

Cloud-Seminarski Rad

Mihajlo Jovičić 2023/0414

Korisnički zahtev i opis namene aplikacije

Aplikacija predstavlja webshop servis koji omogućava interakciju između kupaca i prodavaca.

- Prodavci imaju mogućnost da dodaju, izmenjuju ili uklanjaju svoje proizvode.
- Kupci mogu da pregledaju sve proizvode dostupne kod različitih prodavaca i da ih kupuju ukoliko imaju dovoljno sredstava na svom računu. U slučaju uspešne kupovine, sredstva se transferuju sa računa kupca na račun prodavca. Kupci takođe imaju mogućnost da povećaju svoj bilans sredstava.

Svaki proizvod poseduje sledeće podatke, koje unosi prodavac:

- naziv
- opis
- cena
- slika

Pristup aplikaciji, kao i pregled i kupovina proizvoda, je moguć isključivo ulogovanim korisnicima

Opis tehnologija korišćenih u aplikaciji

Projekat je organizovan u tri osnovna dela: backend, baza podataka i frontend, pri čemu svaka komponenta koristi odgovarajuće tehnologije i biblioteke.

Backend

Backend aplikacije je razvijen korišćenjem Node.js i Express.js. Node.js omogućava izvršavanje JavaScript koda na serverskoj strani, dok Express.js pruža jednostavan način za kreiranje REST API-ja i rukovanje HTTP zahtevima. Backend je odgovoran za:

- autentifikaciju i registraciju korisnika,
- upravljanje proizvodima i transakcijama,
- obradu fajlova za upload slika,
- komunikaciju sa bazom podataka i validaciju unosa.

Dodatne biblioteke koje se koriste u backend-u uključuju:

- body-parser za parsiranje JSON zahteva,
- cors za omogućavanje zahteva sa različitih domena,
- multer za upload fajlova,
- bcrypt za šifrovanje lozinki,
- jsonwebtoken za kreiranje i verifikaciju JWT tokena, što omogućava sigurnu autentifikaciju korisnika,
- mysql2 za povezivanje i rad sa MySQL bazom podataka.

Baza podataka

Za čuvanje podataka koristi se MySQL. Struktura baze je definisana u SQL skripti init.sql i sadrži dve osnovne tabele: users i products.

- Tabela users čuva podatke o korisnicima, uključujući ime, email, šifrovanu lozinku, bilans sredstava i ulogu (prodavac ili kupac).
- Tabela products čuva podatke o proizvodima, uključujući naziv, opis, cenu, putanju do slike i identifikator korisnika koji je dodao proizvod.

Baza omogućava sigurno čuvanje podataka, relacione veze između korisnika i proizvoda, kao i efikasno izvršavanje upita.

Frontend

Frontend je napravljen korišćenjem React.js, biblioteke koja omogućava kreiranje dinamičnih i interaktivnih korisničkih interfejsa kroz komponente koje se mogu višekratno koristiti.

U frontend-u su korišćene sledeće biblioteke:

- react-router-dom za upravljanje navigacijom između stranica,
- axios za komunikaciju sa backend API-jem,
- jwt-decode za dekodiranje JWT tokena i proveru autentifikacije korisnika,

Struktura frontend projekta uključuje:

- pages – stranice aplikacije, npr. lista proizvoda, prijava i registracija,
- styles – CSS fajlove za stilizaciju aplikacije,
- App.js – glavnu komponentu koja povezuje sve stranice i rute,
- index.js – ulaznu tačku aplikacije koja pokreće React i povezuje ga sa HTML dokumentom.

Backend i baza podataka pružaju sigurno upravljanje podacima i transakcijama, dok frontend omogućava pregledan i interaktivan interfejs za korisnike.

Korisničko uputstvo

MSHOP

[Login](#) [Registracija kupca](#) [Registracija prodavca](#)

Prijava

Email:

Lozinka:

Prijavi se

Svaki neulogovani korisnik koji pokuša da pristupi početnoj stranici (/) ili bilo kojoj drugoj zaštićenoj ruti automatski se preusmerava na stranicu za prijavu. Sa te stranice korisnik može da ode na rutu za registraciju kupca ili na rutu za registraciju prodavca, u zavisnosti od toga koju vrstu naloga želi da kreira.

MSHOP

Login Registracija kupca Registracija prodavca

Registracija prodavca

Ime:

Email:

Lozinka:

Registruj se

Registracija kupca i prodavca su slične, backend sam u pozadini određuje rolu koju će dodeliti novom korisniku u zavisnosti od rute sa koje je pristupio

MSHOP

0.00 RSD Povećaj balans Dodaj proizvod Vasi proizvodi Logout

Dodaj proizvod

Naziv:

Opis:

Cena:

Slika:

Odaбери fajл | Није одабрано

Sačuvaj

Ulogovani prodavci mogu dodavati svoje proizvode, pri čemu je potrebno dodati naziv, opis, cenu i sliku proizvoda pri čemu backend automatski čuva ko je vlasnik proizvoda

Vasa lista proizvoda



Lopta

1000.00 RSD

Izmeni

Obrisi



Autic

350.00 RSD

Izmeni

Obrisi

Prodavac ima mogućnost da vidi samo svoje proizvode, I da iskoristi opcije obriši koja će izbaciti proizvod iz baze, ili da izmeni proizvod

Izmeni proizvod




Naziv:	<input type="text" value="Autic"/>
Opis:	<input type="text" value="Hot wheels autic"/>
Cena:	<input type="text" value="350.00"/>
Slika:	<input type="text" value="Odaбери fajл Није одабрано"/>
<input type="button" value="Sačuvaj"/>	

Pri izmeni proizvoda ima vide se predjašnji podaci i po potrebi mogu da se unesu novi


MSHOP0.00 RSDPovećaj balansLogout

Lista proizvoda




Lopta
1000.00 RSD

Kupi




Autic
350.00 RSD

Kupi



Lego Set
2000.00 RSD

Kupi



Kosarkaska lopta
1000.00 RSD

Kupi

Kupac ima mogućnost da vidi proizvode svih prodavaca I ukoliko želi može da ih kupi.

Ako ne ma dovoljno sredstava na računu izaćiće mu sledeći popaut

localhost:3001 наводи:
Nemate dovoljno sredstava za kupovinu

Потврди

A dodavanje sredstava je moguće preko opcije Povećaj bilans koja izgleda ovako:

Dodaj budžet

Unesite iznos

Dodaj

Ukoliko se kupovina uspešno ralizuje proizvod će biti uklonjen,
sredstva kupca smanjena a sredstva prodavca povećana

Prikaz reprezentativnih delova koda

```
db > init.sql
1 CREATE TABLE IF NOT EXISTS users (
2   id INT AUTO_INCREMENT PRIMARY KEY,
3   name VARCHAR(100),
4   email VARCHAR(100) UNIQUE,
5   password VARCHAR(100),
6   balance DECIMAL(10,2) DEFAULT 0.00,
7   role ENUM('seller', 'customer') DEFAULT 'customer'
8 );
9
10 CREATE TABLE IF NOT EXISTS products (
11   id INT AUTO_INCREMENT PRIMARY KEY,
12   name VARCHAR(100),
13   description TEXT,
14   price DECIMAL(10,2),
15   image_path VARCHAR(255),
16   user_id INT,
17   FOREIGN KEY (user_id) REFERENCES users(id)
18 );
19
```

Ovo je init.sql fajl u kom definišemo tabele users(korisnik) i products(proizvodi)

Glavni fajl db servisa

Server.js backend rute:

```
56 // Dohvatanje svih proizvoda
57 app.get('/products', (req, res) => {
58   const query = 'SELECT * FROM products';
59   db.query(query, (err, results) => {
60     if (err) return res.status(500).json({ error: 'Greška servera' });
61     res.json(results);
62   });
63 });
64
65 // Dohvatanje proizvoda za trenutno ulogovanog sellera
66 app.get('/seller/products', (req, res) => {
67   const authHeader = req.headers.authorization;
68   if (!authHeader) return res.status(401).json({ error: 'Niste autorizovani' });
69
70   const token = authHeader.split(' ')[1];
71   if (!token) return res.status(401).json({ error: 'Niste autorizovani' });
72
73   let decoded;
74   try {
75     decoded = jwt.verify(token, SECRET_KEY);
76   } catch (err) {
77     return res.status(401).json({ error: 'Nevažeći token' });
78   }
79   const query = 'SELECT * FROM products WHERE user_id = ?';
80   db.query(query, [decoded.id], (err, results) => {
81     if (err) return res.status(500).json({ error: 'Greška servera' });
82     res.json(results);
83   });
84 });
85
86 // Dodavanje proizvoda za user_id
87 app.post('/products', upload.single('image'), (req, res) => {
88   const { name, description, price, user_id } = req.body;
89   const imagePath = req.file ? `uploads/${req.file.filename}` : null;
90
91   if (!user_id) {
92     return res.status(400).json({ error: "Nije prosleđen user_id" });
93   }
94
95   const query = 'INSERT INTO products (name, description, price, image_path, user_id) VALUES (?, ?, ?, ?, ?)';
96   db.query(query, [name, description, price, imagePath, user_id], (err, result) => {
97     if (err) return res.status(500).json({ error: 'Greška servera' });
98     res.json({ message: 'Proizvod uspešno dodat!', productId: result.insertId });
99   });
100 });
101
102 // Ažuriranje proizvoda sa opcionalnom slikom
103 app.put('/products/:id', upload.single('image'), (req, res) => {
104   const { id } = req.params;
105   const { name, description, price } = req.body;
106
107   if (req.file) {
108     const imagePath = `uploads/${req.file.filename}`;
109     const query = 'UPDATE products SET name=?, description=?, price=?, image_path=? WHERE id=?';
110     db.query(query, [name, description, price, imagePath, id], (err, result) => {
111       if (err) return res.status(500).json({ error: 'Greška servera' });
112       res.json({ message: 'Proizvod uspešno ažuriran sa slikom!' });
113     });
114   } else {
115     const query = 'UPDATE products SET name=?, description=?, price=? WHERE id=?';
116     db.query(query, [name, description, price, id], (err, result) => {
117       if (err) return res.status(500).json({ error: 'Greška servera' });
118       res.json({ message: 'Proizvod uspešno ažuriran!' });
119     });
120   }
121 });
122 }
```

```

123 // Brisanje proizvoda
124 app.delete('/products/:id', (req, res) => {
125   const { id } = req.params;
126   const query = 'DELETE FROM products WHERE id = ?';
127   db.query(query, [id], (err, result) => {
128     if (err) return res.status(500).json({ error: 'Greška servera' });
129     res.json({ message: 'Proizvod uspešno obrisano!' });
130   });
131 });
132
133 // Dohvatanje jednog proizvoda po ID-u
134 app.get('/products/:id', (req, res) => {
135   const { id } = req.params;
136   const query = 'SELECT * FROM products WHERE id = ?';
137   db.query(query, [id], (err, results) => {
138     if (err) return res.status(500).json({ error: 'Greška servera' });
139     if (results.length === 0) return res.status(404).json({ error: 'Proizvod nije pronađen' });
140     res.json(results[0]);
141   });
142 });

```

```

144 // Registracija korisnika (customer ili seller)
145 app.post('/users/register', async (req, res) => {
146   const { name, email, password, role } = req.body;
147
148   // Provera da li su obavezna polja popunjena
149   if (!name || !email || !password) {
150     return res.status(400).json({ error: 'Sva polja su obavezna!' });
151   }
152
153   try {
154     // Proveravamo da li korisnik sa istim emailom već postoji
155     db.query('SELECT * FROM users WHERE email = ?', [email], async (err, results) => {
156       if (err) return res.status(500).json({ error: 'Greška servera' });
157
158       if (results.length > 0) {
159         return res.status(400).json({ error: 'Korisnik već postoji' });
160       }
161
162       // Šifrujemo lozinku
163       const hashedPassword = await bcrypt.hash(password, 10);
164
165       // role može biti 'seller' ili 'customer' (ako nije poslato, default je 'customer')
166       const userRole = role === 'seller' ? 'seller' : 'customer';
167
168       // Ubacujemo korisnika u bazu
169       db.query(
170         'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
171         [name, email, hashedPassword, userRole],
172         (err, result) => {
173           if (err) return res.status(500).json({ error: 'Greška servera' });
174
175           res.json({ message: 'Registracija uspešna!' });
176         }
177       );
178     });
179   } catch (err) {
180     console.error(err);
181     res.status(500).json({ error: 'Greška servera' });
182   }
183 });

```

```

185 // Prijava korisnika
186 app.post('/users/login', (req, res) => {
187   const { email, password } = req.body;
188
189   if (!email || !password) {
190     return res.status(400).json({ error: 'Unesite email i lozinku' });
191   }
192
193   db.query('SELECT * FROM users WHERE email = ?', [email], async (err, results) => {
194     if (err) return res.status(500).json({ error: 'Greška servera' });
195
196     if (results.length === 0) {
197       return res.status(400).json({ error: 'Neispravan email ili lozinka' });
198     }
199
200     const user = results[0];
201
202     // Proveravamo lozinku
203     const isMatch = await bcrypt.compare(password, user.password);
204
205     if (!isMatch) {
206       return res.status(400).json({ error: 'Neispravan email ili lozinka' });
207     }
208
209     // nakon što proverimo lozinku
210     const token = jwt.sign(
211       { id: user.id, role: user.role, name: user.name },
212       SECRET_KEY,
213       { expiresIn: '1h' } // token važi 1 sat
214     );
215
216     res.json({
217       message: 'Uspešno prijavljen!',
218       token,
219       user: {
220         id: user.id,
221         name: user.name,
222         email: user.email,
223         role: user.role,
224       },
225     });
226   });
227 });

```

```

228 app.post('/addbalance', (req, res) => {
229   const authHeader = req.headers.authorization;
230   if (!authHeader) return res.status(401).json({ error: 'Niste autorizovani' });
231
232   const token = authHeader.split(' ')[1];
233   if (!token) return res.status(401).json({ error: 'Niste autorizovani' });
234
235   let decoded;
236   try {
237     | decoded = jwt.verify(token, SECRET_KEY);
238   } catch (err) {
239     | return res.status(401).json({ error: 'Nevažeći token' });
240   }
241
242   const { amount } = req.body;
243   if (!amount || isNaN(amount)) {
244     | return res.status(400).json({ error: 'Nevažeći iznos' });
245   }
246
247   const query = 'UPDATE users SET balance = balance + ? WHERE id = ?';
248   db.query(query, [parseFloat(amount), decoded.id], (err, result) => {
249     | if (err) return res.status(500).json({ error: 'Greška servera' });
250     | res.json({ message: 'Balance uspešno ažuriran.' });
251   });
252 });
253
254 // Ruta za dohvaćanje trenutnog balansa ulogovanog korisnika
255 app.get('/user/balance', (req, res) => {
256   const authHeader = req.headers.authorization;
257   if (!authHeader) return res.status(401).json({ error: 'Niste autorizovani' });
258
259   const token = authHeader.split(' ')[1];
260   if (!token) return res.status(401).json({ error: 'Niste autorizovani' });
261
262   let decoded;
263   try {
264     | decoded = jwt.verify(token, SECRET_KEY);
265   } catch (err) {
266     | return res.status(401).json({ error: 'Nevažeći token' });
267   }
268
269   const query = 'SELECT balance FROM users WHERE id = ?';
270   db.query(query, [decoded.id], (err, result) => {
271     | if (err) {
272       | console.error('Greška prilikom dohvaćanja balansa:', err);
273       | return res.status(500).json({ error: 'Greška servera' });
274     }
275
276     | if (result.length === 0) {
277       | return res.status(404).json({ error: 'Korisnik nije pronađen' });
278     }
279
280     | res.json({ balance: result[0].balance });
281   });
282 });

```

```

284 // Kupovina proizvoda
285 app.post('/buy/:productId', (req, res) => {
286   const authHeader = req.headers.authorization;
287   if (!authHeader) return res.status(401).json({ error: 'Niste autorizovani' });
288
289   const token = authHeader.split(' ')[1];
290   if (!token) return res.status(401).json({ error: 'Niste autorizovani' });
291
292   let decoded;
293   try {
294     decoded = jwt.verify(token, SECRET_KEY);
295   } catch (err) {
296     return res.status(401).json({ error: 'Nevažeći token' });
297   }
298
299   const buyerId = decoded.id;
300   const { productId } = req.params;
301
302   // Dohvati proizvod
303   const getProductQuery = 'SELECT * FROM products WHERE id = ?';
304   db.query(getProductQuery, [productId], (err, productResults) => {
305     if (err) return res.status(500).json({ error: 'Greška servera (dohvatanje proizvoda)' });
306     if (productResults.length === 0) return res.status(404).json({ error: 'Proizvod nije pronađen' });
307
308     const product = productResults[0];
309     const sellerId = product.user_id;
310     const price = parseFloat(product.price);
311
312     // Proveri da li kupac ima dovoljno novca
313     const getBuyerQuery = 'SELECT balance FROM users WHERE id = ?';
314     db.query(getBuyerQuery, [buyerId], (err, buyerResults) => {
315       if (err) return res.status(500).json({ error: 'Greška servera (dohvatanje kupca)' });
316       if (buyerResults.length === 0) return res.status(404).json({ error: 'Kupac nije pronađen' });
317
318       const buyerBalance = parseFloat(buyerResults[0].balance);
319
320       if (buyerBalance < price) {
321         return res.status(400).json({ error: 'Nemate dovoljno sredstava za kupovinu' });
322       }
323
324       // Ažuriraj balance kupca i prodavca i obriši proizvod
325       db.beginTransaction((err) => {
326         if (err) return res.status(500).json({ error: 'Greška u transakciji' });
327
328         const updateBuyer = 'UPDATE users SET balance = balance - ? WHERE id = ?';
329         const updateSeller = 'UPDATE users SET balance = balance + ? WHERE id = ?';
330         const deleteProduct = 'DELETE FROM products WHERE id = ?';
331
332         db.query(updateBuyer, [price, buyerId], (err) => {
333           if (err) return db.rollback(() => res.status(500).json({ error: 'Greška pri ažuriranju kupca' }));
334
335           db.query(updateSeller, [price, sellerId], (err) => {
336             if (err) return db.rollback(() => res.status(500).json({ error: 'Greška pri ažuriranju prodavca' }));
337
338             db.query(deleteProduct, [productId], (err) => {
339               if (err) return db.rollback(() => res.status(500).json({ error: 'Greška pri brisanju proizvoda' }));
340
341               db.commit((err) => {
342                 if (err) return db.rollback(() => res.status(500).json({ error: 'Greška pri potvrđivanju transakcije' }));
343
344                 res.json({ message: 'Kupovina uspešno obavljena!' });
345               });
346             });
347           });
348         });
349       });
350     });
351   });
352 });

```

Frontend App.js:

```
1 import React from "react";
2 import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
3 import ProductList from "../pages/ProductList";
4 import SellerProductList from "../pages/SellerProductList";
5 import AddProduct from "../pages/AddProduct";
6 import EditProduct from "../pages/EditProduct";
7 import RegisterCustomer from "../pages/RegisterCustomer";
8 import RegisterSeller from "../pages/RegisterSeller";
9 import Login from "../pages/Login";
10 import Layout from "../components/Layout";
11 import ProtectedRoute from "../components/ProtectedRoute";
12 import AddBalance from "../pages/AddBalance";
13 import "../styles/App.css";
14
15 function App() {
16   return (
17     <Router>
18       <Layout>
19         <Routes>
20           <Route path="/seller/productlist"
21             element=<ProtectedRoute allowedRoles={['seller']}><SellerProductList /></ProtectedRoute> />
22           <Route path="/addproduct"
23             element=<ProtectedRoute allowedRoles={['seller']}><AddProduct /></ProtectedRoute> />
24           <Route path="/edit/:id" element=<ProtectedRoute allowedRoles={['seller']}><EditProduct /></ProtectedRoute> />
25           <Route path="/addbalance" element=<ProtectedRoute allowedRoles={['seller','customer']}><AddBalance /></ProtectedRoute> />
26           <Route path="/register/customer" element=<RegisterCustomer /> />
27           <Route path="/register/seller" element=<RegisterSeller /> />
28           <Route path="/login" element=<Login /> />
29           <Route path="/" element=<ProtectedRoute allowedRoles={['seller','customer']}><ProductList/></ProtectedRoute> />
30         </Routes>
31       </Layout>
32     </Router>
33   );
34 }
35
36 export default App;
```

Frontend Layout.js

```
1 import React, { useEffect, useState } from "react";
2 import { Link, useNavigate } from "react-router-dom";
3 import { jwtDecode } from "jwt-decode";
4 import "../styles/Layout.css";
5
6 function Layout({ children }) {
7   const navigate = useNavigate();
8   const [userBalance, setUserBalance] = useState(0);
9   const [userRole, setUserRole] = useState(null);
10
11   useEffect(() => {
12     const token = localStorage.getItem("token");
13     if (token) {
14       try {
15         const decoded = jwtDecode(token);
16         setUserRole(decoded.role);
17
18         // ♦ Pozivamo backend da dohvatimo stvarni balans iz baze
19         fetch(`${process.env.REACT_APP_API_URL}/user/balance`, {
20           headers: { Authorization: `Bearer ${token}` },
21         })
22           .then((res) => res.json())
23           .then((data) => {
24             if (data && data.balance !== undefined) {
25               setUserBalance(data.balance);
26             }
27           })
28           .catch((err) => console.error("Greška pri dohvaćanju balansa:", err));
29       } catch (err) {
30         console.error("Nevažeći token");
31       }
32     }
33   }, []);
34
35   const handleLogout = () => {
36     localStorage.removeItem("token");
37     navigate("/login");
38     window.location.reload();
39   };
40 }
```



```

41     return (
42         <div className="layout">
43             <nav className="nav">
44                 <div className="nav-left">
45                     <Link to="/" className="nav-title">MSHOP</Link>
46                 </div>
47
48                 <div className="additional">
49                     {userRole && (
50                         <span className="nav-balance">{userBalance} RSD</span>
51                     )}
52                     {userRole && (
53                         <Link to="/addbalance" className="nav-link">Povećaj balans</Link>
54                     )}
55                     {userRole === "seller" && (
56                         <Link to="/addproduct" className="nav-link">Dodaj proizvod</Link>
57                     )}
58                     {userRole === "seller" && (
59                         <Link to="/seller/productlist" className="nav-link">Vasi proizvodi</Link>
60                     )}
61                     {userRole && (
62                         <button onClick={handleLogout} className="nav-link">Logout</button>
63                     )}
64                     {!userRole && (
65                         <Link to="/login" className="nav-link">LogIn</Link>
66                     )}
67                     {!userRole && (
68                         <Link to="/register/customer" className="nav-link">Registracija kupca</Link>
69                     )}
70                     {!userRole && (
71                         <Link to="/register/seller" className="nav-link">Registracija prodavca</Link>
72                     )}
73                 </div>
74             </nav>
75
76             <div className="content">{children}</div>
77         </div>
78     );
79 }
80 export default Layout;

```

ProductForm.js:

```
1 import React, { useState } from "react";
2 import '../styles/ProductForm.css';
3
4 function ProductForm({ initialData = {}, onSubmit }) {
5   const [name, setName] = useState(initialData.name || "");
6   const [description, setDescription] = useState(initialData.description || "");
7   const [price, setPrice] = useState(initialData.price || "");
8   const [image, setImage] = useState(null);
9
10  const handleSubmit = (e) => {
11    e.preventDefault();
12    onSubmit({ name, description, price: parseFloat(price), image });
13  };
14
15  const handleFileChange = (e) => {
16    if (e.target.files.length > 0) {
17      setImage(e.target.files[0]);
18    }
19  };
20
21  return (
22    <form onSubmit={handleSubmit}>
23      <div>
24        <label>Naziv:</label>
25        <input value={name} onChange={(e) => setName(e.target.value)} required />
26      </div>
27      <div>
28        <label>Opis:</label>
29        <input
30          value={description}
31          onChange={(e) => setDescription(e.target.value)}
32          required
33        />
34      </div>
35      <div>
36        <label>Cena:</label>
37        <input
38          type="number"
39          value={price}
40          onChange={(e) => setPrice(e.target.value)}
41          required
42        />
43      </div>
44      <div>
45        <label>Slika:</label>
46        <input type="file" accept="image/*" onChange={handleFileChange} />
47      </div>
48      <button type="submit" id="save">Sačuvaj</button>
49    </form>
50  );
51 }
52
53 export default ProductForm;
54
```

ProtectedRoute.js

```
1  import React from "react";
2  import { Navigate } from "react-router-dom";
3  import { jwtDecode } from "jwt-decode";
4
5  function ProtectedRoute({ children, allowedRoles }) {
6      const token = localStorage.getItem("token");
7
8      if (!token) {
9          return <Navigate to="/login" />;
10     }
11
12     let user;
13     try {
14         user = jwtDecode(token);
15     } catch {
16         return <Navigate to="/login" />;
17     }
18
19     if (!allowedRoles.includes(user.role)) {
20         return <Navigate to="/login" />;
21     }
22
23     return children;
24 }
25
26 export default ProtectedRoute;
27
```

RegisterForm.js

```
1 import React, { useState } from "react";
2 import { useNavigate } from "react-router-dom";
3
4 function RegisterForm({ role }) {
5   const [name, setName] = useState("");
6   const [email, setEmail] = useState("");
7   const [password, setPassword] = useState("");
8   const [message, setMessage] = useState("");
9   const navigate = useNavigate();
10
11   const handleRegister = async (e) => {
12     e.preventDefault();
13
14     try {
15       const res = await fetch(`${process.env.REACT_APP_API_URL}/users/register`, {
16         method: "POST",
17         headers: { "Content-Type": "application/json" },
18         body: JSON.stringify({ name, email, password, role }),
19       });
20
21       const data = await res.json();
22
23       if (res.ok) {
24         setMessage(data.message);
25         navigate("/login");
26       } else {
27         setMessage(data.error);
28       }
29     } catch (err) {
30       console.error(err);
31       setMessage("Greška servera");
32     }
33   };
34
35   return (
36     <div>
37       <h2>Registracija {role === "seller" ? "prodavca" : "kupca"}</h2>
38       <form onSubmit={handleRegister}>
39         <div>
40           <label>Ime:</label>
41           <input value={name} onChange={(e) => setName(e.target.value)} required />
42         </div>
43         <div>
44           <label>Email:</label>
45           <input type="email" value={email} onChange={(e) => setEmail(e.target.value)} required />
46         </div>
47         <div>
48           <label>Lozinka:</label>
49           <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} required />
50         </div>
51         <button type="submit">Registruj se</button>
52       </form>
53       {message && <p>{message}</p>}
54     </div>
55   );
56 }
57
58 export default RegisterForm;
59
```

Docker-compose.yml

```
1  version: '3.9'
2
3  name: webshop
4
5  > Run All Services
6  services:
7    > Run Service
8    db:
9      image: mysql:8.0
10     container_name: miniweb_db
11     restart: always
12     environment:
13       MYSQL_ROOT_PASSWORD: rootpass
14       MYSQL_DATABASE: mini_webshop
15       MYSQL_USER: webshop
16       MYSQL_PASSWORD: webshoppass
17     ports:
18       - "3306:3306"
19     volumes:
20       - db_data:/var/lib/mysql
21       - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
22     healthcheck:
23       test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
24       interval: 8s
25       timeout: 8s
26       retries: 12
27
28     > Run Service
29    backend:
30      build: ./backend
31      container_name: miniweb_backend
32      restart: always
33      ports:
34        - "3000:3000"
35      environment:
36        DB_HOST: db
37        DB_USER: webshop
38        DB_PASSWORD: webshoppass
39        DB_NAME: mini_webshop
40        DB_PORT: 3306
41      volumes:
42        - ./backend:/app
43        - /app/node_modules
44        - ./backend/uploads:/app/uploads
45      depends_on:
46        db:
47          condition: service_healthy
48
49    > Run Service
50    frontend:
51      build: ./frontend
52      container_name: miniweb_frontend
53      restart: always
54      ports:
55        - "3001:3000"
56      volumes:
57        - ./frontend:/app
58        - /app/node_modules
59      environment:
60        REACT_APP_API_URL=http://localhost:3000
61
62      depends_on:
63        - backend
64
65    volumes:
66      db_data:
67
68
69    networks:
70      webshop_network:
71      driver: bridge
72
73
```

-name: webshop Definiše ime čitavog Docker Compose projekta

-Definisali smo tri servisa(kontejnjera koji zajedno čine aplikaciju):

- db
- backend
- frontend

-U db servisu preko image smo kao osnovu projekta definisali mysql verzije 8

-U svakom servisu smo definisali container_name tj. ime za svaki kontejnjer

-U db servisu preko restart: always podešavamo da se kontejnjer svaki put restartuje kada se zaustavi zbog neke greške

-U environment postavkama se nalaze varijable koje koristi sql docker slika da bi se baza pravilno podesila pri prvom pokretanju

-Preko port u svakom servisu povezujemo port lokalnog računara i kontejnjera datog servisa

-Za db servis imamo jedan named volumen db_data koji cuva podatke baze podataka i jedan bind mount volume do init.sql fajla

-Pošto backend zavisi od db servisa napravio sam healthcheck koji proverava da li je db servis pokrenut bez problema

-U backend i frontend servisu koristimo build koji sa prosledjenom folderom servisa govori da se na osnovu Dockerfile u tom folderu napravi image za budući kontejnjer

-U backend servisu environment sadrži promenljive koje Node.js backend koristi da zna kako da se poveže na bazu podataka

-Za backend servis imamo bind mount volumen celog backenda kako bi odmah primenjivali izmene u kodu bez ponovnog buildovanja, imamo jedan anonimni volumen kako bi smo sačuvali node_modules

koje prethodni `bind_mount` pregazi I imamo još jedan `bind mount` koji služi da sve slike sačuvane u `uploads` budu sačuvane I nakon završetka rada

- Postavljamo da backend servis zavisi od db servisa I to pod uslovom da db servis posalje povratnu informaciju da je healthy

- Za frontend servis imamo jedan `bind mount` volumen koji omogućava da uživo vidimo sve promene u kodu bez ponovnog builda I imamo jedan anonimni volume koji čuva `node_modules` koje bi `bind mount` ne bi pregazio

- Preko `environment` u frontend servisu postavljamo promenljivu okruženja koja definiše putanju do backend servera

- Posto u frontendu imamo `depends_on` ali bez nekog određenog uslova onda se samo označava da docker compose prvo pokuša da pokrene backend pa tek onda frontend

- Nakon servisa definisao sam jedan `named` volumen koji sam koristio u db servisu I jedan `network` sa `bridge` tipom koji sam dodao svim servisima

Uspesno pokretanje docker-compose

```
mihajlo@DESKTOP-KLFAORV:~/projekatcloud/v3/probacloud$ sudo docker compose up
WARN[0000] /home/mihajlo/projekatcloud/v3/probacloud/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[*] Running 12/12
✓ db Pulled
[*] Building 3.7s (21/21) FINISHED
miniweb_frontend | > frontend@0.1.0 start
miniweb_frontend | > react-scripts start
miniweb_frontend |
miniweb_backend | [nodemon] 3.1.10
miniweb_backend | [nodemon] to restart at any time, enter `rs`
miniweb_backend | [nodemon] watching path(s): *.*
miniweb_backend | [nodemon] watching extensions: js,mjs,cjs,json
miniweb_backend | [nodemon] starting `node server.js`
miniweb_backend | [dotenv@17.2.3] injecting env (0) from .env -- tip: 🔍 add access controls to secret
miniweb_backend | Server je pokrenut na http://localhost:3000
miniweb_backend | Uspesno povezan na bazu!
miniweb_frontend | (node:26) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: `on
miniweb_frontend | (Use `node --trace-deprecation ...` to show where the warning was created)
miniweb_frontend | (node:26) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: `o
miniweb_frontend | Starting the development server...
miniweb_frontend |
miniweb_frontend | Compiled successfully!
miniweb_frontend |
miniweb_frontend | You can now view frontend in the browser.
miniweb_frontend |
miniweb_frontend | Local: http://localhost:3000
miniweb_frontend | On Your Network: http://172.18.0.4:3000
miniweb_frontend |
miniweb_frontend | Note that the development build is not optimized.
miniweb_frontend | To create a production build, use npm run build.
miniweb_frontend |
miniweb_frontend | webpack compiled successfully

mihajlo@DESKTOP-KLFAORV:~/projekatcloud/v3/probacloud$ sudo docker ps
[sudo] password for mihajlo:
CONTAINER ID   IMAGE             COMMAND                  CREATED        STATUS        PORTS                                                                 NAMES
80295aa3ec8f   webshop-frontend  "docker-entrypoint.s..." 3 minutes ago  Up 17 seconds  3001/tcp, 0.0.0.0:3001->3000/tcp, [::]:3001->3000/tcp  miniweb_frontend
6b4bc89d50c0   webshop-backend  "docker-entrypoint.s..." 3 minutes ago  Up 17 seconds  0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp  miniweb_backend
6948502c547b   mysql:8.0        "docker-entrypoint.s..." 3 minutes ago  Up 24 seconds (healthy)  0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp  miniweb_db

mihajlo@DESKTOP-KLFAORV:~/projekatcloud/v3/probacloud$ sudo docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
webshop-frontend  latest   89e5f37814d4   3 minutes ago  1.39GB
webshop-backend  latest   45fbb530289d   3 minutes ago  1.13GB
fav-img         1        6d2b412ace09   3 weeks ago    1.16GB
app             1        6fe7ca243375   3 weeks ago    1.16GB
mysql           8.0      94753e67a0a9   3 weeks ago    780MB
hello-world     latest   1b44b5a3e06a   2 months ago   10.1kB

mihajlo@DESKTOP-KLFAORV:~/projekatcloud/v3/probacloud$ sudo docker volume ls
DRIVER      VOLUME NAME
local       aabb34b02f38b9b3cd5d60c9a88ac63108ffc8a50a4e8ea61307021ebaef5e0b
local       f5a70129450c7966dd86c3e96d2ab78d68f47d32b336825e3a378362f50848a
local       webshop_db_data

mihajlo@DESKTOP-KLFAORV:~/projekatcloud/v3/probacloud$ sudo docker network ls
docker: unknown command: docker network

Run 'docker --help' for more information

mihajlo@DESKTOP-KLFAORV:~/projekatcloud/v3/probacloud$ sudo docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
ac1f934d614d   bridge    bridge    local
1ae295af3260   host      host      local
62f2c1104bcd   none      null      local
afc7734de730   v2_default bridge    local
f29517b946c5   webshop_webshop_network bridge     local
mihajlo@DESKTOP-KLFAORV:~/projekatcloud/v3/probacloud$
```