan programski kod na klijentskoj strani koji koristi React veznu točku `useEffect` koja poziva funkciju zaduženu za dohvaćanje informacija o svim ponudama s vanjskog API-ja prilikom prvog učitavanja stranice ili ako dođe do izmjene parametara filtera, sortiranja ili broja trenutne stranice. Također se kreira i lokalna pohrana dohvaćenih podataka platformi.

```
useEffect(() => {
    const fetchData = async () => {
        setLoading(true);

        try {
            const fetchParams = {
                ...params,
                pageNumber: params.pageNumber,
                sortBy: itemSorter.sortBy,
                desc: itemSorter.desc,
            };

            const [dealsResult, storesData] = await Promise.all([
                fetchDeals(fetchParams),
                fetchStores(),
            ]);

            setDeals(dealsResult.data);
            setTotalPages(dealsResult.totalPageCount || 1);

            const storesMap = storesData.reduce((acc, store) => {
                acc[store.storeID] = {
```

```
                    name: store.storeName,
                    logo: store.images.logo,
                    banner: store.images.banner,
                    icon: store.images.icon,
                };
                return acc;
            }, {});

            localStorage.setItem("stores", JSON.stringify(storesMap));
            setStores(storesMap);
        } catch (error) {
            console.error("Dohvaćanje podataka neuspješno:", error);
        } finally {
            setLoading(false);
        }
    };

    fetchData();
}, [params, itemSorter, convertPriceWithSymbol]);
```

API rute na strani poslužitelja obrađuju dolazne zahtjeve od klijenta, dohvaćaju potrebne podatke iz vanjskog API-ja, a zatim vraćaju odgovor nazad klijentu. U nastavku je prikazan programski kod krajnje točke API-ja za dohvaćanje podataka o ponudama koji čita parametre upita iz dolaznog zahtjeva te gradi URL za dohvaćanje podataka.

```
export async function GET(req) export async function GET(req) {
    const { searchParams } = new URL(req.url);

    try {
        const BASE_URL = process.env.NEXT_PUBLIC_CHEAP_SHARK_API_URL + "/deals";

        let url = `${BASE_URL}?`;

        const { storeID, pageNumber, lowerPrice, upperPrice, sortBy, desc } =
            Object.fromEntries(searchParams);

        if (pageNumber) url += `pageNumber=${pageNumber}&`;
        if (storeID) url += `storeID=${storeID}&`;
        if (lowerPrice) url += `lowerPrice=${lowerPrice}&`;
        if (upperPrice) url += `upperPrice=${upperPrice}&`;
        if (sortBy) url += `sortBy=${sortBy}&`;
        if (desc) url += `desc=${desc}&`;

        const res = await fetch(url);

        if (!res.ok) {
            throw new Error("Dohvaćanje podataka ponuda neuspješno!");
        }

        const parsedData = await res.json();
        const totalPageCount = res.headers.get("X-Total-Page-Count");

        const response = NextResponse.json(parsedData);
```

```
        response.headers.set("total-page-count", totalPageCount);

        return response;
    } catch (error) {
        console.error(error);
        return NextResponse.json(
            { error: "Interna pogreška kod dohvaćanja ponuda!" },
            { status: 500 }
        );
    }
}
```

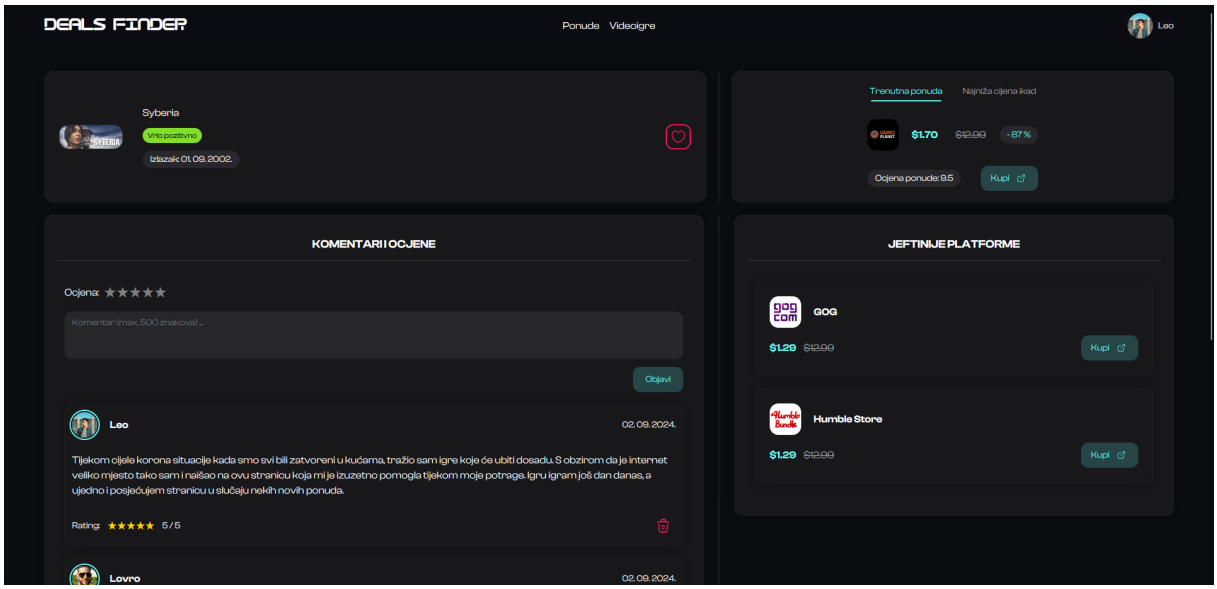
U nastavku je prikazan programski kod krajnje točke API-ja za dohvaćanje podataka o platformama.

```
export async function GET() {
    try {
        const BASE_URL = process.env.NEXT_PUBLIC_CHEAP_SHARK_API_URL;
        const res = await fetch(`${BASE_URL}/stores`);

        if (!res.ok) {
            throw new Error("Dohvaćanje podataka platformi neuspješno!");
        }

        const parsedData = await res.json();
        return NextResponse.json(parsedData);
    } catch (error) {
        console.error(error);
        return NextResponse.json(
            { error: "Interna pogreška kod dohvaćanja podataka platformi!" },
            { status: 500 }
        );
    }
}
```

Nakon što klijent primi podatke od poslužitelja oni se nadalje obrađuju, šalju komponentama i prikazuju korisniku.

7.5.3. Detaljan pregled određene ponude

Klikom na gumb za detalje korisnika se prebacuje na novu stranicu koja pruža više informacija o samoj ponudi. Sljedeća slika prikazuje stranicu s detaljima ponude.



Slika 7: Stranica detalja ponude (Izvor: vlastita izrada)

Detalji ponude dijelom se prikazuju korištenjem podataka o ponudi iz lokalne pohrane koja se kreira klikom na gumb za detalje, konkretno za prikaz svih podataka koji se korisniku prikazuju i na samom pregledu svih ponuda dok se podaci o izdavaču, datumu izlaska i jeftinijim platformama dohvaćaju korištenjem vanjskog API-ja (CheapShark API). U nastavku je prikazan programski kod na klijentskoj strani koji dohvaća podatke o ponudi iz lokalne pohrane i vanjskog API-ja.

```
useEffect(() => {
    if (dealID) {
        const selectedDealInfo = localStorage.getItem("selectedDeal");
        if (selectedDealInfo) {
            setSelectedDeal(JSON.parse(selectedDealInfo));
        }

        const fetchDeal = async () => {
            setLoading(true);
            try {
                const fetchedDeal = await fetchDealID(dealID);

                setGameInfo(fetchedDeal.gameInfo);
                setCheaperStores(fetchedDeal.cheaperStores);
                setCheapestPrice(fetchedDeal.cheapestPrice);
            } catch (error) {
                console.error("Dohvaćanje podataka neuspješno:", error);
            } finally {
                setLoading(false);
            }
        };

        fetchDeal();
    }
}, [dealID, convertPriceWithSymbol]);
```

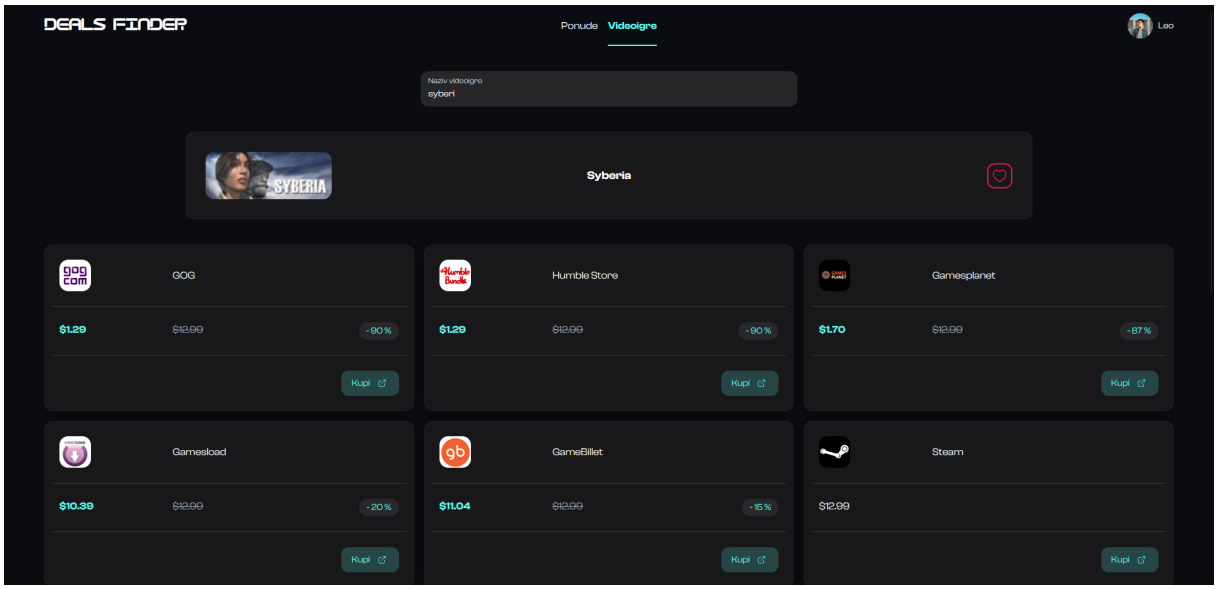
API ruta na strani poslužitelja obrađuje dolazne zahtjeve od klijenta, dohvaćaju potrebne podatke iz vanjskog API-ja, a zatim vraćaju odgovor nazad klijentu. U nastavku je prikazan programski kod krajnje točke API-ja za dohvaćanje detalja ponude. Izdvaja se parametar ID-a ponude iz zahtjeva te se gradi URL s ID-om ponude za dohvaćanje detalja.

```
export async function GET(req, { params }) export async function GET(req, { params
    }) {
    try {
        const BASE_URL = process.env.NEXT_PUBLIC_CHEAP_SHARK_API_URL;
        const { dealID } = params;

        if (!dealID) {
            return NextResponse.json(
                { error: "ID ponude nije definiran!" },
                { status: 400 }
            );
        }

        const url = `${BASE_URL}/deals?id=${encodeURIComponent(dealID)}`;

        try {
            const res = await fetch(url);
            if (!res.ok) {
                throw new Error(
                    "Dohvaćanje podataka detalja ponude neuspješno!"
                );
            }

            const parsedData = await res.json();
            return NextResponse.json(parsedData);
        } catch (error) {
            return NextResponse.json({ error: error.message }, { status: 500 });
        }
    } catch (error) {
        console.error(error);
        return NextResponse.json(
            { error: "Interna pogreška kod dohvaćanja detalja ponude!" },
            { status: 500 }
        );
    }
}
```

Nakon što klijent primi podatke od poslužitelja oni se nadalje obrađuju, šalju komponentama i prikazuju korisniku.

7.5.4. Pretraga videoigre na temelju naziva

Uz pregled svih ponuda, korisniku se nudi pretraga ponuda na temelju naziva videoigre na zasebnoj stranici. Sljedeća slika prikazuje stranicu s primjerom pretrage videoigre.



Slika 8: Primjer pretrage videoigre (Izvor: vlastita izrada)

Funkcionalnost je omogućena korištenjem vanjskog API-ja (CheapShark API). Prilikom upisa naziva videoigre korisniku se ispisuje lista naziva svih videoigara koje CheapShark API posjeduje u svojoj bazi podataka te klikom na naziv videoigre korisniku će se prikazati sve ponude za odabranu videoigru.

U nastavku je prikazan programski kod na klijentskoj strani koji provjerava unos korisnika u polje za naziv videoigre i provjerava koju je videoigru korisnik odabrao te na temelju tih podataka dohvaća ponude odabrane videoigre.

```
useEffect(() => {
    const fetchData = async () => {
        if (!search && !selectedGame) return;

        setLoading(true);
        try {
            if (search && search.length > 3) {
                const gamesData = await fetchGames(search);
                setGames(gamesData);
            }

            if (selectedGame) {
                const gameData = await fetchGameID(selectedGame);
                const storesData = await fetchStores();

                setGameDeals(gameData.deals);
                setGameInfo(gameData.info);

                const storesMap = storesData.reduce((acc, store) => {
                    acc[store.storeID] = {
                        name: store.storeName,
                        logo: store.images.logo,
                        banner: store.images.banner,
                        icon: store.images.icon,
```

```
                    };
                    return acc;
                }, {});
                setStores(storesMap);
            }
        } catch (error) {
            console.error("Dohvaćanje podataka neuspješno:", error);
        } finally {
            setLoading(false);
        }
    };

    fetchData();
}, [search, selectedGame]);
```

API rute na strani poslužitelja obrađuju dolazne zahtjeve od klijenta, dohvaćaju potrebne podatke iz vanjskog API-ja, a zatim vraćaju odgovor nazad klijentu. U nastavku je prikazan programski kod krajnje točke API-ja za dohvaćanje podataka o videoigrama koji čita parametar naziva videoigre iz dolaznog zahtjeva te na temelju njega dohvaća podatke o videoigrama.

```
export async function GET(req) {
    const { searchParams } = new URL(req.url);

    try {
        const BASE_URL = process.env.NEXT_PUBLIC_CHEAP_SHARK_API_URL;

        let url = `${BASE_URL}/games?title=${searchParams.get("title")}`;

        const res = await fetch(url);

        if (!res.ok) {
            throw new Error("Dohvaćanje podataka videoigara neuspješno!");
        }

        const parsedData = await res.json();
        return NextResponse.json(parsedData);
    } catch (error) {
        console.error(error);
        return NextResponse.json(
            { error: "Interna pogreška kod dohvaćanja podataka videoigara!" },
            { status: 500 }
        );
    }
}
```

U nastavku je prikazan programski kod krajnje točke API-ja za dohvaćanje podataka o ponudama odabrane videoigre. Izdvaja se parametar ID-a videoigre iz zahtjeva te se gradi URL s ID-om videoigre za dohvaćanje ponuda.

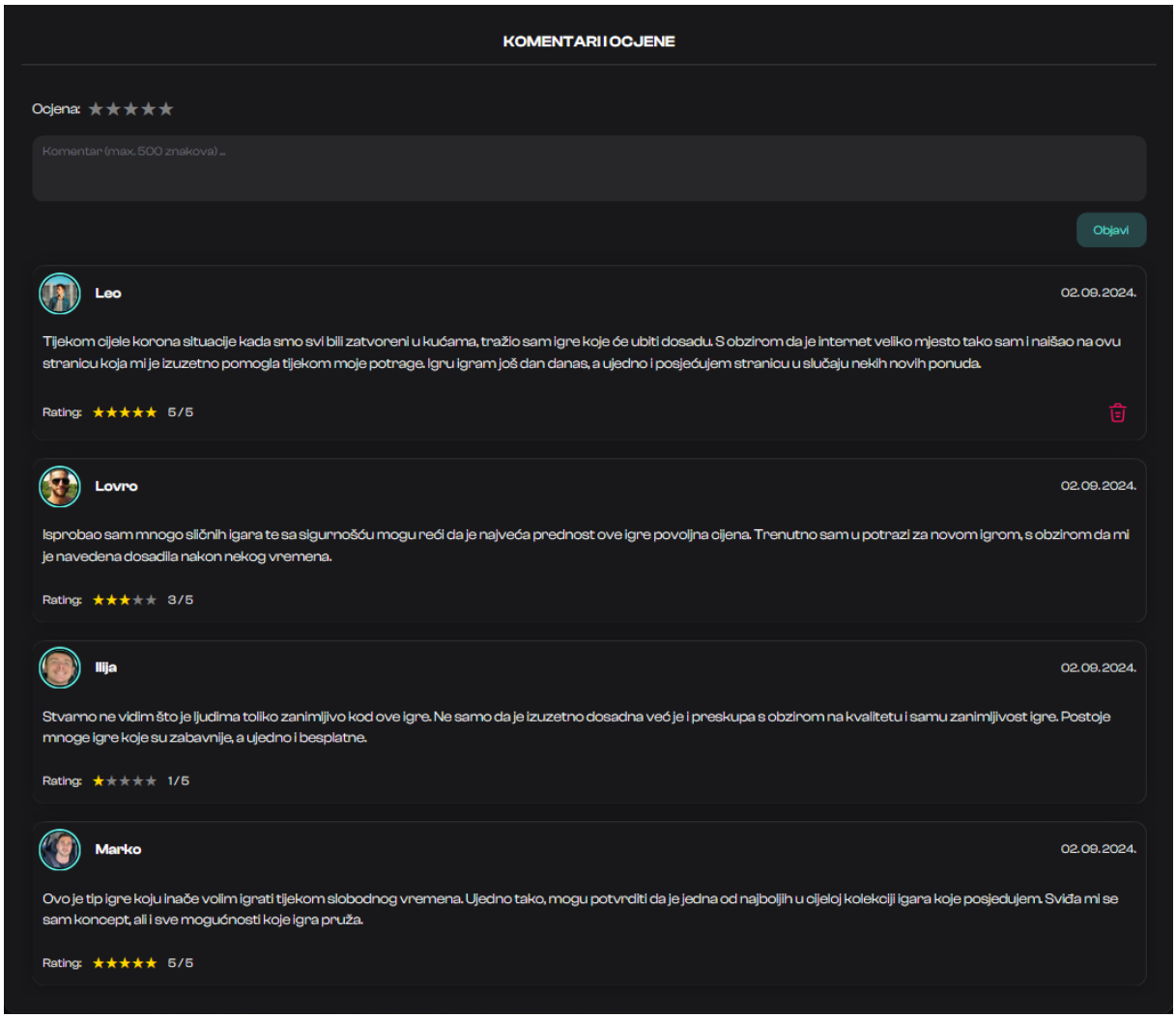
```
export async function GET(req, { params }) {
    try {
        const BASE_URL = process.env.NEXT_PUBLIC_CHEAP_SHARK_API_URL;
        const { gameID } = params;
```

```
        if (!gameID) {
            return NextResponse.json(
                { error: "ID videoigre nije definiran!" },
                { status: 400 }
            );
        }

        const url = `${BASE_URL}/games?id=${gameID}`;

        try {
            const res = await fetch(url);
            if (!res.ok) {
                throw new Error("Dohvaćanje podataka videoigre neuspješno!");
            }

            const parsedData = await res.json();
            return NextResponse.json(parsedData);
        } catch (error) {
            return NextResponse.json({ error: error.message }, { status: 500 });
        }
    } catch (error) {
        console.error(error);
        return NextResponse.json(
            { error: "Interna pogreška kod dohvaćanja podataka videoigre!" },
            { status: 500 }
        );
    }
}
```

Nakon što klijent primi podatke od poslužitelja oni se nadalje obrađuju, šalju komponentama i prikazuju korisnicima.

7.5.5. Objava komentara i ocjena

Komentari i ocjene prikazani su na stranici s detaljima ponude te su povezani za videoigru na koju se ponuda odnosi. Svi korisnici mogu pregledati komentare i ocjene drugih korisnika s time da isključivo prijavljeni korisnici imaju mogućnost objavljivanja komentara i ocjena. Sljedeća slika prikazuje dio s komentarima koji se nalazi na stranici s detaljima ponude.



Slika 9: Sekcija komentara (Izvor: vlastita izrada)

Prilikom objave komentara i ocjene provjerava se je li korisnik unio komentar i odredio ocjenu videoigre. U ovisnosti na rezultat provjere korisnik će se obavijestiti o pogrešci ili o uspješnoj objavi komentara i ocjene.

U nastavku je prikazan programski kod funkcije na klijentskoj strani koji u prvom dijelu dohvaća videoigre i platforme koje postoje u bazi podataka. U drugom dijelu poziva se funkcija koja se aktivira pritiskom na gumb za objavu komentara koja provjerava je li unesen komentar i ocjena te nakon toga provjerava postojanje videoigre i platforme u bazi podataka. U slučaju nepostojanja videoigre ili platforme ona će biti dodana u bazu podataka prije objave komentara.

```
useEffect(() => {
    const fetchData = async () => {
        try {
            const games = await getGames();
            const stores = await getStores();
            setExistingGames(games);
            setExistingStores(stores);
        } catch (error) { console.error( "Pogreška prilikom dohvaćanja podataka
            o videoigrama i platformama iz baze:", error); }
    };
```

```
        fetchData();
    }, []);

    const handleSubmit = async () => {
        if (!rating) {
            notifyError("Niste unijeli ocjenu!");
        } else if (!message) {
            notifyError("Niste unijeli komentar!");
        } else {
            const dateNow = new Date().toISOString().slice(0, 10);
            try {
                const gameExists = existingGames.some((game) => game.ID === Number(
                    gameID));

                if (!gameExists) {
                    const gameData = await fetchGameID(gameID);
                    const storesData = JSON.parse(localStorage.getItem("stores"));
                    const storeID = gameData.deals[0].storeID;
                    const storeData = storesData[storeID];
                    const storeExists = existingStores.some((store) => store.id ===
                        storeID);

                    if (!storeExists) {await addStore(storeID, storeData);}
                    await addGame(gameID, gameData);
                }

                const commentData = {
                    text: message,
                    rating,
                    date: dateNow,
                    gameID,
                    userID: session.user.id,
                };

                await addComment(commentData);
                const updatedComments = await getComments(gameID);

                setComments(updatedComments);
                setMessage("");
                notifySuccess("Komentar uspješno dodan!");
            } catch (error) {
                console.error(
                    "Pogreška prilikom objavljivanja komentara i ocjene videoigre:",
                    error
                );}}};
```

API ruta na strani poslužitelja prima zahtjev i izvlači podatke o komentaru iz zahtjeva. Obavlja se provjera postojanja sesije korisnika kako bi se provjerilo je li korisnik prijavljen te se provjerava postoje li svi podaci potrebni za unos komentara kao i odgovara li podatak ID-a korisnika ID-u prijavljenog korisnika. Na kraju se s pomoću funkcije `addComment` iz klase `CommentDAO` kreira novi komentar u bazi podataka i šalje odgovor nazad klijentu. U nastavku je prikazan programski kod krajnje točke API-ja za dodavanje komentara.

```
export async function POST(req) {
    const session = await getServerSession(authOptions);
    if (!session) { return NextResponse.json( { error: "Pristup zabranjen!" }, {
        status: 401 } );}

    try {
        const commentDAO = new CommentDAO();
        const commentData = await req.json();
        const missingField = !commentData.text
            ? "Tekst komentara nije definiran!"
            : !commentData.rating
            ? "Ocjena videoigre nije definirana!"
            : !commentData.date
            ? "Datum postavljanja komentara nije definiran!"
            : !commentData.gameID
            ? "ID videoigre nije definiran!"
            : !commentData.userID
            ? "ID korisnika nije definiran!"
            : null;

        if (commentData.userID != session.user.id) {
            return NextResponse.json(
                { error: "Nedovoljno prava!" },
                { status: 403 }
            );
        }

        if (missingField) { return NextResponse.json({ error: missingField }, {
            status: 400 }); }
        await commentDAO.addComment(commentData);

        return NextResponse.json(
            { message: "Komentar uspješno dodan!" },
            { status: 201 }
        );
    } catch (error) {
        const errorCode =
            error.code === "ER_DUP_ENTRY"
                ? "Komentar je već dodan!"
                : error.code === "ER_TRUNCATED_WRONG_VALUE"
                ? "Vrijednost nekog polja nije valjana!"
                : error.code === "ER_NO_REFERENCED_ROW_2"
                ? "ID videoigre ne postoji!"
                : null;

        if (errorCode) { return NextResponse.json({ error: errorCode }, { status:
            400 }); }

        console.error("Interna pogreška prilikom dodavanja komentara:", error);
        return NextResponse.json(
            { error: "Interna pogreška prilikom dodavanja komentara!" },
            { status: 500 }
        );}}
```

U nastavku je prikazan programski kod funkcije `addComment`.

```
async addComment(commentData) {
    const query =
        "INSERT INTO comments (text, rating, date, game_id, user_id) VALUES (?, ?,
            ?, ?, ?)";
    const values = [
        commentData.text,
        commentData.rating,
        commentData.date,
        commentData.gameID,
        commentData.userID,
    ];
    const comment = await pool.query(query, values);
    return comment;
}
```

7.5.6. Dodavanje videoigre na listu želja

Prijavljenom se korisniku nudi mogućnost dodavanja videoigara na listu želja te se dodane videoigre prikazuju na posebnoj stranici. Korisnici mogu dodati videoigru pritiskom na gumb s ikonom srca koji se nalazi na stranici s detaljima ponude te na stranici pretrage videoigre na temelju naziva. Korisniku se također nudi mogućnost uklanjanja videoigre s liste želja ponovnim pritiskom na gumb. Sljedeća slika prikazuje gumb za dodavanje videoigre na listu želja.

Slika 10: Gumb za dodavanje videoigre na listu želja (Izvor: vlastita izrada)

U nastavku je prikazan programski kod funkcije na klijentskoj strani koja se poziva prilikom pritiska na gumb. Prvo se provjerava postojanje sesije kako bi se utvrdilo je li korisnik prijavljen te se nakon toga provjerava postojanje videoigre i platforme u bazi podataka. U slučaju nepostojanja videoigre ili platforme ona će biti dodana u bazu podataka prije dodavanja videoigre na listu želja.

```
const handleWishListAdd = async () => {
    if (!session) {
        notifyError("Morate biti prijavljeni kako biste dodali videoigru na listu
        želja!");
        return;
    }

    try {
        const games = await getGames();
```

```
        const stores = await getStores();

        try {
            const gameExists = games.some(
                (game) => game.ID === Number(gameID)
            );

            if (!gameExists) {
                const gameData = await fetchGameID(gameID);
                const storesData = JSON.parse(
                    localStorage.getItem("stores")
                );

                const storeID = gameData.deals[0].storeID;
                const storeData = storesData[storeID];

                const storeExists = stores.some(
                    (store) => store.id === Number(storeID)
                );

                if (!storeExists) {
                    await addStore(storeID, storeData);
                }

                await addGame(gameID, gameData);
            }

            await addToWishlist(gameID, session.user.id);
            setIsWishlisted(true);
            notifySuccess("Videoigra dodana na listu želja!");
        } catch (error) {
            notifyError("Neuspješno dodavanje videoigre na listu želja!");
            console.error(
                "Pogreška prilikom dodavanja videoigre na listu želja:",
                error
            );
        }
    } catch (error) {
        console.error(
            "Pogreška prilikom dohvaćanja podataka o videoigrama i platformama iz
                baze:",
            error
        );}};
```

API ruta na strani poslužitelja prima zahtjev i izvlači podatke o ID-u videoigre i ID-u korisnika. Obavlja se provjera postojanja sesije korisnika kako bi se provjerilo je li korisnik prijavljen te se provjerava postoje li svi podaci potrebni za dodavanje videoigre na listu želja kao i odgovara li podatak ID-a korisnika ID-u prijavljenog korisnika. Na kraju se s pomoću funkcije `addToWishlist` iz klase `WishlistDAO` dodaje videoigra na listu želja u bazi podataka i šalje odgovor nazad klijentu. U nastavku je prikazan programski kod krajnje točke API-ja za dodavanje videoigre na listu želja.

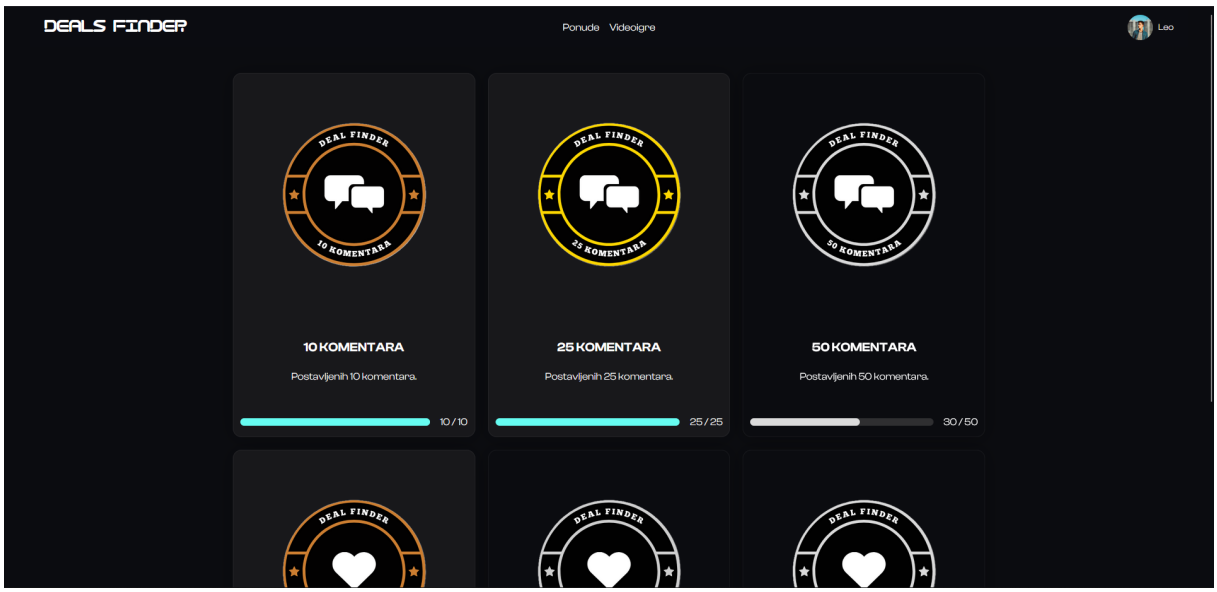
```
export async function POST(req) {
    const session = await getServerSession(authOptions);
    if (!session) {
        return NextResponse.json({ error: "Pristup zabranjen!" }, { status: 401 });
    }

    try {
        const wishlistDAO = new WishlistDAO();
        const { gameID, userID } = await req.json();
        const missingField = !gameID
            ? "ID videoigre nije definiran!"
            : !userID
            ? "ID korisnika nije definiran!"
            : null;

        if (session.user.id != userID && session.user.role != "admin") {
            return NextResponse.json(
                { error: "Nedovoljno prava!" },
                { status: 403 }
            );
        }

        if (missingField) {
            return NextResponse.json({ error: missingField }, { status: 400 });
        }

        await wishlistDAO.addToWishlist(gameID, userID);

        return NextResponse.json(
            { message: "Videoigra dodana na listu želja!" },
            { status: 201 }
        );
    } catch (error) {
        if (error.code === "ER_DUP_ENTRY") {
            return NextResponse.json(
                { message: "Videoigra je već dodana na listu želja!" },
                { status: 200 }
            );
        } else if (error.code === "ER_NO_REFERENCED_ROW_2") {
            return NextResponse.json(
                { message: "Videoigra i/ili korisnik ne postoji!" },
                { status: 400 }
            );
        }

        console.error("Interna pogreška prilikom dodavanja videoigre:", error);
        return NextResponse.json(
            { error: "Interna pogreška prilikom dodavanja videoigre!" },
            { status: 500 }
        );}}
```

U nastavku je prikazan programski kod funkcije `addToWishlist`.

```
async addToWishlist(gameID, userID) {
    const query = "INSERT INTO wishlist (game_id, user_id) VALUES (?, ?)";
    const values = [gameID, userID];
    const wishlistItem = await pool.query(query, values);
    return wishlistItem;
}
```

7.5.7. Pregled značaka korisnika

Registriranim korisnicima nudi se mogućnost sakupljanja određenih značaka u svrhu nagrađivanja njihove aktivnosti na web aplikaciji. Korisnici mogu prikupiti različite značke za objavljivanje komentara i dodavanje videoigra na listu želja. Sljedeća slika prikazuje stranicu sa značkama koje korisnik može sakupiti.



Slika 11: Stranica sa značkama (Izvor: vlastita izrada)

U nastavku je prikazan programski kod funkcije na klijentskoj strani. Prvo se provjerava postojanje sesije kako bi se utvrdilo je li korisnik prijavljen te se nakon toga dohvaćaju podaci o svim značkama koje korisnik može sakupiti, broj komentara korisnika i videoigara na listi želja korisnika te ostvarene značke od strane korisnika.

```
useEffect(() => {
    const fetchData = async () => {
        if (session) {
            try {
                const [
                    badgeData,
                    commentsNumber,
                    wishlistNumber,
                    fetchedUserBadges,
                ] = await Promise.all([
                    getBadges(),
```

```
                    getCommentsNumberByUserId(session.user.id),
                    getWishlistNumberByUserId(session.user.id),
                    getUserBadges(session.user.id),
                ]);

                const userProgressData = {
                    comments: commentsNumber.comments_number,
                    wishlist: wishlistNumber.wishlist_number,
                };

                setBadges(badgeData);
                setUserProgress(userProgressData);
                setUserBadges(fetchedUserBadges);
            } catch (error) {
                console.error("Dohvaćanje podataka neuspješno:", error);
            }
        }
    };

    fetchData();
}, [session]);
```

API rute na strani poslužitelja obrađuju dolazne zahtjeve od klijenta, dohvaćaju potrebne podatke iz baze podataka, a zatim vraćaju odgovor nazad klijentu. U nastavku je prikazan programski kod krajnje točke API-ja za dohvaćanje podataka o prikupljenim značkama korisnika koji čita parametar ID-a korisnika iz dolaznog zahtjeva. Parametar se prosljeđuje funkciji `getUserBadges` iz klase `UserBadgesDAO` za dohvaćanje korisnikovih značaka i šalje odgovor nazad klijentu. U nastavku je prikazan programski kod krajnje točke API-ja za dohvaćanje značaka korisnika.

```
export async function GET(req, { params }) {
    const session = await getServerSession(authOptions);

    if (!session) {
        return NextResponse.json(
            { error: "Pristup zabranjen!" },
            { status: 401 }
        );
    }

    try {
        const userBadgesDAO = new UserBadgesDAO();
        const userID = params.userID;

        if (session.user.id != userID && session.user.role != "admin") {
            return NextResponse.json(
                { error: "Nedovoljno prava!" },
                { status: 403 }
            );
        }

        const userBadges = await userBadgesDAO.getUserBadges(userID);
```

```
        return NextResponse.json(userBadges, { status: 200 });
    } catch (error) {
        console.error(
            "Interna pogreška prilikom dohvaćanja značaka korisnika:",
            error
        );
        return NextResponse.json(
            {
                error: "Interna pogreška prilikom dohvaćanja značaka korisnika!",
            },
            { status: 500 }
        );
    }
}
```

U nastavku je prikazan programski kod funkcije `getUserBadges`.

```
async getUserBadges(userID) {
    const query = "SELECT * FROM user_badges WHERE user_id = ?";
    const [badges] = await pool.query(query, [userID]);
    return badges;
}
```

Nakon što klijent primi podatke od poslužitelja oni se nadalje obrađuju, šalju komponentama i prikazuju korisniku.

7.5.8. Kritički osvrt

Razvoj web aplikacije korištenjem biblioteke React i okvira Next.js predstavljalo je jedno novo iskustvo i izazov s obzirom na to da se radilo o prvom susretu s tim tehnologijama. Mogla se primijetiti lakoća učenja i korištenja u odnosu na programski okvir Angular. Korištenjem React biblioteke i Next.js okvira, ali i ostalih biblioteka i modula koji su korišteni za izradu web aplikacije, primijećena je znatna lakoća izrade same web aplikacije kao i vrijeme koje je potrebno za izradu u odnosu na izradu web aplikacija korištenjem isključivo HTML-a, CSS-a i JavaScripta.

Korištenjem biblioteke NextUI znatno se olakšao i ubrzao proces razvoja korisničkog sučelja aplikacije s obzirom na to da biblioteka pruža veliku količinu već gotovih komponenata koje je samo potrebno prilagoditi zahtjevima aplikacije.

Cjelokupni razvoj aplikacije bio je izazovan, svaki dio je nosio svoje izazove i probleme, koje je bilo potrebno riješiti te je teško izdvojiti najzahtjevniji dio razvijene aplikacije. Može se istaknuti razvoj sučelja, odnosno dizajna stranice s prikazom detalja ponude koji je zasigurno pružio najviše izazova i problema tijekom razvoja rasporeda svih komponenata za navedenu stranicu.

Uočena je lakoća prijenosa određenih podataka i informacija komponentama korištenjem Reactove vezne točke `useState` koja izrazito olakšava komunikacija svih komponenata s roditeljskom komponentom. Također je uočena lakoća implementacije osvježavanja informacija sučelja aplikacije korištenjem `useEffect` bez potrebe za osvježavanjem cijele stranice što znatno poboljšava korisničko iskustvo.

Struktura Next.js aplikacije omogućuje jednostavnost snalaženja i mogućnost izrade dobro organizirane aplikacije što predstavlja veliki značaj prilikom kretanja kroz datoteke aplikacije. Prepoznata je vrijednost načina izrade putanja koje se definiraju kreiranjem direktorija unutar direktorija **app** čime je uklonjena potreba definiranja putanja unutar koda.

Uz React i Next.js, korištenje Tailwind CSS-a koji omogućuje stiliziranje elemenata kroz klase, u cijelosti je uklonilo potrebu korištenja CSS datoteka čime se proces stiliziranja komponenata znatno olakšava i pojednostavljuje.

Zaključno, uočena je vrijednost korištenja ovih tehnologija, a ostale mogućnosti i značajke koje ove tehnologije pružaju, a nisu korištene prilikom izrade web aplikacije, zasigurno mogu samo učvrstiti njihovu vrijednost.

8. Zaključak

U ovom radu opisan je proces razvoja web aplikacije korištenjem React biblioteke, Next.js okvira i MySQL baze podataka. Cilj rada bio je pokazati na koji način moderne tehnologije za razvoj web aplikacija mogu biti iskorištene za razvoj skalabilnih, responzivnih i funkcionalnih aplikacija koje ispunjavaju zahtjeve korisnika.

Tijekom procesa istraživanja i razvoja aplikacije, primijećene su brojne koristi korištenja ovih tehnologija. React je dokazao svoju vrijednost kao korisna biblioteka za stvaranje interaktivnih korisničkih sučelja zbog svoje komponentne arhitekture i virtualnog DOM-a, što rezultira brzim i efikasnim kreiranjem kompleksnih sučelja. Next.js je dodatno poboljšao razvoj omogućujući renderiranje na poslužiteljskoj strani i statičko generiranje stranica, čime su optimizirane performanse i poboljšano korisničko iskustvo, kao i SEO optimizacija. MySQL baza podataka osigurava stabilno i efikasno upravljanje podacima, pružajući sigurnu i brzu pohranu te dohvat informacija.

Kroz izradu web aplikacije, demonstrirana je praktična primjena teorijskih koncepta tehnologija koje su korištene. Uspješno su implementirane funkcionalnosti kao što su registracija korisnika, prikazivanje i pretraživanje ponuda videoigara, i interakcija korisnika kroz komentare i ocjene. Sve ove funkcionalnosti pokazuju na koji način se moderne tehnologije mogu primijeniti za razvoj složenih web aplikacija.

Zaključno, ovaj rad pruža korisne informacije o modernim tehnologijama za razvoj web aplikacija i pokazuje njihovu važnost kroz praktični primjer.

Popis literature

[1] B. Getting, *Basic Definitions: Web 1.0, Web. 2.0, Web 3.0*, travanj 2007. adresa: https://www.practicalecommerce.com/Basic-Definitions-Web-1-0-Web-2-0-Web-3-0.

[2] M. Kariyawasam, *The Evolution of the Web: How Application Development Changed as a Result.* Lipanj 2020. adresa: https://medium.com/@mehani_kariyawasam/the-evolution-of-the-web-how-application-development-changed-as-a-result-84354e6349bf.

[3] G. Kappel, B. Pröll, S. Reich i W. Retschitzegger, *Web Engineering*. John Wiley & Sons Ltd., 2006.

[4] GeeksforGeeks, *What is Web App?* Siječanj 2024. adresa: https://www.geeksforgeeks.org/what-is-web-app/.

[5] Karol, *Web Development Technology: How to Choose the Best One*, listopad 2020. adresa: https://blurify.com/blog/web-development-technology-how-to-choose-the-best-one/.

[6] D. R. Brooks, *An Introduction to HTML and JavaScript*, 1. izdanje. Springer London, 2007.

[7] J. C. Meloni i J. Kyrnin, *HTML, CSS, and JavaScript All in One*. Sams Publishing, 2012.

[8] D. Flanagan, *JavaScript: The Definitive Guide*, 6. izdanje. O'Reilly Media, Inc, 2011.

[9] Stackoverflow, *Web frameworks and technologies*, 2023. adresa: https://survey.stackoverflow.co/2023/#technology.

[10] J. Potter, *Downloads over time of JavaScript front-end frameworks*, 2024. adresa: https://npmtrends.com/@angular/core-vs-lit-vs-react-vs-solid-js-vs-svelte-vs-vue.

[11] Google, *Interest over time of JavaScript back-end frameworks*, 2024. adresa: https://trends.google.com/trends/explore?cat=31&q=React%20javascript,Vue%20javascript,Angular%20javascript,Svelte%20javascript,Lit%20javascript.

[12] M. Joshi, *Angular vs React vs Vue: Core Differences*, svibanj 2023. adresa: https://www.browserstack.com/guide/angular-vs-react-vs-vue.

[13] Google, *Interest over time of JavaScript back-end frameworks*, 2024. adresa: https://trends.google.com/trends/explore?cat=31&q=%2Fm%2F0bbxf89,Laravel,Flask,ASP.NET%20CORE,Spring%20Boot.

[14] B. Basumatary i N. Agnihotri, „Benefits and Challenges of Using NodeJS," *International Journal of Innovative Research in Computer Science & Technology*, svibanj 2022.

[15] M. Deery, *What Is Flask and How Do Developers Use It? A Quick Guide*, kolovoz 2023. adresa: https://careerfoundry.com/en/blog/web-development/what-is-flask/.

[16] Z. Deineko, S. Sotnik, O. Vovk i V. Lyashenko, „Features of Database Types," *International Journal of Engineering and Information Systems (IJEAIS)*, 2021.

[17] DB-Engines, *DB-Engines Ranking*, srpanj 2024. adresa: https://db-engines.com/en/ranking.

[18] M. Drake i ostezer, „SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems," *DigitalOcean*, ožujak 2022. adresa: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

[19] A. Fedosejev, *React.js Essentials*. Packt Publishing Ltd, 2015.

[20] C. Gackenheimer, *Introduction to React*. Apress, 2015.

[21] React, *Built-in React Hooks*, 2024. adresa: https://react.dev/reference/react/hooks.

[22] React, *Start a New React Project*, 2024. adresa: https://react.dev/learn/start-a-new-react-project.

[23] M. Fariz, S. Lazuardy i D. Anggraini, „Modern Front End Web Architectures with React.Js and Next.Js," *International Research Journal of Advanced Engineering and Science*, sv. 7, 2022.

[24] Vercel, *Next.js Docs*, 2024. adresa: https://nextjs.org/docs.

[25] Vercel, *Data Fetching, Caching and Revalidating*, 2024. adresa: https://nextjs.org/docs/app/building-your-application/data-fetching/fetching-caching-and-revalidating.

[26] Vercel, *Next.js Docs - Styling*, 2024. adresa: https://nextjs.org/docs/app/building-your-application/styling.

[27] Vercel, *Next.js Docs - Optimizations*, 2024. adresa: https://nextjs.org/docs/app/building-your-application/optimizing.

[28] Vercel, *Next.js Docs - TypeScript*, 2024. adresa: https://nextjs.org/docs/app/building-your-application/configuring/typescript.

[29] Oracle, *The Main Features of MySQL*, 2024. adresa: https://dev.mysql.com/doc/refman/8.4/en/features.html.

[30] V. M. Grippa i S. Kuzmichev, *Learning MySQL*. O'Reilly Media, Inc., 2021.

Popis slika

Popis tablica

1. Prilog 1 - programski kod web aplikacije

Programski kod implementirane web aplikacije dostupan je na sustavu FOI radovi u zip arhivi.