

UNIVERZITET U KRAGUJEVCU
FAKULTET TEHNIČKIH NAUKA U ČAČKU

Uroš Petrović

**Razvoj veb aplikacije za pametno naručivanje
putem QR kodova
korišćenjem modernih tehnologija**

Diplomski rad

Čačak, oktobar 2025.

UNIVERZITET U KRAGUJEVCU
FAKULTET TEHNIČKIH NAUKA U ČAČKU



**Razvoj veb aplikacije za pametno naručivanje
putem QR kodova
korišćenjem modernih tehnologija**

Diplomski rad

Vrsta studija: Osnovne akademske studije

Naziv studijskog programa: Informacione Tehnologije

Predmet: Internet Programiranje

Profesor:

Dr Željko Jovanović

Student:

Uroš Petrović 10-2021

Čačak, oktobar 2025.

Rezime

Ovaj rad opisuje proces razvoja veb aplikacije za pametno naručivanje pod nazivom "QrCode-Smart-Order". Aplikacija omogućava korisnicima da putem skeniranja QR koda pregledaju meni i naruče proizvode bez direktnog kontakta sa osobljem.

U radu je detaljno objašnjen celokupan proces izrade, od postavljanja projektne strukture do finalne implementacije. Korišćene su moderne tehnologije kao što su Next.js za full-stack razvoj, Prisma kao ORM za rad sa bazom podataka, Tailwind CSS za stilizovanje, React za korisnički interfejs, JWT za autentifikaciju i Docker za kontejnerizaciju aplikacije.

Cilj rada je da prikaže praktičnu primenu ovih tehnologija na realnom primeru i objasni prednosti njihovog korišćenja u modernom veb razvoju.

Ključne reči:

- QR Kod
- React
- Nextjs
- Tailwind
- Tech Stack
- ORM
- JWT
- MySQL
- Monolit

Summary

This paper describes the development process of a web application for smart ordering called "QRCode-Smart-Order". The application allows users to browse the menu and order products by scanning a QR code, without direct contact with the staff.

The paper explains in detail the entire development process, from setting up the project structure to the final implementation. Modern technologies were used, such as Next.js for full-stack development, Prisma as an ORM for working with the database, Tailwind CSS for styling, React for the user interface, JWT for authentication, and Docker for containerization of the application.

The aim of the paper is to demonstrate the practical application of these technologies in a real-world example and to explain the advantages of their use in modern web development.

Keywords:

- QR Code
- React
- Nextjs
- Tailwind
- Tech Stack
- ORM
- JWT
- MySQL
- Monolith

Sadržaj

1	Uvod.....	5
2	Digitalni Identifikatori.....	6
2.1	<i>Alati i jezici za rad sa QR kodovima.....</i>	<i>8</i>
2.2	<i>Data Matrix.....</i>	<i>9</i>
2.3	<i>Aztec Code.....</i>	<i>10</i>
2.4	<i>Portable Data File 417.....</i>	<i>11</i>
2.5	<i>MaxiCode.....</i>	<i>12</i>
2.6	<i>Napredni Jednodimenzionalni (1D) Barkodovi.....</i>	<i>13</i>
2.6.1	<i>Code 39.....</i>	<i>13</i>
2.6.2	<i>Code 128.....</i>	<i>13</i>
2.6.3	<i>Codabar.....</i>	<i>14</i>
2.7	<i>Programerska Podrška za Rad sa Digitalnim Identifikatorima.....</i>	<i>15</i>
2.7.1	<i>Generisanje Kodova.....</i>	<i>15</i>
2.7.2	<i>Čitanje i Skeniranje Kodova.....</i>	<i>16</i>
3	Analiza i izbor razvojnih tehnologija.....	17
3.1	<i>Next.js – Frejmvork za budućnost.....</i>	<i>17</i>
3.2	<i>Prisma – Rad sa bazom podataka na novom nivou.....</i>	<i>18</i>
3.3	<i>Tailwind CSS – Brzo i efikasno stilizovanje.....</i>	<i>18</i>
3.4	<i>JWT – Bezbedna autentifikacija.....</i>	<i>19</i>
3.5	<i>Docker – Konzistentno okruženje.....</i>	<i>19</i>
4	Praktična Demonstracija.....	20
4.1	<i>Opis sistema.....</i>	<i>20</i>
4.1.1	<i>Use Case Dijagram.....</i>	<i>21</i>
4.2	<i>Postavljanje Projekta.....</i>	<i>22</i>
4.3	<i>Definisanje šeme baze podataka.....</i>	<i>23</i>
4.4	<i>Backend – API Rute – Kontroleri i Servisi.....</i>	<i>24</i>
4.5	<i>Frontend – Korisnički interfejs.....</i>	<i>25</i>
4.5.1	<i>Početna strana.....</i>	<i>26</i>
4.5.2	<i>Menu sa preporukama.....</i>	<i>27</i>
4.5.3	<i>Poručivanje.....</i>	<i>29</i>
4.5.4	<i>Panel za radnike.....</i>	<i>30</i>
4.5.5	<i>Registracija i prijava radnika.....</i>	<i>32</i>
4.5.6	<i>QrKod Generisanje.....</i>	<i>36</i>
4.6	<i>Povezivanje celine – Docker Compose.....</i>	<i>38</i>
5	Struktura projekta (tree-view).....	39
6	Zaključak.....	40
7	Literatura.....	41

1 Uvod

U današnjem digitalnom dobu, potreba za brzim i efikasnim rešenjima u ugostiteljstvu je sve veća. Tradicionalni načini naručivanja, koji se oslanjaju na fizičke menije i direktnu komunikaciju sa osobljem, često mogu biti spori, podložni ljudskim greškama i neefikasni tokom perioda visoke posećenosti. Kao odgovor na ove izazove, razvijena je veb aplikacija "QRCode-Smart-Order". Ona rešava ovaj problem tako što omogućava gostima da putem svojih pametnih telefona, jednostavnim skeniranjem QR koda, trenutno pristupe digitalnom meniju i izvrše narudžbinu bez čekanja i direktnog kontakta sa osobljem.

Ovaj projekat je realizovan sa ciljem da se istraže i primene moderne veb tehnologije koje omogućavaju brz i efikasan razvoj, kao i kreiranje skalabilnih i održivih rešenja. Kroz ovaj rad, biće detaljno prikazan celokupan proces izrade aplikacije, od idejnog rešenja i arhitekture, preko izbora tehnologija, do finalne implementacije i postavljanja. Poseban fokus stavljen je na tehnologije koje čine jezgro aplikacije:

- **Next.js:** React frejmwork koji je odabran zbog svoje sposobnosti za full-stack razvoj, omogućavajući istovremenu izgradnju korisničkog interfejsa i serverske logike unutar jedinstvenog projekta. Njegov hibridni pristup renderovanju (SSR i SSG) i jednostavno kreiranje API ruta čine ga idealnim za ovakav tip aplikacije. [1] [2]
- **Prisma:** Moderan ORM (Object-Relational Mapping) koji pojednostavljuje i osigurava interakciju sa bazom podataka kroz tipski bezbedan klijent, čime se smanjuje broj potencijalnih grešaka u radu sa podacima. [3]
- **Tailwind CSS:** "Utility-first" CSS frejmwork koji drastično ubrzava proces stilizovanja korisničkog interfejsa omogućavajući razvoj direktno unutar HTML strukture. [4]
- **React:** JavaScript biblioteka za kreiranje korisničkih interfejsa, čija komponentna arhitektura omogućava razvoj modularnih i lako održivih delova aplikacije [5] [6]
- **JWT (JSON Web Tokens):** Standard za bezbednu i stateless autentifikaciju korisnika, ključan za zaštitu ruta i podataka na serveru. [7]
- **Docker:** Platforma za kontejnerizaciju koja omogućava pakovanje aplikacije i svih njenih zavisnosti u izolovane kontejnere, čime se garantuje konzistentno i predvidivo okruženje za razvoj i produkciju. [8]

Cilj rada je da prikaže praktičnu primenu ovih tehnologija na realnom primeru i objasni prednosti njihovog korišćenja u modernom veb razvoju.

2 Digitalni Identifikatori

U svetu digitalne transformacije, sposobnost brzog i pouzdanog povezivanja fizičkog sveta sa digitalnim informacijama je od suštinskog značaja. Digitalni identifikatori, kao što su barkodovi i QR kodovi, predstavljaju optičke tehnologije koje služe upravo toj svrsi. Iako obavljaju sličnu funkciju, postoje fundamentalne razlike u njihovoj strukturi, kapacitetu i primeni.

1. **Barkod (Bar Code):** Barkod je jednodimenzionalni (1D) optički kod koji se sastoji od niza paralelnih linija i razmaka različitih širina. Podaci se kodiraju u širini i rasporedu ovih linija, a čitaju se pomoću specijalizovanih skenera koji mere refleksiju svetlosti. Zbog svoje linearne prirode, barkodovi imaju relativno mali kapacitet i obično skladište kratke alfanumeričke nizove, najčešće identifikacione brojeve proizvoda (npr. EAN i UPC kodovi u maloprodaji).

Glavne karakteristike barkodova su:

- **Jednostavnost:** Lako se generišu i štampaju.
- **Nizak kapacitet:** Mogu da uskladište samo malu količinu podataka (obično do 20-25 karaktera).
- **Jednosmerno skeniranje:** Zahtevaju skeniranje pod određenim uglom, paralelno sa linijama koda.
- Iako su i dalje nezamenljivi u oblastima kao što su trgovina i upravljanje zalihama, njihova ograničenja su dovela do razvoja naprednijih, dvodimenzionalnih kodova.



Slika 1 Primer barkoda

2. **QR Kod (Quick Response Code):** QR kod je dvodimenzionalni (2D) matrični kod koji je razvila japanska kompanija Denso Wave. Za razliku od barkoda, podaci se kodiraju u dva smjera (horizontalno i vertikalno) unutar matrice crnih i belih kvadrata. Ova struktura omogućava QR kodovima da skladište znatno veću količinu podataka i različite tipove informacija.

Ključne prednosti QR kodova u odnosu na barkodove su:

- **Visok kapacitet podataka:** Mogu da uskladište hiljade alfanumeričkih karaktera, omogućavajući kodiranje URL adresa, kontakt informacija (vCard), Wi-Fi pristupnih podataka, ili običnog teksta.
- **Sistem za korekciju grešaka (Error Correction):** QR kodovi poseduju ugrađeni Reed-Solomon sistem za korekciju grešaka. To im omogućava da budu čitljivi čak i ako je do 30% njihove površine oštećeno ili prekriveno.
- **Omnidirekciono skeniranje:** Zahvaljujući markerima za pozicioniranje (tri veća kvadrata u uglovima), QR kodovi se mogu brzo skenirati iz bilo kog ugla.
- **Direktna interakcija putem pametnih telefona:** Uspon pametnih telefona sa ugrađenim kamerama učinio je QR kodove izuzetno popularnim, jer ne zahtevaju specijalizovane skenere.
- U kontekstu projekta "QrCode-Smart-Order", QR kod služi kao ključna spona između fizičkog prostora (stola u restoranu) i digitalne platforme. Skeniranjem koda, korisnik trenutno pristupa meniju i platformi za naručivanje, čime se proces automatizuje i pojednostavljuje.



Slika 2 QrCode u vidu pretrage na internetu



Slika 3 QrCode ka website-u

2.1 Alati i jezici za rad sa QR kodovima

Rad sa QR kodovima u softverskom razvoju obuhvata dve osnovne operacije: generisanje i čitanje (skeniranje).

1. Generisanje QR kodova

Generisanje QR kodova se vrši programski, korišćenjem specijalizovanih biblioteka. Za veb aplikacije razvijene u JavaScript ekosistemu, kao što je i ova, postoji veliki broj rešenja. Neke od popularnih biblioteka su:

- **qrcode.react**: React komponenta koja na jednostavan način generiše QR kodove u SVG ili Canvas formatu, što je idealno za frontend aplikacije.
- **node-qrcode**: Moćna biblioteka za Node.js okruženje, koja omogućava generisanje QR kodova na serverskoj strani i njihovo čuvanje kao slike (PNG, SVG) ili predstavljanje kao Base64 string.

U ovom projektu, generisanje QR kodova bi se obavljalo u administrativnom delu aplikacije, gde bi za svaki sto ili lokaciju bio generisan jedinstveni QR kod koji sadrži URL do digitalnog menija za tu specifičnu lokaciju.

2. Čitanje i skeniranje QR kodova:

Proces čitanja i skeniranja QR kodova je u potpunosti izmešten iz same veb aplikacije "QrCode-Smart-Order" i oslanja se na native mogućnosti pametnih telefona korisnika. Aplikacija nije zadužena za implementaciju skenera, već samo za generisanje kodova i prihvatanje saobraćaja koji iz njih proističe.

Korisnički tok je sledeći:

1. Ugostiteljski objekat štampa jedinstvene QR kodove generisane kroz aplikaciju i postavlja ih na odgovarajuće stolove.
2. Gost koji želi da naruči koristi svoj pametni telefon i otvara podrazumevanu aplikaciju za kameru.
3. Usmeravanjem kamere ka QR kodu, operativni sistem telefona (iOS ili Android) automatski prepoznaje kod i dekodira URL adresu koja je u njemu sadržana.
4. Na ekranu telefona se pojavljuje obaveštenje koje korisniku nudi da otvori datu veb adresu u internet pregledaču.
5. Klikom na obaveštenje, pokreće se internet pregledač i učitava se stranica aplikacije "QrCode-Smart-Order" koja je specifična za sto sa kojeg je kod skeniran.

Ovakav pristup značajno pojednostavljuje razvoj aplikacije, jer eliminiše potrebu za implementacijom kompleksne logike za pristup kameri i obradu video signala unutar pregledača. Umesto toga, oslanja se na visoko optimizovane i pouzdane alate koji su već integrisani u sve moderne pametne telefone.

2.2 Data Matrix

Data Matrix je dvodimenzionalni matrični kod koji se sastoji od crnih i belih ćelija, odnosno tačaka, raspoređenih u kvadratni ili pravougaoni uzorak. Prepoznatljiv je po "L" obliku na jednoj strani koda, koji se naziva "Finder Pattern", i koji služi za određivanje orijentacije i gustine matrice prilikom skeniranja.

Ključne karakteristike:

- **Visoka gustina podataka:** Data Matrix može da uskladišti veliku količinu podataka na izuzetno maloj površini. U stanju je da kodira do 2,335 alfanumeričkih karaktera. Zbog ove karakteristike, idealan je za označavanje veoma malih predmeta, kao što su elektronske komponente, hirurški instrumenti i farmaceutski proizvodi.
- **Izuzetna otpornost na oštećenja:** Kao i QR kod, Data Matrix koristi Reed-Solomon algoritam za korekciju grešaka. Može biti uspešno očitano čak i ako je do 40% površine koda oštećeno, što ga čini pouzdanim za primenu u industrijskim okruženjima gde su oštećenja česta.
- **Skalabilnost:** Veličina Data Matrix koda se može prilagoditi potrebama, od minijature kodova veličine nekoliko kvadratnih milimetara do velikih kodova za logističke potrebe.
- **Čitljivost pod niskim kontrastom:** Efikasno se čita čak i pri slabom osvetljenju ili kada je kontrast između crnih i belih ćelija smanjen.

Primena:

Njegova primarna primena je u industriji, posebno u **proizvodnji, logistici, farmaceutskoj industriji i zdravstvu**. Koristi se za praćenje proizvoda kroz lanac snabdevanja (track-and-trace), označavanje serijskih brojeva na komponentama i verifikaciju autentičnosti lekova.



Data Matrix Code



QR Code

Slika 4 Data Matrix

2.3 Aztec Code

Aztec kod je još jedan 2D matrični kod, nazvan po centralnom uzorku koji podseća na astečku piramidu gledanu odozgo. Ovaj centralni "Finder Pattern" omogućava skeneru da locira kod bez potrebe za zonom tišine (praznim prostorom) oko njega, što ga čini efikasnijim u pogledu iskorišćenosti prostora.

Ključne karakteristike:

- **Efikasnost prostora:** Nepostojanje zahteva za praznim prostorom oko koda znači da može zauzeti manju površinu na medijumu u poređenju sa QR kodom istog kapaciteta.
- **Visoka pouzdanost i korekcija grešaka:** Aztec kodovi nude podesiv nivo korekcije grešaka, od 5% do 95% kapaciteta. To im omogućava da budu čitljivi čak i pri značajnim oštećenjima ili lošim uslovima štampe.
- **Dobar kapacitet:** Može da uskladišti do 3,832 numerička ili 3,067 alfanumeričkih karaktera.

Primena:

Aztec kodovi se najčešće koriste u **transportnoj industriji**. Mnoge železničke kompanije (npr. Eurostar, Deutsche Bahn) koriste ih na voznim kartama, jer se mogu brzo i pouzdano skenirati na mobilnim uređajima, čak i ako je ekran oštećen ili slabo osvetljen. Takođe se koriste u avio-industriji za bording karte i u nekim sistemima za naplatu putarine.



Slika 5 Aztec Code

2.4 Portable Data File 417

PDF417 (Portable Data File 417) je naslagani (stacked) linearni barkod. Iako pripada 2D klasi, njegova struktura podseća na više jednodimenzionalnih barkodova naslaganih jedan na drugi. Broj 417 u nazivu potiče iz njegove strukture: svaka reč koda (codeword) se sastoji od 4 trake i 4 razmaka, a svaka je dugačka 17 modula.

Ključne karakteristike:

- **Veliki kapacitet podataka:** PDF417 može da kodira preko 1.1 kilobajta mašinski čitljivih podataka, što ga čini "prenosivom datotekom podataka". Može skladištiti tekst, brojeve, fajlove, pa čak i slike.
- **Korekcija grešaka:** Posедуje ugrađen mehanizam za korekciju grešaka koji omogućava rekonstrukciju podataka čak i ako je deo koda uništen.
- **Zahteva skeniranje u jednom pravcu:** Za razliku od matričnih kodova (QR, Data Matrix), PDF417 se mora skenirati linearno, slično kao tradicionalni barkod, ali može tolerisati određeni ugao nagiba.

Primena:

Zbog sposobnosti da skladišti veliku količinu podataka, PDF417 se koristi u aplikacijama koje zahtevaju čuvanje kompletnih informacija unutar samog koda, bez potrebe za pristupom bazi podataka. Najpoznatija primena je na **ličnim dokumentima** (vozačke dozvole u mnogim državama SAD, lične karte, pasoši), **transportnim dokumentima** (FedEx ga koristi na paketima) i u **carinskim deklaracijama**.



Slika 6 PDF417

2.5 MaxiCode

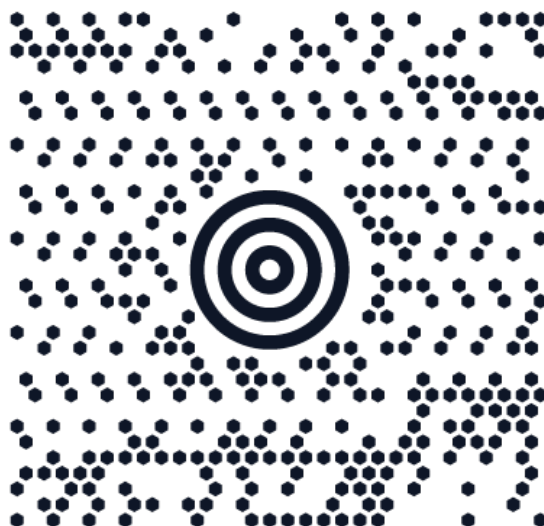
MaxiCode je 2D kod fiksne veličine, razvijen od strane kompanije UPS (United Parcel Service) za potrebe logistike i automatskog sortiranja paketa. Vizuelno je jedinstven, sa centralnim uzorkom koji podseća na metu za gađanje ("bullseye") okruženim heksagonalnom mrežom tačaka.

Ključne karakteristike:

- **Fiksna veličina:** Svaki MaxiCode je veličine približno 1x1 inč (2.54x2.54 cm).
- **Brzo skeniranje:** Dizajniran je za izuzetno brzo, omnidirekciono skeniranje na pokretnim trakama koje se kreću velikom brzinom (do 150 metara u minuti).
- **Ograničen kapacitet:** Može da uskladišti oko 93 karaktera podataka. Centralni deo koda obično sadrži ključne informacije za sortiranje, kao što su poštanski broj, zemlja i klasa usluge.
- **Visoka tolerancija na oštećenja:** Robustan je i može se očitati čak i ako je odštampan na površinama lošeg kvaliteta.

Primena:

MaxiCode je gotovo isključivo vezan za **logističku i transportnu industriju**, pre svega za automatsko sortiranje i praćenje pošiljki. Njegov dizajn je optimizovan za jednu svrhu – brzu i pouzdanu identifikaciju paketa u pokretu.



Slika 7 MaxiCode

2.6 Napredni Jednodimenzionalni (1D) Barkodovi

Iako 2D kodovi nude veći kapacitet i fleksibilnost, jednodimenzionalni barkodovi su i dalje dominantni u mnogim oblastima zbog svoje jednostavnosti, niske cene štampe i široke rasprostranjenosti postojećih skenera. Pored standardnih EAN i UPC kodova, u upotrebi su i simbologije koje nude veću fleksibilnost.

2.6.1 Code 39

Code 39 (ili Code 3 of 9) je jedan od prvih alfanumeričkih barkodova. Može da kodira velika slova (A-Z), brojeve (0-9) i nekoliko specijalnih karaktera (-, ., \$, /, +, %,). Svaki karakter je predstavljen sa 9 elemenata (5 traka i 4 razmaka), od kojih su 3 uvek široka (otuda naziv "3 od 9"). Zbog svoje jednostavnosti, ne zahteva složene algoritme za generisanje.

Primena:

Code 39 je i dalje popularan u **industriji, logistici i zdravstvu**. Koristi se za interne sisteme praćenja, označavanje imovine, inventar i identifikacione kartice. Zbog mogućnosti kodiranja slova, pogodan je za serijske brojeve i oznake delova.

Code 39 (REGULAR)



Slika 8 Code 39

2.6.2 Code 128

Code 128 je linearni barkod visoke gustine koji može da kodira svih 128 ASCII karaktera. Svoju efikasnost duguje mogućnosti dinamičkog prebacivanja između tri seta karaktera (Code Set A, B i C). Set C je posebno optimizovan za kodiranje numeričkih podataka, jer pakuje dve cifre u jedan simbol, čime se postiže izuzetna gustina.

Primena:

Zbog svoje kompaktnosti i svestranosti, Code 128 je jedan od najrasprostranjenijih 1D barkodova u **transportu i logistici**. Standard GS1-128 (ranije poznat kao UCC/EAN-128) se zasniva na Code 128 i koristi se za označavanje logističkih jedinica (paleta i kutija) podacima kao što su serijski broj, rok trajanja i težina.



Slika 9 Code 128

2.6.3 Codabar

Codabar je simbologija koja kodira numeričke cifre (0-9) i nekoliko specijalnih karaktera. Dizajniran je da bude lak za štampanje na standardnim štampačima, čak i onim sa nižom rezolucijom, i da ostane čitljiv. Svaki karakter je nezavisan i sadrži 4 trake i 3 razmaka.

Primena:

Iako je starija tehnologija, Codabar se i dalje koristi u specifičnim oblastima kao što su **biblioteke** (za označavanje knjiga), **banke krvi** (za praćenje uzoraka) i u nekim kurirskim službama (npr. FedEx za brojeve na otpremnicama).



Slika 10 Codabar

2.7 Programerska Podrška za Rad sa Digitalnim Identifikatorima

Rad sa digitalnim identifikatorima u softverskom razvoju, kao i u ovom projektu, podrazumeva dve ključne operacije: generisanje i čitanje/skeniranje kodova. Za obe operacije postoji širok spektar alata i biblioteka koje omogućavaju laku integraciju u veb, mobilne i desktop aplikacije.

2.7.1 Generisanje Kodova

Generisanje barkodova i 2D kodova se programski svodi na korišćenje biblioteka koje prihvataju ulazni tekst (npr. URL, serijski broj) i generišu sliku (PNG, SVG) ili vektorski format koda.

JavaScript Ekosistem (Veb i Node.js):

- **qrcode.react** i **node-qrcode**: Kao što je već pomenuto u radu, ove biblioteke su specijalizovane za QR kodove i široko su prihvaćene.
- **bwip-js**: Izuzetno moćna i svestrana biblioteka koja podržava generisanje preko 100 različitih tipova 1D i 2D kodova, uključujući QR Code, Data Matrix, Aztec, PDF417, Code 128 i mnoge druge. Može se koristiti i na klijentskoj (u pregledaču) i na serverskoj strani (Node.js).
- **jsBarcode**: Popularna biblioteka fokusirana na generisanje 1D barkodova (Code 128, Code 39, EAN, UPC, itd.) u SVG, Canvas ili IMG formatu.

Python Ekosistem:

- **python-barcode**: Biblioteka za generisanje 1D barkodova. Jednostavna je za korišćenje i podržava izlaz u SVG formatu.
- **qrcode**: Najpopularnija Python biblioteka za generisanje QR kodova.
- **treepoem**: Omotač (wrapper) oko open-source projekta BWIPP (Barcode Writer in Pure PostScript), koji omogućava generisanje velikog broja različitih simbologija, slično kao bwip-js.

2.7.2 Čitanje i Skeniranje Kodova

Čitanje kodova je kompleksniji proces koji uključuje obradu slike ili video strima kako bi se detektovao i dekodirao kod.

- **Nativne Mogućnosti Mobilnih Uređaja:** Kao što je objašnjeno u kontekstu ovog projekta, najjednostavniji pristup za korisnike je oslanjanje na ugrađene skenere u aplikacijama za kameru na iOS i Android platformama. Ove platforme obično prepoznaju najčešće tipove kodova kao što su QR, Data Matrix i Aztec.

Biblioteke za Veb Aplikacije (JavaScript):

- **zxing-js/library:** JavaScript port popularne Java biblioteke ZXing ("Zebra Crossing"). To je svestrana biblioteka koja može da dekodira veliki broj 1D i 2D kodova direktno u pregledaču, koristeći kameru uređaja (getUserMedia API).
- **scandit-sdk-for-the-web:** Komercijalno rešenje koje nudi superiorne performanse skeniranja, čak i u lošim uslovima (oštećeni kodovi, slabo osvetljenje, skeniranje pod uglom). Podržava ogroman broj simbologija.

Biblioteke za Serversku Obradu i Desktop Aplikacije:

- **pyzbar:** Python biblioteka koja predstavlja omotač oko popularne ZBar biblioteke napisane u C-u. Može da dekodira različite tipove kodova iz slika.
- **pylibdmtx:** Python biblioteka specifično namenjena za čitanje Data Matrix kodova.
- **ZXing (Java, C++):** Originalna i jedna od najrobusnijih open-source biblioteka za obradu barkodova. Mnoge druge biblioteke su zasnovane na njoj ili su njeni portovi na druge jezike.

Oslanjanje na moderne biblioteke apstrahuje kompleksnost koja stoji iza algoritama za kodiranje i dekodiranje, omogućavajući programerima da se fokusiraju na poslovnu logiku aplikacije, kao što je to bio slučaj i u razvoju "QRCode-Smart-Order" sistema.

3 Analiza i izbor razvojnih tehnologija

Izbor tehnologija je fundamentalan korak u razvoju svakog softverskog rešenja. Za projekat "QRCode-Smart-Order", odabrane su tehnologije koje predstavljaju moderne standarde u veb razvoju, sa ciljem postizanja efikasnosti, skalabilnosti i visokih performansi.

3.1 Next.js – Frejmwork za budućnost

Next.js je izabran kao osnova projekta zbog svojih brojnih prednosti koje ga čine idealnim za full-stack razvoj:

- **Hibridno renderovanje:** Next.js nudi mogućnost kombinovanja Server-Side Rendering (SSR) i Static Site Generation (SSG). Za ovu aplikaciju, stranice menija koje se ne menjaju često mogu biti generisane statički (SSG) prilikom build-a aplikacije, što omogućava njihovo trenutno učitavanje. Sa druge strane, stranice koje prikazuju status narudžbine ili personalizovane podatke korisnika mogu koristiti SSR za dinamičko generisanje sadržaja na serveru pri svakom zahtevu. Ova fleksibilnost osigurava optimalne performanse i odličnu SEO optimizaciju. [1]
- **Integrisane API Rute:** Mogućnost kreiranja backend logike unutar app/api direktorijuma eliminiše potrebu za odvojenim serverskim aplikacijama i složenim konfiguracijama. Ovo pojednostavljuje arhitekturu, čineći je monolitnom, i olakšava upravljanje celokupnim kodom na jednom mestu. [1]
- **Brz razvoj (Developer Experience):** Funkcionalnosti kao što su Fast Refresh, koji omogućava trenutni prikaz promena u kodu bez gubljenja stanja komponenti, i automatsko rutiranje zasnovano na fajl sistemu, značajno ubrzavaju i olakšavaju razvojni proces.

The image shows the Next.js logo, which consists of the word "NEXT" in a bold, black, sans-serif font, followed by ".JS" in a smaller, black, sans-serif font.

Slika 11 Next.js

3.2 Prisma – Rad sa bazom podataka na novom nivou

Prisma je odabrana kao ORM (Object-Relational Mapping) alat kako bi se pojednostavila i osigurala komunikacija sa MySQL bazom podataka. Ključne prednosti Prisme su:

- **Tipizirane šeme (Type Safety):** Definicija celokupne šeme baze podataka nalazi se u jednom, čitljivom `schema.prisma` fajlu. Na osnovu ove šeme, Prisma automatski generiše tipove za TypeScript, što znači da je svaka interakcija sa bazom podataka tipski bezbedna. Ovo eliminiše čitavu klasu grešaka (npr. pogrešno ime kolone) tokom razvoja. [3]
- **Prisma Client:** Auto-generisani i intuitivni klijent pruža moćan API za rad sa podacima, čineći upite jednostavnim i čitljivim. Umesto pisanja sirovih SQL upita, developer koristi metode kao što su `prisma.product.findMany()` ili `prisma.order.create()`. [3]
- **Migracije:** Prisma Migrate je alat koji automatski prati promene u `schema.prisma` fajlu i generiše SQL migracione fajlove. Ovo olakšava evoluciju šeme baze podataka na konzistentan i predvidiv način kroz ceo životni ciklus projekta. [3]



Slika 12 Prisma ORM

3.3 Tailwind CSS – Brzo i efikasno stilizovanje

Za stilizovanje je korišćen Tailwind CSS iz sledećih razloga:

- **Utility-First:** Umesto pisanja posebnih CSS fajlova i klasa, stilovi se primenjuju direktno u HTML-u (JSX-u) korišćenjem predefinisanih, kompozitnih klasa (npr. `flex`, `pt-4`, `text-center`). Ovakav pristup drastično ubrzava proces stilizovanja jer eliminiše potrebu za prebacivanjem između HTML i CSS fajlova. [4]
- **Visoka prilagodljivost:** Iako dolazi sa bogatim setom predefinisanih klasa, Tailwind se lako može konfigurisati i proširiti kroz `tailwind.config.js` fajl, omogućavajući kreiranje jedinstvenog dizajnerskog sistema koji odgovara brendu aplikacije.
- **Optimizacija performansi:** Tokom procesa izgradnje (build) za produkciju, Tailwind automatski skenira sve fajlove i uklanja sve neiskorišćene stilove, što rezultira najmanjim mogućim CSS fajlom i bržim učitavanjem stranica. [4]



Slika 13 Tailwind CSS

3.4 JWT – Bezbedna autentifikacija

Autentifikacija korisnika je ključan deo aplikacije, a rešena je pomoću JSON Web Tokens (JWT). Prednosti JWT-a su:

- **Stateless (bez stanja):** Server ne mora da čuva informacije o sesiji korisnika. Svaki JWT sadrži sve potrebne informacije unutar sebe. Kada klijent pošalje zahtev sa validnim tokenom, server može da verifikuje njegov potpis i odobri pristup bez potrebe za upitom u bazu podataka, što olakšava horizontalno skaliranje aplikacije. [7]
- **Bezbednost:** Tokeni su digitalno potpisani korišćenjem tajnog ključa (secret), što garantuje da podaci unutar tokena nisu menjani i da potiču iz pouzdanog izvora.
- **Fleksibilnost:** JWT je otvoreni standard i može se koristiti za autentifikaciju na različitim platformama, što aplikaciju čini spremnom za buduća proširenja, kao što je razvoj mobilne aplikacije koja bi koristila isti backend.



Slika 14 JWT Token

3.5 Docker – Konzistentno okruženje

Da bi se osigurala konzistentnost, prenosivost i jednostavna instalacija, aplikacija je kontejnerizovana pomoću Dockera i Docker Compose-a. Glavne prednosti ovog pristupa su:

- **Izolovano okruženje:** Svaki deo sistema, kao što su Next.js aplikacija i MySQL baza podataka, izvršava se u svom sopstvenom, izolovanom kontejneru. Ovo sprečava konflikte između zavisnosti i osigurava da aplikacija radi na isti način bez obzira na to gde se pokreće. [8]
- **Pojednostavila instalacija:** Zahvaljujući **docker-compose.yml** fajlu, pokretanje celokupnog projekta – uključujući bazu podataka i aplikaciju – svodi se na izvršavanje jedne komande: **docker-compose up**. Ovo drastično olakšava postavljanje razvojnog okruženja za nove članove tima. [8]
- **Konzistentnost između okruženja:** Docker garantuje da će se ista verzija softvera, biblioteka i zavisnosti koristiti u razvojnog, testnog i produkcionog okruženju, čime se eliminišu problemi tipa "ali na mojoj mašini radi".



Slika 15 Docker

4 Praktična Demonstracija

Proces implementacije aplikacije "QRCode-Smart-Order" podeljen je u nekoliko logičkih faza, od inicijalnog postavljanja projekta do finalnog povezivanja svih delova u funkcionalnu celinu.

4.1 Opis sistema

Veb aplikacija "QRCode-Smart-Order" je softversko rešenje razvijeno sa ciljem da modernizuje i optimizuje proces naručivanja u ugostiteljskim objektima. Sistem rešava probleme tradicionalnog naručivanja, kao što su sporost i neefikasnost, tako što gostima omogućava da putem svojih pametnih telefona pristupe digitalnom meniju i izvrše narudžbinu skeniranjem QR koda. Aplikacija je dizajnirana da opslužuje dve osnovne grupe korisnika sa jasno definisanim ulogama i funkcionalnostima.

Korisnici Sistema

Sistem predviđa dve osnovne vrste korisnika (aktera):

1. **Gost:** Predstavlja klijenta u ugostiteljskom objektu. Gost ne mora da kreira nalog niti da se prijavljuje na sistem. Njegova interakcija sa aplikacijom je direktna i počinje skeniranjem QR koda postavljenog na stolu.
2. **Radnik:** Predstavlja zaposleno osoblje (konobare, šankere, menadžere) koje koristi administrativni deo aplikacije za prijem i upravljanje porudžbinama. Za pristup ovom delu sistema neophodna je registracija i prijava.

Opis Funkcionalnosti

Sistem je podeljen na dva dela, u skladu sa tipovima korisnika: interfejs za goste i panel za radnike.

Funkcionalnosti za Goste:

Interakcija gosta sa sistemom je dizajnirana da bude brza, intuitivna i ne zahteva preuzimanje dodatnih aplikacija. Proces se odvija na sledeći način:

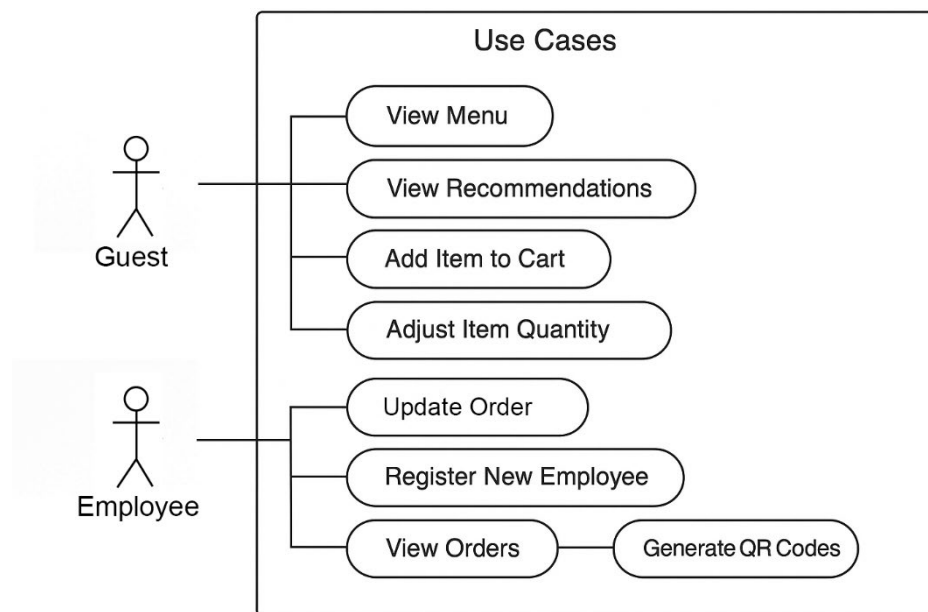
- **Pristup Meniju putem QR Koda:** Gost koristi kameru svog pametnog telefona da skenira jedinstveni QR kod na stolu. Operativni sistem telefona prepoznaje kod i nudi otvaranje veb adrese u internet pregledaču, koja vodi do stranice menija specifične za taj sto.
- **Pregled Menija i Preporuka:** Po otvaranju stranice, gostu se prikazuje digitalni meni sa svim dostupnim artiklima. Svaki artikal sadrži opis, cenu i fotografiju. Poseban deo menija ističe preporučene proizvode, koje sistem predlaže na osnovu prethodnih porudžbina sa tog stola.
- **Kreiranje i Slanje Porudžbine:** Korisnik može jednostavno dodati željene proizvode u korpu. U korpi je moguće pregledati sve odabrane stavke, prilagoditi njihove količine i videti ukupan iznos za plaćanje. Potvrdom porudžbine, ona se automatski prosleđuje osoblju restorana u realnom vremenu.

Funkcionalnosti za Radnike:

Panel za radnike je centralni deo sistema za upravljanje operacijama i dostupan je samo autorizovanom osoblju.

- **Autentifikacija i Autorizacija:** Da bi pristupili panelu, radnici se moraju prijaviti sa svojim korisničkim imenom i lozinkom. Sistem koristi JSON Web Tokens (JWT) za bezbednu i *stateless* autentifikaciju, čime se osigurava da samo ovlašćeni korisnici mogu pristupiti zaštićenim rutama i podacima.
- **Registracija Novih Radnika:** Aplikacija omogućava registrovanim radnicima da kreiraju naloge za novo osoblje.
- **Upravljanje Porudžbinama:** Glavna funkcionalnost panela je prikaz svih pristiglih narudžbina u realnom vremenu. Za svaku narudžbinu prikazan je broj stola, spisak naručenih artikala, količine i ukupna cena. Radnici mogu da prate i ažuriraju status svake porudžbine kroz faze: "PENDING" (na čekanju), "IN_PROGRESS" (u pripremi), "COMPLETED" (završeno) i "CANCELED" (otkazano).
- **Generisanje QR Kodova:** U sklopu aplikacije postoji i sekcija za generisanje jedinstvenih QR kodova za svaki sto u objektu.

4.1.1 Use Case Dijagram



Slika 16 Use Case Dijagram

4.2 Postavljanje Projekta

Prvenstveno se kreira Next.js projekat korišćenjem komande: `npx create-next-app`. Nakon osnovnog postavljanja, projektu su dodate neophodne zavisnosti putem npm (Node Package Manager). To uključuje Prisma klijent i CLI za interakciju sa bazom podataka, Tailwind CSS sa svojim zavisnostima (postcss, autoprefixer) za stilizovanje, i biblioteke za rad sa JWT (jsonwebtoken) i validaciju (zod). [1]

```
PS C:\Users\K\Documents\Github\QrCode-Smart-Order\testdir> npx create-next-app
✓ What is your project named? ... qrcodeapp
✓ Would you like to use TypeScript? ... No / Yes
✓ Which linter would you like to use? » ESLint
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like your code inside a `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to use Turbopack? (recommended) ... No / Yes
✓ Would you like to customize the import alias (`@/*` by default)? ... No / Yes
Creating a new Next.js app in C:\Users\K\Documents\Github\QrCode-Smart-Order\testdir\qrcodeapp.

Using npm.

Initializing project with template: app-tw

Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- @tailwindcss/postcss
- tailwindcss
- eslint
- eslint-config-next
- @eslint/eslintrc

added 331 packages, and audited 332 packages in 24s

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

found 0 vulnerabilities
Success! Created qrcodeapp at C:\Users\K\Documents\Github\QrCode-Smart-Order\testdir\qrcodeapp
```

Slika 17 Inicijalizacija next.js projekta

4.3 Definisanje šeme baze podataka

Srce interakcije sa podacima leži u Prisma šemi. U fajlu **prisma/schema.prisma** definisani su svi modeli podataka koji su potrebni aplikaciji. Definisani su modeli:

- User (za korisnike/osoblje)
- Table (broj stola)
- Product (za stavke u meniju)
- Order (porudžbina)
- OrderItem (product koji je dodat u porudžbinu)
- Recommendation (preporuke na strain stola na osnovu broja porudžbina za tim stolom)
- OrderStatus (za praćenje i izmenu statusa porudžbine)

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model User {
  id          Int      @id @default(autoincrement())
  name        String?
  username    String?  @unique
  password    String?
  createdAt   DateTime @default(now())
}

model Table {
  id          Int      @id @default(autoincrement())
  number      Int      @unique
  qrCodeUrl   String
  createdAt   DateTime @default(now())
  orders      Order[]
  Recommendation Recommendation[]
}

model Product {
  id          Int      @id @default(autoincrement())
  name        String
  description  String?
  imageUrl    String?  @db.LongText
  price       Decimal  @db.Decimal(10, 2)
  isActive    Boolean  @default(true)
  createdAt   DateTime @default(now())
  orderItems  OrderItem[]
  Recommendation Recommendation[]
}

model Order {
  id          Int      @id @default(autoincrement())
  tableId     Int
  table       Table    @relation(fields: [tableId], references: [id])
  status      OrderStatus @default(PENDING)
  createdAt   DateTime @default(now())
  items       OrderItem[]
}

model OrderItem {
  id          Int      @id @default(autoincrement())
  orderId     Int
  productId   Int
  quantity    Int      @default(1)
  order       Order    @relation(fields: [orderId], references: [id])
  product     Product  @relation(fields: [productId], references: [id])
}

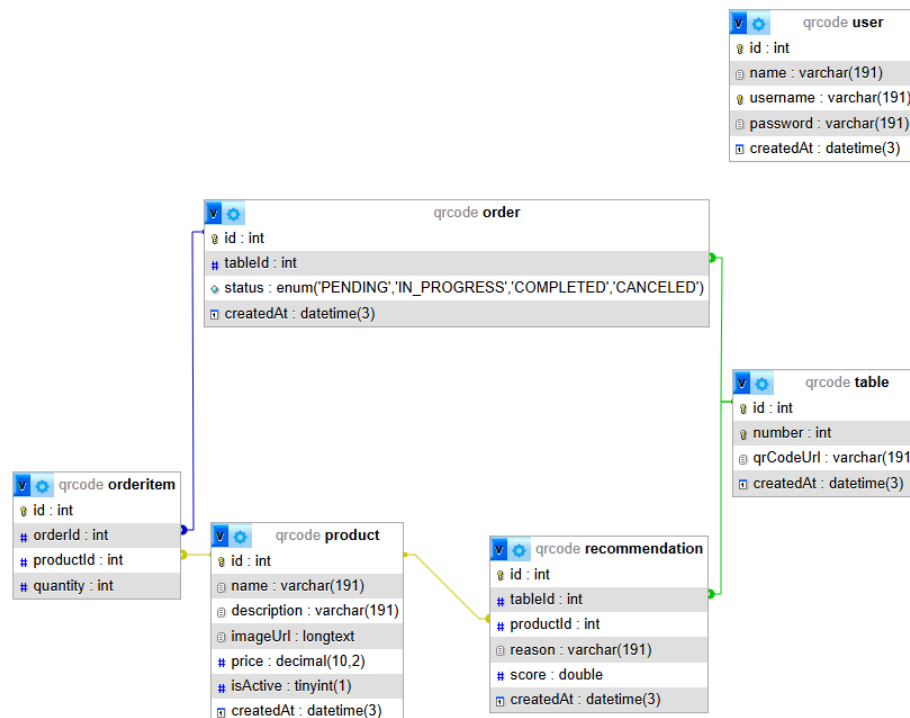
model Recommendation {
  id          Int      @id @default(autoincrement())
  tableId     Int?
  table       Table?   @relation(fields: [tableId], references: [id])
  productId   Int
  product     Product  @relation(fields: [productId], references: [id])
  reason      String
  score       Float
  createdAt   DateTime @default(now())
}

enum OrderStatus {
  PENDING
  IN_PROGRESS
  COMPLETED
  CANCELED
}
```

Slika 18 Prisma Schema

Nakon što je šema definisana, izvršava se migracija komandom **prisma migrate dev**.

Ova komanda je automatski generisala SQL kod i primenila ga na MySQL bazu podataka, kreirajući sve potrebne tabele i relacije.



Slika 19 Dijagram baze

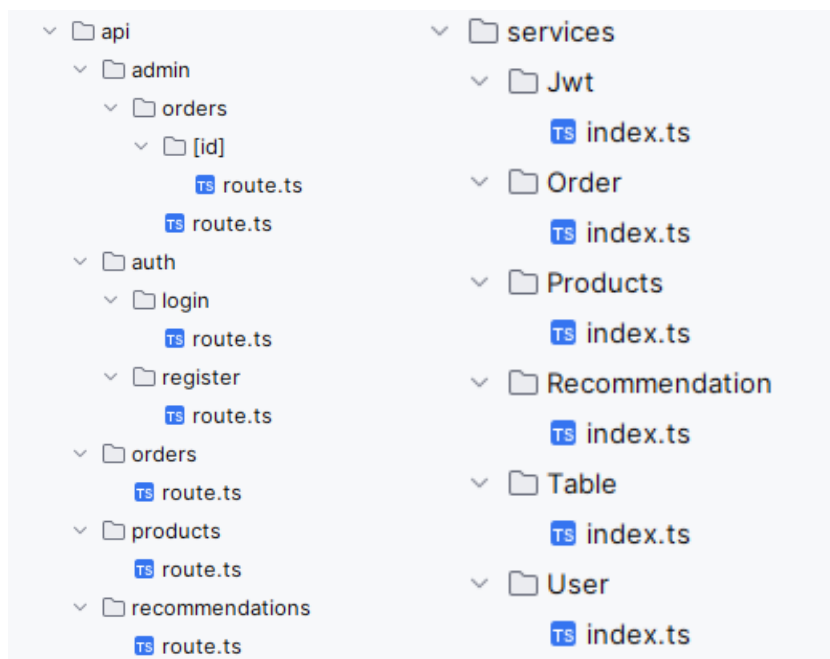
4.4 Backend – API Rute – Kontroleri i Servisi

Backend logika je smeštena unutar **src/app/api** direktorijuma, prateći strukturu zasnovanu na Next.js API rutama. Za bolju organizaciju koda, primenjen je princip razdvajanja odgovornosti, gde fajlovi ruta deluju kao kontroleri, a poslovna logika je izdvojena u servisne fajlove.

- **Rute (Kontroleri):** Svaka API ruta (npr. `src/app/api/products/route.ts`) odgovorna je za prihvatanje HTTP zahteva (GET, POST, PUT, DELETE), validaciju ulaznih podataka i slanje odgovarajućeg odgovora klijentu.
- **Servisi:** Servisni fajlovi (npr. `src/services/Products/index.ts`) sadrže čistu poslovnu logiku i direktno komuniciraju sa bazom podataka putem Prisma klijenta.

Kreirane su rute za sve ključne funkcionalnosti:

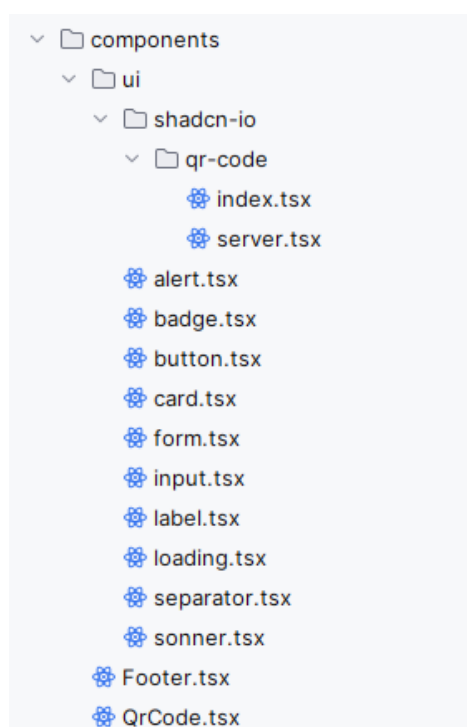
- **Autentifikaciju:** Rute register i login za registraciju i prijavu korisnika, koje generišu JWT nakon uspešne autentifikacije.
- **CRUD operacije:** Kompletne CRUD (Create, Read, Update, Delete) operacije za upravljanje proizvodima, korisnicima i narudžbinama.



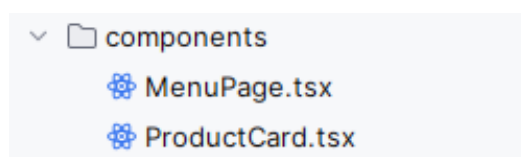
Slika 20 Next.js API rute/Kontroleri i Servisi

4.5 Frontend – Korisnički interfejs

Korisnički interfejs je izgrađen kao skup React komponenata smeštenih u **components** direktorijum. Arhitektura frontenda je modularna i zasnovana na principu komponenti za ponovnu upotrebu.



Slika 21 Glavni folder komponenata

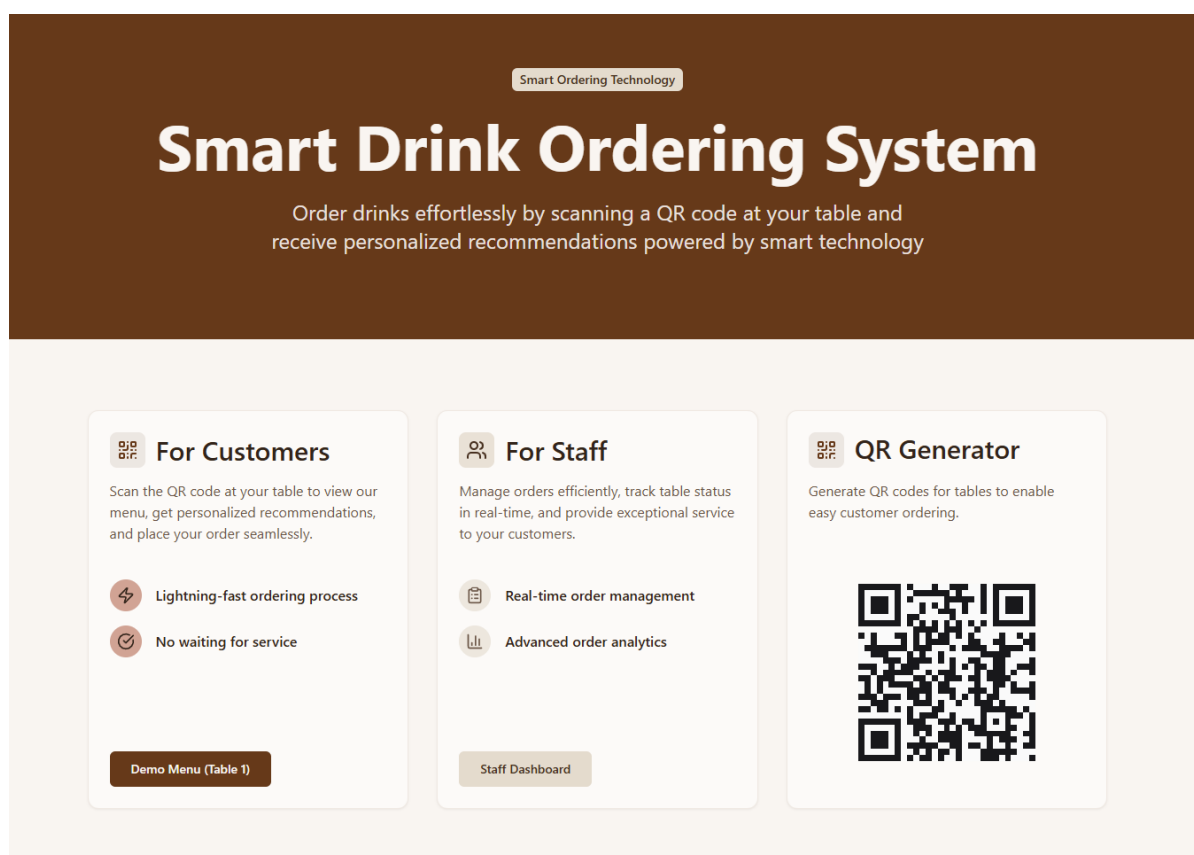


Slika 22 Menu Page komponente

4.5.1 Početna strana

Početna stranica aplikacije sastoji se iz tri osnovne sekcije:

- U sekciji *Menu* korisnici mogu da pregledaju celokupnu ponudu pića, kao i da izvrše porudžbinu direktno putem stranice.
- *Panel za radnike* namenjen je osoblju restorana i omogućava pristup alatima za praćenje narudžbina i upravljanje stolovima.
- Treća sekcija, *QR kod*, služi za jednostavno povezivanje korisnika sa određenim stolom u restoranu, čime se olakšava proces poručivanja i praćenja statusa narudžbine.



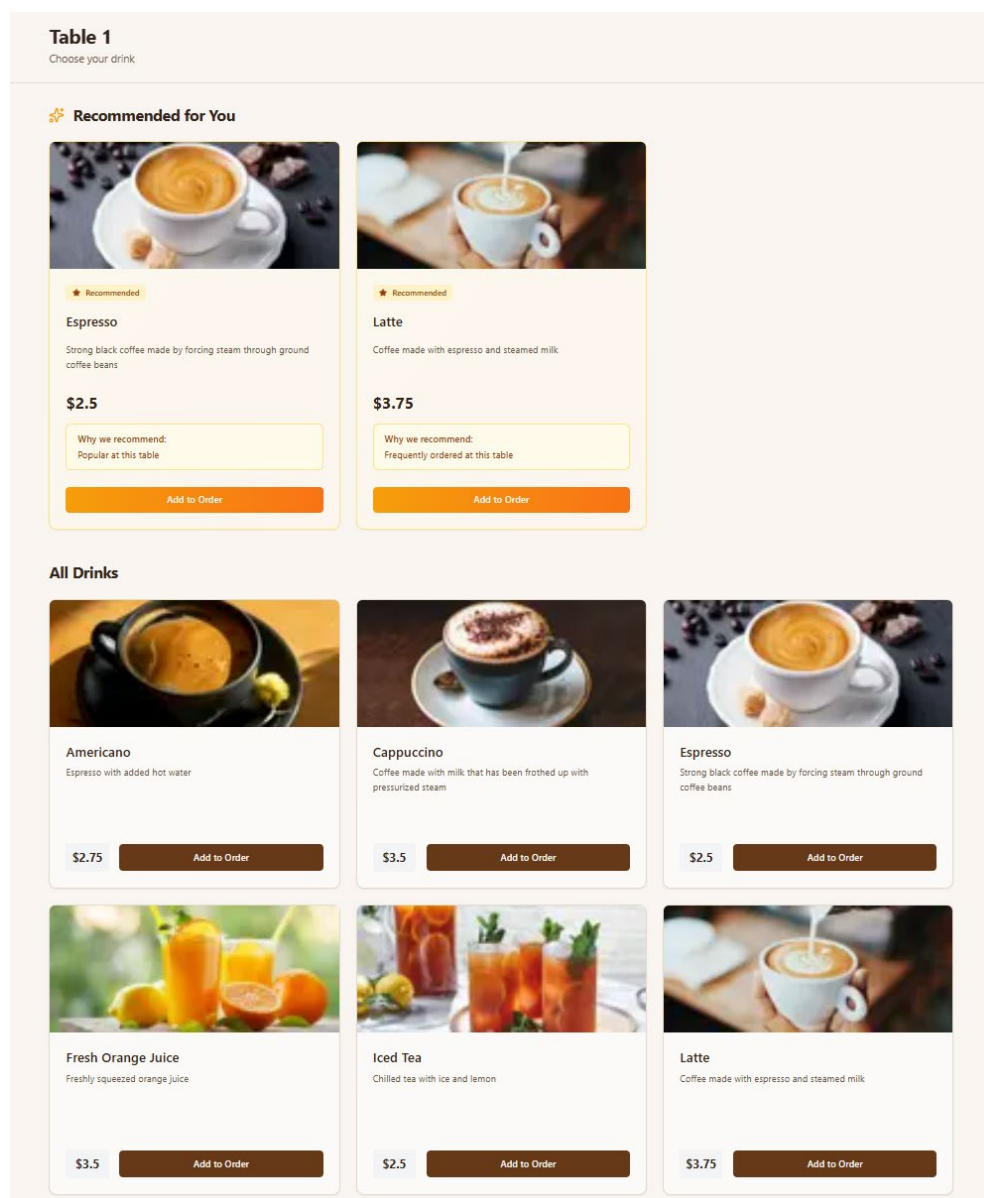
Slika 23 Početna strana

4.5.2 Menu sa preporukama

Menu predstavlja centralni deo aplikacije i sadrži **preporuke** kao i **spisak dostupnih artikala** koji se mogu poručiti.

U ovom odeljku korisnici mogu da pregledaju kompletan izbor pića. Sekcija sa **preporukama** ističe najpopularnije proizvode, koje sistem predlažu na osnovu prethodnih porudžbina.

Pored toga, svaki artikal sadrži osnovne informacije — **opis, cenu i fotografiju**, a korisnici mogu brzo dodati željene proizvode u korpu i završiti porudžbinu direktno iz menija.



Slika 24 Prikaz menija

```

export default function MenuPage() : Element { Show usages  ⚙ Uros Petrovic
  const searchParams : ReadonlyURLSearchParams = useSearchParams();
  const tableId : string | null = searchParams.get( 'name: "table"' );
  const { products, recommendations, loading, error } = useOptions(tableId);
  const {
    cart,
    setCart,
    addToCart,
    updateCartItemQuantity,
    removeFromCart,
    submitOrder,
    cartLoading,
    cartError,
  } = useCart();

  const handleSubmitOrder : () => Promise<void> = async () : Promise<void> => { Show usages  ⚙ Uros Petrovic
    try {
      const payload = {
        tableId,
        items: cart.map((item : CartItemType ) : { productId: number; quantity: number } => ({
          productId: item.productId,
          quantity: item.quantity,
        })),
      };
      await submitOrder(payload);
      setCart([]);
    } catch (error) {
      console.error(error);
    }
  };

  if (loading) {
    return (
      <div className="h-screen flex items-center justify-center">
        <Loading size="lg" text="Please wait..." />
      </div>
    );
  }

  if (error || cartError)
    throw new Error( message: 'Error creating order, ${error || cartError}' );

  if (!tableId) {
    return (
      <div className="flex flex-col items-center justify-center min-h-screen p-4">
        <Alert className="max-w-md">
          <Clock className="h-4 w-4" />
          <AlertDescription className="text-center">
            <div className="font-semibold mb-2">Table Not Found</div> Please
            scan a valid QR code to access your table#39;s menu.
          </AlertDescription>
        </Alert>
      </div>
    );
  }
}

```

Slika 25 Menu stranica – funkcionalnost

Komponenta `MenuPage` predstavlja centralni deo korisničkog interfejsa aplikacije koja omogućava pregled menija i kreiranje porudžbine. Kod je napisan u React biblioteci, korišćenjem modernih React Hook-ova i TypeScript tipizacije, čime se obezbeđuje robusnost i čitljivost.

1. Učitavanje podataka i osnovna inicijalizacija

Na početku komponente koristi se React Hook `useSearchParams()` kako bi se iz URL adrese preuzeo parametar koji sadrži identifikator stola (`tableId`). Ovaj identifikator služi za povezivanje korisnika sa tačno određenim stolom u restoranu, čime se omogućava personalizovano prikazivanje menija i porudžbina.

Zatim se poziva prilagođeni hook `useOptions(tableId)`, koji vraća neophodne podatke za prikaz menija, kao što su:

- lista proizvoda (`products`),
- preporuke (`recommendations`),
- status učitavanja (`loading`),
- eventualne greške (`error`).

Na ovaj način komponenta dobija sve podatke koji su joj potrebni za dalje funkcionisanje.

2. Upravljanje korpom i porudžbinama

Hook useOptions obezbeđuje i funkcije za rad sa korpom:

- cart – trenutni sadržaj korpe,
- setCart – postavlja novi sadržaj korpe,
- addToCart, removeFromCart, updateCartItemQuantity – manipulacija artiklima u korpi,
- submitOrder – funkcija za slanje porudžbine na server.

Na ovaj način komponenta u potpunosti kontroliše proces dodavanja proizvoda, menjanja količina i konačnog slanja narudžbine.

3. Slanje porudžbine

Centralni deo logike je funkcija handleSubmitOrder, koja se izvršava kada korisnik potvrdi narudžbinu.

Funkcija formira objekat (payload) koji sadrži:

- ID stola (tableId),
- listu proizvoda iz korpe, gde je za svaki proizvod navedeno njegovo productId i quantity.

Nakon uspešnog slanja, korpa se prazni pozivom setCart([]), čime se sprečava ponovno slanje iste porudžbine.

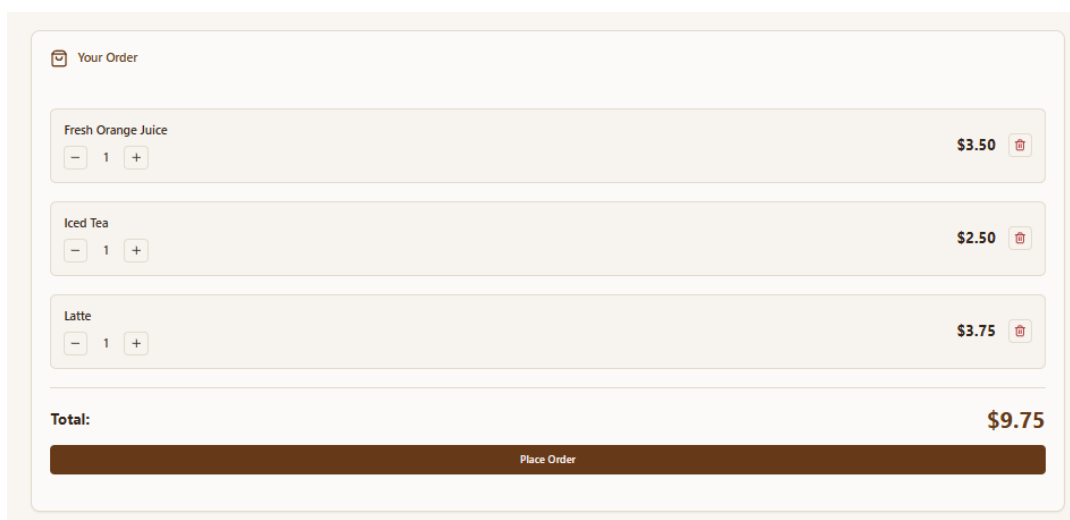
U slučaju greške, ona se ispisuje u konzolu radi lakše dijagnostike.

4.5.3 Poručivanje

Poručivanje odabranih artikala omogućava korisnicima jednostavan i brz način da završe proces naručivanja.

Nakon što izaberu željene artikle iz menija, korisnici mogu pregledati svoju porudžbinu, prilagoditi količine i potvrditi poručivanje.

Sistem automatski prosleđuje narudžbinu osoblju restorana, ceo proces je intuitivan, brz i optimizovan tako da pruži prijatno korisničko iskustvo uz minimalan broj koraka.

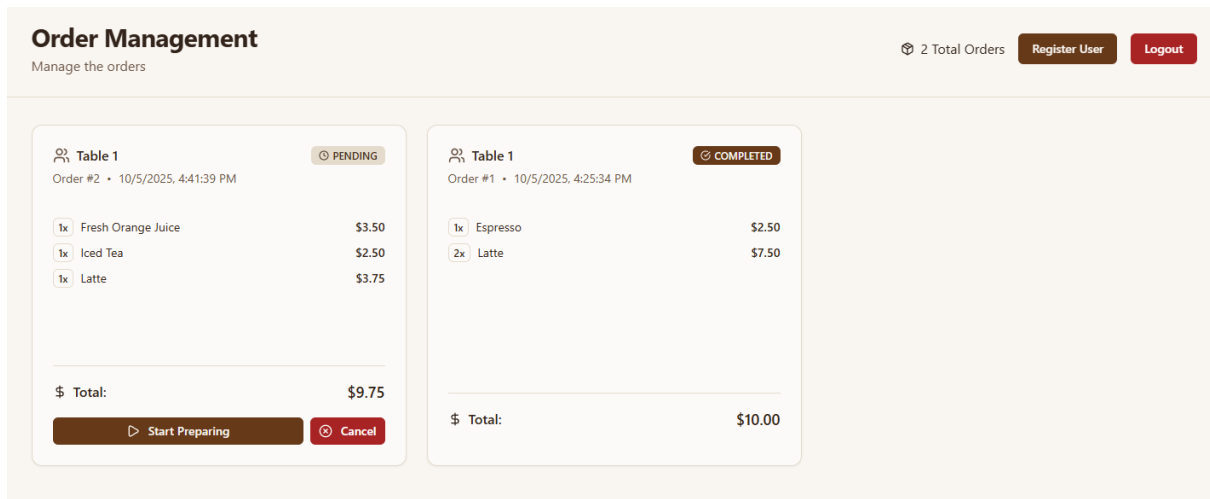


Slika 26 Poručivanje

4.5.4 Panel za radnike

Panel za radnike predstavlja namenski deo sistema koji omogućava osoblju restorana da **efikasno prati, organizuje i kontroliše sve porudžbine** u realnom vremenu.

Putem ovog panela zaposleni imaju uvid u sve pristigle narudžbine, mogu da ih **potvrde, ažuriraju status** (u pripremi, gotovo, isporučeno). Interfejs je jednostavan i pregledan, što omogućava lako korišćenje čak i u periodima veće gužve.



Slika 27 Panel za radnike

```
export function useOrders() : { orders: OrderType[]; loading: boolean; ... } { Show usages
const [orders, setOrders] = useState<OrderType[]>([]);
const [loading, setLoading] = useState( initialState: true);
const [error, setError] = useState( initialState: false);
const { apiGet, apiPatch } = useApi();

async function fetchOrders() : Promise<void> { Show usages
  try {
    const response : Response = await apiGet( url: '/api/admin/orders');
    if (!response.ok) throw new Error( message: 'Failed to fetch orders');
    const data : any = await response.json();
    setOrders(data);
  } catch (err: any) {
    setError(err);
    console.error(err.message);
  } finally {
    setLoading( value: false);
  }
}

async function updateOrderStatus( Show usages
  orderId: any,
  status: "PENDING" | "IN_PROGRESS" | "COMPLETED" | "CANCELED",
) : Promise<void> {
  try {
    const response : any = await apiPatch( url: '/api/admin/orders/${orderId}', status);
    if (!response) throw new Error( message: 'Failed to update order status');

    // Update local state
    setOrders((prevOrders : OrderType[] ) : OrderType[] =>
      prevOrders.map((order : OrderType ) : OrderType =>
        order.id === orderId ? { ...order, status } : order,
      ),
    );
  } catch (err: any) {
    setError(err);
    console.error(err.message);
  }
}

useEffect(() : () => void => {
  fetchOrders();

  const interval : Timeout = setInterval(() : void => {
    fetchOrders();
  }, delay: 10000); // 10 sekundi

  return () : void => clearInterval(interval);
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, []);

return {
  orders,
  loading,
  error,
  fetchOrders,
  updateOrderStatus,
};
};
```

Slika 28 Radnik panel - funkcionalnost - Custom Hook

Definicija inicijalnih stanja i hook-ova:

- `useState` se koristi za upravljanje stanjima kao što su `orders` (lista porudžbina), `loading` (indikator učitavanja), `error` (poruka o grešci) i `status` (status porudžbine: `'PENDING'`, `'IN_PROGRESS'`, `'COMPLETED'`, `'CANCELED'`).
- `useEffect` se koristi za pokretanje inicijalnog dohvaćanja podataka o porudžbinama kada se komponenta montira, uz postavljanje intervala od 10 sekundi za periodičko osvježavanje.

Funkcija `fetchOrders`:

- Asinhrona funkcija koja dohvata podatke o porudžbinama putem API poziva (`api/admin/orders`).
- Postavlja `loading` na `true` dok se zahtev obrađuje.
- U slučaju uspešnog odgovora, ažurira stanje `orders` sa dohvaćenim podacima.
- U slučaju greške, postavlja `error` sa porukom o grešci.
- Na kraju postavlja `loading` na `false`.

Funkcija `updateOrderStatus`:

- Asinhrona funkcija koja ažurira status određene porudžbine putem API poziva (`api/admin/orders/{orderId}`).
- Prima parametre `orderId` (ID porudžbine) i `status` (novi status).
- U slučaju uspešnog ažuriranja, ažurira lokalno stanje `orders` tako što pronalazi i menja status odgovarajuće porudžbine.
- U slučaju greške, postavlja `error` sa porukom o grešci.

Upravljanje ciklusom:

- `useEffect` postavlja `setInterval` da poziva `fetchOrders` svakih 10 sekundi (10000 ms) za kontinuirano osvežavanje podataka.
- Kada se komponenta deaktivira, `clearInterval` se koristi za čišćenje intervala i sprečavanje memorijskih curenja.

4.5.5 Registracija i prijava radnika

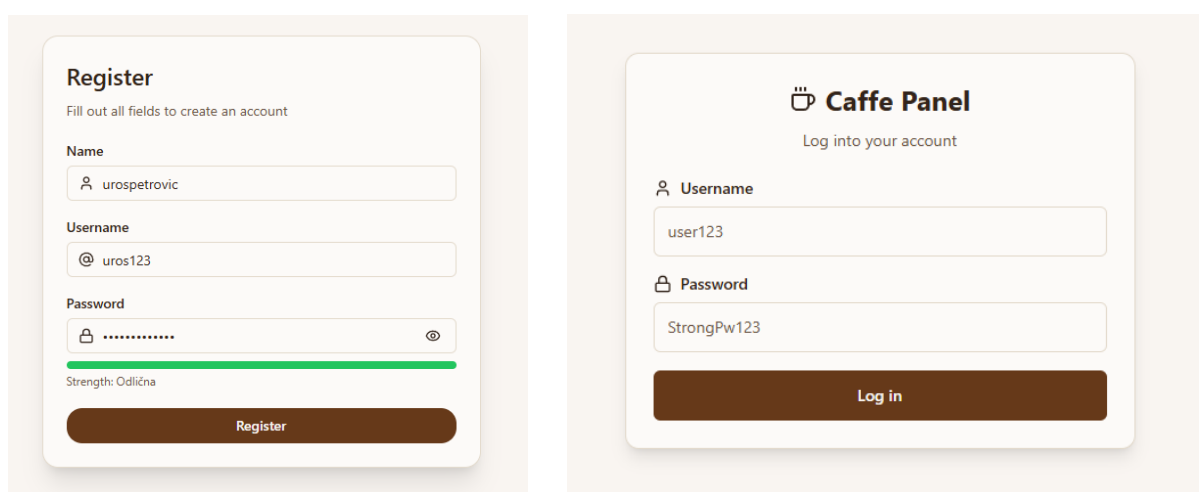
Panel za registraciju i prijavu radnika služi kao ulazna tačka za pristup **glavnom panelu za radnike**.

Ovaj deo sistema omogućava zaposlenima da **registruju** nove radnike kreiranjem korisničkog naloga ili da se **prijave** pomoću postojećih kredencijala.

Proces je jednostavan, siguran i zaštićen JWT tokenom [7] koji osigurava da samo ovlašćeno osoblje ima pristup funkcijama glavnog panela.

Nakon uspešne prijave, radnici mogu pregledati aktivne porudžbine i započeti rad bez nepotrebnog zadržavanja.

Ovaj panel obezbeđuje **kontrolu pristupa, sigurnost podataka i organizovan radni tok** unutar aplikacije. Sigurnost pored tokena omogućava konstantna provera postojanja tokena preko **Context-a i Middleware-a** [5] [1]



The image displays two side-by-side screenshots of a web application interface. The left screenshot shows a 'Register' form with the title 'Register' and the instruction 'Fill out all fields to create an account'. It contains three input fields: 'Name' (with the value 'urospetrovic'), 'Username' (with the value '@ uros123'), and 'Password' (with masked characters '.....'). Below the password field is a green progress bar and the text 'Strength: Odlična'. A brown 'Register' button is at the bottom. The right screenshot shows a 'Caffe Panel' login form with the title 'Caffe Panel' and the instruction 'Log into your account'. It contains two input fields: 'Username' (with the value 'user123') and 'Password' (with the value 'StrongPw123'). A brown 'Log in' button is at the bottom.

Slika 29 Registracija – forma

```

export function useRegister() : {loading: boolean; error: string | boolean} {
  const [loading, setLoading] = useState<boolean>({ initialState: false });
  const [error, setError] = useState<string | boolean>({ initialState: false });
  const {apiPost} = useApi();

  const registerUser : (data: any) => Promise<void> = async (data: any) : Promise<void> => {
    setLoading( value: true);
    setError( value: false);
    try {
      await apiPost( url: '/api/auth/register', data)
      toast.success( message: 'Register successful!');
    } catch (e) {
      console.error(e);
      toast.error( message: 'Something went wrong!');
      setError( value: 'Error while registering');
    } finally {
      setLoading( value: false);
    }
  };

  return {
    loading,
    error,
    registerUser,
  };
}

```

Slika 30 Registracija - funkcionalnost - Custom Hook

Definicija inicijalnih stanja i hook-ova:

- `useState` se koristi za upravljanje stanjima: `loading` (indikator učitavanja), `error` (poruka o grešci), i `success` (indikator uspešne registracije), svi inicijalizovani na `false` ili prazan string.

Funkcija `registerUser`:

- Asinhrona funkcija koja prima podatke (`data`) i šalje ih na API endpoint za registraciju (`/api/auth/register`).
- Postavlja `loading` na `true` dok se zahtev obrađuje.
- U slučaju uspešnog odgovora, prikazuje obaveštenje o uspešnoj registraciji koristeći `toast.success`.
- U slučaju greške, postavlja `error` sa porukom "Something went wrong" ili "Error while registering" i prikazuje obaveštenje o grešci koristeći `toast.error`.
- Na kraju postavlja `loading` na `false`, čime se završava proces učitavanja.

Struktura koda:

- Kod koristi `try-catch` blokove za upravljanje greškama tokom asinhronog poziva.
- Funkcija `toast` (verovatno iz biblioteke poput `react-toastify`) se koristi za prikazivanje obaveštenja korisniku.
- `useApi` (pretpostavlja se da je custom hook ili util) se koristi za izvršavanje HTTP POST zahteva.

```

export const useUserContext: any = () :any => { Show usages  ⚡ Uros Petrovic
  const context :any = useContext(UserContext);
  if (context === undefined) {
    throw new Error( message: "User Context must be used within an User Provider");
  }
  return context;
};

// Decode the token as a current session Details
// Frontend Decode (no secret and validation)
export async function decodeToken(token: any): Promise<UserSession | null> { Show usages  ⚡ Uros Petrovic
  try {
    if (!token) {
      console.error( message: "Token is empty");
      return null;
    }
    const decodedToken: any = await decodeJwt(token);
    if (!decodedToken) {
      console.error( message: "Decoded token is empty");
      return null;
    }

    if (!decodedToken.username) {
      console.error( message: "Invalid Token");
      return null;
    }
    return decodedToken;
  } catch (error) {
    console.error( message: "Error decoding token:", error);
    return null;
  }
}

// User context provider component
export const UserProvider: React.FC<{ children: React.ReactNode }> = ({ Show usages  ⚡ Uros Petrovic
  children,
}: {children: React.ReactNode } ) :Element => {
  const [user, setUser] = useState<UserSession | null>( initialState: null);
  const [loading, setLoading] = useState( initialState: true);

  const login : (userData: UserSession, token: string) =>... = (userData: UserSession, token: string) :void => { S
    if (userData) {
      Cookies.set(tokenName, token, { expires: 0.7, path: "/" });
      setUser(userData);
    }
  };

  const logout : () => Promise<void> = async () :Promise<void> => { Show usages  ⚡ Uros Petrovic
    await Cookies.remove(tokenName);
    setUser( value: null);
  };

  const fetchCurrentUser : () => Promise<void> = useCallback(async () :Promise<void> => {
    setLoading( value: true);
    try {
      const userToken :string|undefined = Cookies.get(tokenName);
      if (userToken) {
        const decodedUser :UserSession|null = await decodeToken(userToken);
        if (!decodedUser) throw new Error( message: "Bad User Session");
        setUser(decodedUser);
      }
    } catch (error: any) {
      const message :any = error.message.error || error.message;
      console.error( message: "Error fetching user:", message);
    } finally {
      setLoading( value: false);
    }
  }, []);

  useEffect(() :void => {
    setLoading( value: true);
    fetchCurrentUser();
  }, [fetchCurrentUser]);

  return (
    <UserContext.Provider
      value={{
        user,
        login,
        logout,

```

Slika 31 Prijava - funkcionalnost - Custom Context

1. Definicija konteksta:

- `useUserContext = useContext(UserContext)` vraća trenutni kontekst korisnika. Ako kontekst nije definisan (npr. izvan `UserProvider`-a), baca grešku sa porukom "UserContext must be used within a UserProvider".

2. Dekodiranje tokena (`decodeToken`):

- Asinhrona funkcija `decodeToken` prima token i dekodira ga (pretpostavlja se pomoću neke biblioteke poput `jwt-decode`).
- Ako je token prazan, ispisuje poruku u konzoli i vraća `null`.
- Ako dekodiran token postoji, proverava da li sadrži `username`. Ako ne sadrži, ispisuje "InvalidToken" i vraća `null`.
- U slučaju greške prilikom dekodiranja, ispisuje poruku o grešci i vraća `null`.
- U suprotnom, vraća dekodirani token koji sadrži podatke o sesiji korisnika (npr. `UserSession` tipa).

3. UserProvider komponenta:

- Ovo je React komponenta koja pruža kontekst korisniku. Koristi `useState` za upravljanje stanjima: `user` (trenutni korisnik, inicijalno `null`), `loading` (indikator učitavanja, inicijalno `true`).
- Ako postoje podaci korisnika (`userData`) iz tokena (čuvanih u kolačićima), postavlja ih u stanje `user` i postavlja kolačić sa tokenom koji ističe za 0.7 dana.

4. Funkcija `logout`:

- Asinhrona funkcija koja uklanja token iz kolačića (`Cookies.remove`) i postavlja `user` na `null`.

5. Funkcija `fetchCurrentUser`:

- Asinhrona funkcija koja postavlja `loading` na `true` i dohvata trenutnog korisnika.
- Uzima token iz kolačića (`Cookies.get`).
- Ako token postoji, dekodira ga koristeći `decodeToken`. Ako je dekodiranje neuspješno, baca grešku "Bad User Session".
- U slučaju greške, prikazuje poruku o grešci u konzoli.
- Na kraju postavlja `loading` na `false`.

6. Upravljanje ciklusom (`useEffect`):

- Poziva `fetchCurrentUser` kada se komponenta montira, čime se osigurava da se korisnikova sesija proveri prilikom učitavanja aplikacije.

7. Vrednosti konteksta:

- `UserProvider` pruža vrednosti `value={{ user, login, logout, loading }}` svim komponentama unutar hijerarhije, omogućavajući pristup korisničkim podacima, funkcijama za logovanje/izlogovanje i statusu učitavanja.

Ovaj kontekst se koristi kroz celu aplikaciju za centralizovano upravljanje sesijom korisnika, uključujući proveru autentifikacije, rola korisnika (preko dekodiranog tokena), i sinhronizaciju stanja. Token se čuva u kolačićima, a kontekst osigurava da su podaci o korisniku dostupni svuda gde je potrebno, uz adekvatno upravljanje greškama i učitavanjem.

4.5.6 QrKod Generisanje

```
export type QRCodeProps = HTMLAttributes<HTMLDivElement> & {
  data: string;
  foreground?: string;
  background?: string;
  robustness?: "L" | "M" | "Q" | "H";
};

const okLchRegex = /okLch\(((0-9.]+)\s+((0-9.]+)\s+((0-9.]+)\s+)\)/;

const getOkLch = (color: string, fallback: [number, number, number] = [color: string, fallback: [number, number, number]]) => {
  const okLchMatch = RegExpMatchArray | null = color.match(okLchRegex);

  if (!okLchMatch) {
    return { l: fallback[0], c: fallback[1], h: fallback[2] };
  }

  return {
    l: Number.parseFloat(okLchMatch[1]),
    c: Number.parseFloat(okLchMatch[2]),
    h: Number.parseFloat(okLchMatch[3]),
  };
};

export const QRCode = ({ data, foreground, background, robustness, ...props }) => {
  const [svg, setSVG] = useState<string | null>({ initialState: null });

  useEffect(() => {
    const generateQR = () => Promise<void> = async () => Promise<void> => {
      try {
        const styles = getComputedStyle(document.documentElement);
        const foregroundColor :string =
          foreground ?? styles.getPropertyValue( property: "--foreground");
        const backgroundColor :string =
          background ?? styles.getPropertyValue( property: "--background");

        const foregroundOkLch :{ l: number; c: number; h: number } = getOkLch(
          foregroundColor,
          [0.21, 0.006, 285.885],
        );
        const backgroundOkLch :{ l: number; c: number; h: number } = getOkLch(backgroundColor, [0.985, 0, 0]);

        const newSvg :string = await QR.toString(data, {
          type: "svg",
          color: {
            dark: formatHex(okLch({ mode: "okLch", ...foregroundColorOkLch })),
            light: formatHex(okLch({ mode: "okLch", ...backgroundColorOkLch })),
          },
          width: 200,
          errorCorrectionLevel: robustness,
          margin: 0,
        });

        setSVG(newSvg);
      } catch (err) {
        console.error(err);
      }
    };

    generateQR();
  }, [data, foreground, background, robustness]);

  if (!svg) {
    return null;
  }

  return (
    <div
      className={cn( __inputs: "size-full", "[&_svg]:size-full", className)}
      // biome-ignore lint/security/noDangerouslySetInnerHtml: "Required for SVG"
      dangerouslySetInnerHTML={{ __html: svg }}
      {...props}
    />
  );
};
```

Slika 32 QrKod - generisanje linka

Definicija tipova i podataka:

- `type QRCodeProps = HTMLAttributes<HTMLImageElement> & { background?: string; foreground?: string; robustness?: number; };` definiše propsove za komponentu, uključujući opcione attribute HTML-a, boju pozadine, prednju boju i nivo robustnosti.
- `type QRCode = { data: string; foreground: string; background: string; robustness: number; };` definiše tip za podatke QR koda.

Funkcija getColor:

- Pomoćna funkcija koja obrađuje boju iz propsa. Ako je boja validna (u formatu #HEX ili RGB), vraća je; u suprotnom, koristi padajući izbor (fallback) ili podrazumevanu vrednost.

Komponenta QRCode:

- **Stanje:** Koristi `useState` za upravljanje `svg` (generisanim QR kodom u SVG formatu) i `error` (porukom o grešci, inicijalno `null`).
- **Asinhrona funkcija generateQR:** Generiše QR kod koristeći `qrcode.toString` sa unetim podacima (`data`), bojom pozadine, prednjom bojom i nivoom robustnosti. Rezultat se čuva u `svg` stanju. U slučaju greške, postavlja se `error`.
- **Podaci:** Uzima `data`, `foreground`, `background`, i `robustness` iz propsa ili koristi podrazumevane vrednosti (npr. crna za prednju boju, bela za pozadinu, 0.8 za robustnost).
- **Stilovi:** Dinamički se generišu klase i inline stilovi na osnovu unetih boja i robustnosti, uz validaciju unosa (npr. za `className`).

4.6 Povezivanje celine – Docker Compose

Da bi se svi delovi aplikacije pokrenuli i međusobno komunicirali na jednostavan i konzistentan način, korišćen je Docker Compose. U **docker-compose.yml** fajlu definisana su dva glavna servisa:

- **next-app:** Servis koji gradi i pokreće Next.js aplikaciju na osnovu Dockerfile-a u korenu projekta. Ovde su definisane environment varijable, poput konekcionog stringa za bazu podataka i tajnog ključa za JWT.
- **db:** Servis koji pokreće kontejner sa MySQL bazom podataka, definišući korisničko ime, lozinku i naziv baze. Takođe, definisan je i volumen za perzistentnost podataka, kako se podaci ne bi gubili prilikom restartovanja kontejnera..

Ovaj fajl omogućava da se oba servisa pokrenu, povežu unutar iste Docker mreže i budu spremni za korišćenje jednom jedinom komandom

```

1  services:
2  app:
3      image: node:24
4      container_name: coffee_qr_app
5      working_dir: /src/app
6      command: sh -c "npm install && npx prisma db push && npm run seed && npm run build && npm run start"
7      volumes:
8          - ./src/app
9      ports:
10         - "3000:3000"
11      environment:
12         DATABASE_URL: ${DATABASE_URL:-mysql://root:root@db:3306/qrcode}
13         NEXT_PUBLIC_AUTH_TOKEN: ${NEXT_PUBLIC_AUTH_TOKEN}
14         API_AUTH_TOKEN_SECRET: ${API_AUTH_TOKEN_SECRET}
15         NEXT_PUBLIC_URL: ${NEXT_PUBLIC_URL}
16      depends_on:
17         db:
18             condition: service_healthy
19      restart: unless-stopped
20      env_file:
21         - .env
22
23  db:
24      image: mysql:8.1
25      container_name: mysql_db
26      restart: always
27      environment:
28         MYSQL_ROOT_PASSWORD: root
29         MYSQL_DATABASE: qrcode
30      ports:
31         - "3306:3306"
32      volumes:
33         - db_data:/var/lib/mysql
34      healthcheck:
35         test: ["CMD-SHELL", "mysqladmin ping -h localhost -uroot -proot"]
36         interval: 10s
37         timeout: 5s
38         retries: 5
39         start_period: 30s
40
41  volumes:
42      db_data:
43      node_modules:

```

Slika 33 Docker Compose

5 Struktura projekta (tree-view)

```

QrCode-Smart-Order/
├── .gitignore
├── .idea/
│   ├── .gitignore
│   ├── inspectionProfiles/
│   │   └── Project_Default.xml
│   ├── modules.xml
│   ├── prettier.xml
│   ├── qrcode.iml
│   └── vcs.xml
├── components.json
├── docker-compose.yml
├── env-example
├── eslint.config.mjs
├── next.config.ts
├── package-lock.json
├── package.json
├── postcss.config.js
├── prisma/
│   ├── migrations/
│   │   ├── 20250831161656_initial/
│   │   │   └── migration.sql
│   │   ├── 20250831170612_fullfill_db/
│   │   │   └── migration.sql
│   │   ├── 20250831184452_user_session/
│   │   │   └── migration.sql
│   │   ├── 20250901142613_/
│   │   │   └── migration.sql
│   │   ├── 20250902121002_image_url/
│   │   │   └── migration.sql
│   │   └── migration_lock.toml
│   ├── schema.prisma
│   └── seed.js
├── README.md
├── src/
│   ├── app/
│   │   ├── 403/
│   │   │   └── page.tsx
│   │   ├── (auth)/
│   │   │   ├── admin/
│   │   │   │   ├── layout.tsx
│   │   │   │   ├── orders/
│   │   │   │   │   └── page.tsx
│   │   │   │   └── page.tsx
│   │   │   ├── layout.tsx
│   │   │   ├── login/
│   │   │   │   └── page.tsx
│   │   └── api/
│   │       ├── admin/
│   │       │   ├── orders/
│   │       │   │   ├── [id]/
│   │       │   │   │   └── route.ts
│   │       │   │   └── route.ts
│   │       ├── auth/
│   │       │   ├── login/
│   │       │   │   └── route.ts
│   │       │   ├── register/
│   │       │   │   └── route.ts
│   │       ├── orders/
│   │       │   └── route.ts
│   │       ├── products/
│   │       │   └── route.ts
│   │       ├── recommendations/
│   │       │   └── route.ts
│   │       ├── error.tsx
│   │       ├── favicon.ico
│   │       ├── globals.css
│   │       ├── layout.tsx
│   │       ├── menu/
│   │       │   ├── components/
│   │       │   │   ├── MenuPage.tsx
│   │       │   │   ├── ProductCard.tsx
│   │       │   │   └── RecommendationCard.tsx
│   │       │   └── page.tsx
│   │       ├── page.tsx
│   │       ├── components/
│   │       │   ├── Footer.tsx
│   │       │   ├── QrCode.tsx
│   │       │   └── ui/
│   │       │       ├── alert.tsx
│   │       │       ├── badge.tsx
│   │       │       ├── button.tsx
│   │       │       ├── card.tsx
│   │       │       ├── form.tsx
│   │       │       ├── input.tsx
│   │       │       ├── label.tsx
│   │       │       ├── loading.tsx
│   │       │       ├── separator.tsx
│   │       │       ├── shadcn-io/
│   │       │       │   └── qr-code/
│   │       │       │       ├── index.tsx
│   │       │       │       └── server.tsx
│   │       │       └── sonner.tsx
│   │       ├── context/
│   │       │   └── UserContext.tsx
│   │       ├── hooks/
│   │       │   ├── useApi.tsx
│   │       │   ├── useCart.tsx
│   │       │   ├── useOptions.tsx
│   │       │   ├── useOrders.tsx
│   │       │   └── useRegister.tsx
│   │       ├── lib/
│   │       │   ├── db.ts
│   │       │   ├── session/
│   │       │   │   ├── createUserSession.ts
│   │       │   │   ├── signJWT.ts
│   │       │   │   ├── validateApiToken.ts
│   │       │   │   └── verifyJWT.ts
│   │       │   ├── utils.ts
│   │       │   └── validation.ts
│   │       ├── services/
│   │       │   ├── Jwt/
│   │       │   │   └── index.ts
│   │       │   ├── Order/
│   │       │   │   └── index.ts
│   │       │   ├── Products/
│   │       │   │   └── index.ts
│   │       │   ├── Recommendation/
│   │       │   │   └── index.ts
│   │       │   ├── Table/
│   │       │   │   └── index.ts
│   │       │   ├── User/
│   │       │   │   └── index.ts
│   │       └── types/
│   │           ├── CartItemType.ts
│   │           ├── OrderItemType.ts
│   │           ├── OrderType.ts
│   │           ├── ProductType.ts
│   │           ├── RecommendationType.ts
│   │           └── UserSession.ts
│   ├── tailwind.config.js
│   └── tsconfig.json

```

Slika 34 Tree-view Projekta

6 Zaključak

Projekat "QRCode-Smart-Order" uspešno je demonstrirao kako se primenom modernih veb tehnologija može rešiti konkretan problem iz stvarnog sveta, kao što je optimizacija procesa naručivanja u ugostiteljstvu. Korišćenjem integrisanog pristupa koji nude Next.js za full-stack razvoj, Prisma za bezbedan rad sa podacima, Tailwind CSS za brzo stilizovanje, React za izgradnju korisničkog interfejsa, JWT za sigurnu autentifikaciju i Docker za konzistentno okruženje, razvijena je funkcionalna, skalabilna i lako održiva aplikacija.

Rad je praktično pokazao snagu monolitne arhitekture realizovane unutar Next.js frejmworka, gde su API rute služile kao pozadinska logika, eliminišući potrebu za odvojenom serverskom aplikacijom. Demonstrirano je kako Prisma, kroz svoju deklarativnu šemu i tipski bezbedan klijent, osigurava pouzdanu i sigurnu komunikaciju sa MySQL bazom podataka, smanjujući mogućnost grešaka pri radu sa podacima. Na korisničkom interfejsu, primena Tailwind CSS-a je omogućila izuzetno brzo stilizovanje direktno u JSX kodu, dok je JWT mehanizam efikasno zaštitio administrativne rute. Konačno, ceo sistem, uključujući aplikaciju i bazu podataka, uspešno je kontejnerizovan pomoću Dockera i Docker Compose-a, čime je postavljanje celokupnog okruženja svedeno na jednu komandu.

Iako je trenutna verzija aplikacije potpuno funkcionalna, postoje brojni pravci za dalja unapređenja i proširenja:

- **Implementacija plaćanja putem interneta:** Integracija sa nekim od popularnih payment gateway servisa (npr. Stripe) kako bi korisnici mogli da plate svoje narudžbine direktno putem aplikacije, koristeći kreditne kartice.
- **Sistem za ocenjivanje proizvoda:** Dodavanje mogućnosti da korisnici ocenjuju i ostavljaju recenzije za proizvode, što bi pružilo vredne povratne informacije vlasnicima objekta i pomoglo drugim gostima pri izboru.
- **Admin panel za upravljanje sadržajem proizvoda:** Razvoj sveobuhvatnog administrativnog panela koji bi omogućio osoblju ili menadžerima da lako dodaju, menjaju i brišu proizvode i kategorije, prate statistiku narudžbina i upravljaju sadržajem bez potrebe za direktnom izmenom koda ili baze podataka.

7 Literatura

- [1] Vercel, "Next.js Documentation," [Online]. Available: <https://nextjs.org/docs>.
- [2] M. Panin, RAZVOJ MODERNIH WEB APLIKACIJA PRIMENOM NEXT.JS, Vrnjačka Banja, 2024.
- [3] Prisma, "Prisma ORM Documentation," [Online]. Available: <https://www.prisma.io/docs>.
- [4] T. CSS, "Tailwind CSS Documentation," [Online]. Available: <https://tailwindcss.com/docs>.
- [5] Meta, "React Documentation," [Online]. Available: <https://react.dev/learn>.
- [6] A. B. a. E. Porcello, Learning React: Modern Patterns for Developing React Apps, Chicago, 2020.
- [7] J. W. Tokens, "JWT Documentation," [Online]. Available: <https://jwt.io/>.
- [8] Docker, "Docker Documentation," [Online]. Available: <https://docs.docker.com/>.

Tabela Slika

Slika 1 Primer barkoda.....	6
Slika 2 QRCode u vidu pretrage na internetu.....	7
Slika 3 QRCode ka website-u.....	7
Slika 4 Data Matrix.....	9
Slika 5 Aztec Code.....	10
Slika 6 PDF417.....	11
Slika 7 MaxiCode.....	12
Slika 8 Code 39.....	13
Slika 9 Code 128.....	13
Slika 10 Codabar.....	14
Slika 11 Next.js.....	17
Slika 12 Prisma ORM.....	18
Slika 13 Tailwind CSS.....	18
Slika 14 JWT Token.....	19
Slika 15 Docker.....	19
Slika 16 Use Case Dijagram.....	21
Slika 17 Inicijalizacija next.js projekta.....	22
Slika 18 Prisma Schema.....	23
Slika 19 Dijagram baze.....	24
Slika 20 Next.js API rute/Kontroleri i Servisi.....	25
Slika 21 Glavni folder komponenata.....	25
Slika 22 Menu Page komponente.....	25
Slika 23 Početna strana.....	26
Slika 24 Prikaz menija.....	27
Slika 25 Menu stranica – funkcionalnost.....	28
Slika 26 Poručivanje.....	29
Slika 27 Panel za radnike.....	30
Slika 28 Radnik panel - funkcionalnost - Custom Hook.....	30
Slika 29 Registracija – forma.....	32
Slika 30 Registracija - funkcionalnost - Custom Hook.....	33
Slika 31 Prijava - funkcionalnost - Custom Context.....	34
Slika 32 QRKod - generisanje linka.....	36
Slika 33 Docker Compose.....	38
Slika 34 Tree-view Projekta.....	39