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:

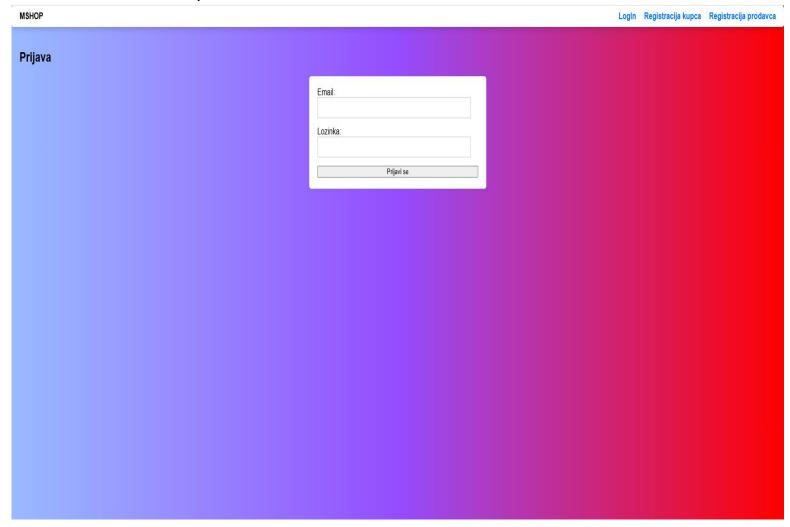
- react-router-dom za upravljanje navigacijom između stranica,
- axios za komunikaciju sa backend API-jem,
- jwt-decode za dekodiranje JWT tokena i proveru autentifikacije korisnika,

Struktura frontend projekta uključuje:

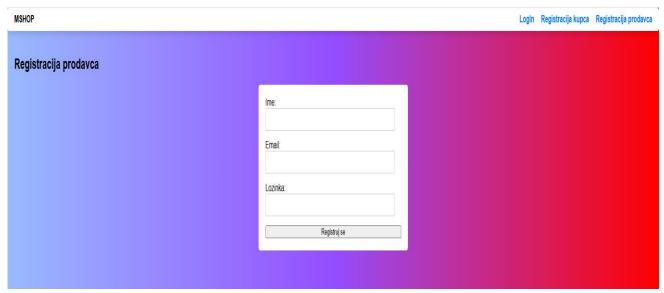
- pages stranice aplikacije, npr. lista proizvoda, prijava i registracija,
- styles CSS fajlove za stilizaciju aplikacije,
- App.js glavnu komponentu koja povezuje sve stranice i rute,
- index.**js** ulaznu tačku aplikacije koja pokreće React i povezuje ga sa HTML dokumentom.

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

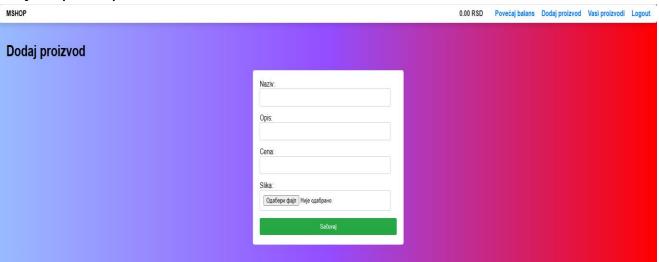
Korisničko uputstvo



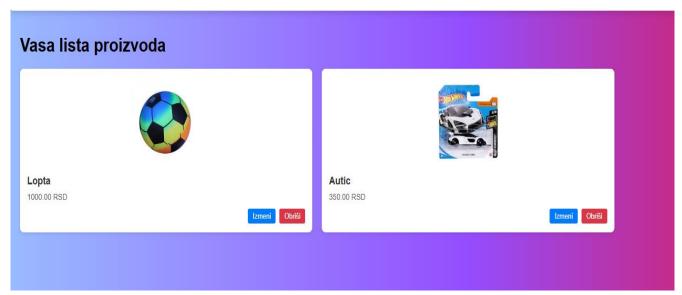
Svaki neulogovani korisnik koji pokuša da pristupi početnoj stranici (/) ili bilo kojoj drugoj zaštićenoj ruti automatski se preusmerava na stranicu za prijavu. Sa te stranice korisnik može da ode na rutu za registraciju kupca ili na rutu za registraciju prodavca, u zavisnosti od toga koju vrstu naloga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



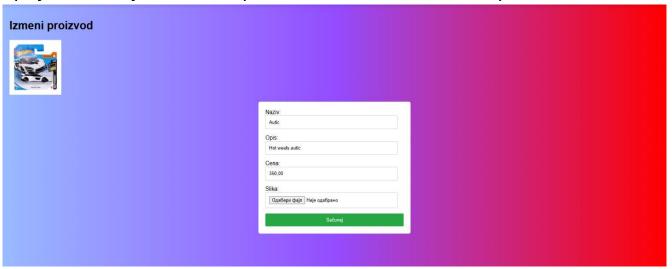
Registracija kupca I prodavca su slične, backend sam u pozadini odredjuje rolu koju će dodeliti novom korisniku u zavisnosti od rute sa koje e pristupio



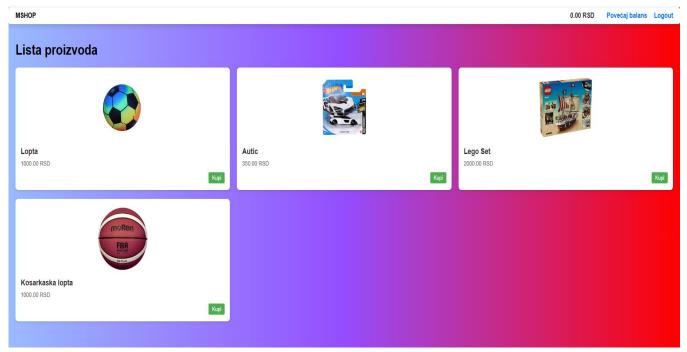
Ulogovani prodavci mogu dodavati svoje proizvode, pri cemu je potrebno dodati naziv, opis, cenu i sliku proizvoda pri čemu backend automatski čuva ko je vlasnik proizvoda



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

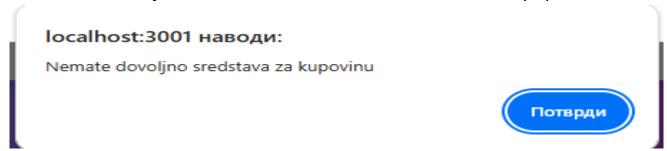


Pri izmeni proizvoda ima vide se predjašnji podaci i po potrebi mogu da se unesu novi

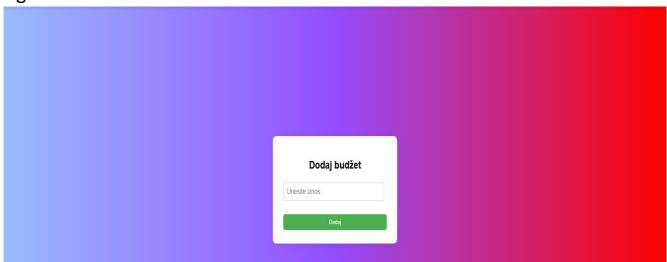


Kupac ima mogućnost da vidi proizvode svih prodavaca I ukoliko želi može da ih kupi.

Ako ne ma dovoljno sredstava na računu izaćiće mu sledeći popaut



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



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

Prikaz reprezentativnih delova koda

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

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

Glavni fajl db servisa

Server.js backend rute:

```
app.get('/products', (req, res) => {
  const query = 'SELECT'* FROM products';
57
58
       db.query(query, (err, results) => {
59
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
     app.get('/seller/products', (req, res) => {
66
       const authHeader = req.headers.authorization;
67
        if (!authHeader) return res.status(401).json({ error: 'Niste autorizovani' });
68
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);
         catch (err) {
76
77
         return res.status(401).json({ error: 'Nevažeći token' });
78
79
       const query = 'SELECT * FROM products WHERE user_id = ?';
       db.query(query, [decoded.id], (err, results) => {
80
         if (err) return res.status(500).json({ error: 'Greška servera' });
81
82
         res.json(results);
83
       });
84
     });
```

```
// Dodavanje proizvoda za user_id
      app.post('/products', upload.single('image'), (req, res) => {
87
        const { name, description, price, user_id } = req.body;
88
89
        const imagePath = req.file ? `uploads/${req.file.filename}` : null;
90
91
        | return res.status(400).json({ error: "Nije prosleđen user_id" });
92
93
94
95
        const query = 'INSERT INTO products (name, description, price, image_path, user_id) VALUES (?, ?, ?, ?) ';
        db.query(query, [name, description, price, imagePath, user_id], (err, result) => {
96
97
          if (err) return res.status(500).json({ error: 'Greška servera' });
         res.json({ message: 'Proizvod uspešno dodat!', productId: result.insertId });
98
        });
99
100
      });
101
102
      // Ažuriranje proizvoda sa opcionalnom slikom
103
      app.put('/products/:id', upload.single('image'), (req, res) => {
        const { id } = req.params;
104
105
        const { name, description, price } = req.body;
106
107
        if (req.file) {
108
          const imagePath = `uploads/${req.file.filename}`;
109
          const query = 'UPDATE products SET name=?, description=?, price=?, image_path=? WHERE id=?';
          db.query(query, [name, description, price, imagePath, id], (err, result) => {
110
           if (err) return res.status(500).json({ error: 'Greška servera' });
res.json({ message: 'Proizvod uspešno ažuriran sa slikom!' });
111
112
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
```

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

```
144
      // Registracija korisnika (customer ili seller)
      app.post('/users/register', async (req, res) => {
145
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) {
            return res.status(400).json({ error: 'Korisnik već postoji' });
159
160
161
162
            // Šifrujemo lozinku
163
            const hashedPassword = await bcrypt.hash(password, 10);
164
            // role može biti 'seller' ili 'customer' (ako nije poslato, default je 'customer')
165
            const userRole = role === 'seller' ? 'seller' : 'customer';
166
167
168
            // Ubacujemo korisnika u bazu
169
            db.query(
170
              'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
              [name, email, hashedPassword, userRole],
171
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},
        SECRET_KEY,
212
213
      { expiresIn: '1h' } // token važi 1 sat
214
      );
215
216
      res.json({
217
        message: 'Uspešno prijavljen!',
218
        token,
219
        user: {
          id: user.id,
220
221
          name: user.name,
222
          email: user.email,
223
         role: user.role,
224
        },
225
        });
226
        });
227
      });
```

```
app.post('/addbalance', (req, res) => {
229
        const authHeader = req.headers.authorization;
        if (!authHeader) return res.status(401).json({ error: 'Niste autorizovani' });
230
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
        const { amount } = req.body;
if (!amount || isNaN(amount)) {
242
243
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' });
          res.json({ message: 'Balance uspešno ažuriran.' });
250
251
        });
252
      });
253
254
      // Ruta za dohvatanje 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) {
            console.error('Greška prilikom dohvatanja balansa:', err);
272
273
            return res.status(500).json({ error: 'Greška servera' });
274
275
276
          if (result.length === 0) {
          return res.status(404).json({ error: 'Korisnik nije pronađen' });
277
278
279
280
          res.json({ balance: result[0].balance });
281
        });
282
```

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

Frontend App.js:

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import ProductList from "./pages/ProductList";
import SellerProductList from "./pages/SellerProductList";
        import AddProduct from "./pages/AddProduct";
import EditProduct from "./pages/EditProduct";
import RegisterCustomer from "./pages/RegisterCustomer";
import RegisterSeller from "./pages/RegisterSeller";
         import Login from "./pages/Login";
import Layout from "./components/Layout";
import ProtectedRoute from "./components/ProtectedRoute";
import AddBalance from "./pages/AddBalance";
15
          function App() {
16
17
18
                      <Layout>
                         <Routes>
19
20
21
23
24
25
26
27
28
29
30
31
32
                             <Route path="/seller/productlist"</pre>
                             element={<ProtectedRoute allowedRoles={['seller']}><SellerProductList /></ProtectedRoute>} />
                             <Route path="/addproduct
                             element={<ProtectedRoute allowedRoles={['seller']}><AddProduct /></ProtectedRoute>}/>
                             <Route path="/edit/:id" element={<ProtectedRoute allowedRoles={['seller']}><EditProduct /></ProtectedRoute>} />
<Route path="/addbalance" element={<ProtectedRoute allowedRoles={['seller','customer']}><AddBalance /></ProtectedRoute>} />

«Route path="/register/customer" element={<RegisterCustomer />} />

«Route path="/register/seller" element={<RegisterSeller />} />

«Route path="/login" element={<Login />} />

«Route path="/login" element={<Login />} />

«Route path="/" element={<ProtectedRoute allowedRoles={['seller','customer']}><ProductList/>//ProtectedRoute>}/>

                          </Routes>
                      </Layout>
                   (/Router>
33
34
         export default App;
```

Frontend Layout.js

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

ProductForm.js:

```
import React, { useState } from "react";
2
     import '../styles/ProductForm.css';
4
     function ProductForm({ initialData = {}, onSubmit }) {
  const [name, setName] = useState(initialData.name || "");
5
       const [description, setDescription] = useState(initialData.description || "");
const [price, setPrice] = useState(initialData.price || "");
6
7
        const [image, setImage] = useState(null);
8
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
24
              <label>Naziv:</label>
25
              <input value={name} onChange={(e) => setName(e.target.value)} required />
26
            </div>
27
28
              <label>Opis:</label>
29
              <input</pre>
30
                value={description}
31
                 onChange={(e) => setDescription(e.target.value)}
32
33
34
35
36
              <label>Cena:</label>
37
              <input
38
                 type="number"
39
                 value={price}
40
                 onChange={(e) => setPrice(e.target.value)}
41
42
43
            </div>
44
45
              <label>Slika:</label>
46
              <input type="file" accept="image/*" onChange={handleFileChange} />
47
48
             </div>
            <button type="submit" id="save">Sačuvaj</button>
49
          </form>
50
        );
51
52
53
     export default ProductForm;
```

ProtectedRoute.js

```
import React from "react";
import { Navigate } from "react-router-dom";
2
     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)) {
       return <Navigate to="/login" />;
20
21
22
23 🖁
      return children;
     Ħ
24
25
26
     export default ProtectedRoute;
27
```

RegisterForm.js

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
 2
     function RegisterForm({ role }) {
   const [name, setName] = useState("");
4
 6
       const [email, setEmail] = useState("
       const [password, setPassword] = useState("");
 7
8
       const [message, setMessage] = useState("");
 9 000
       const navigate = useNavigate();
10
11
        const handleRegister = async (e) => {
12
         e.preventDefault();
13
14
15
           const res = await fetch(`${process.env.REACT_APP_API_URL}/users/register`, {
              method: "POST",
headers: { "Content-Type": "application/json" },
16
17
18
              body: JSON.stringify({ name, email, password, role }),
19
            });
20
21
            const data = await res.json();
22
23
24
            if (res.ok) {
              setMessage(data.message);
              navigate("/login");
25
26
            } else {
   setMessage(data.error);
27
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
40
41
                <label>Ime:</label>
                <input value={name} onChange={(e) => setName(e.target.value)} required />
42
43
44
               <label>Email:</label>
45
                <input type="email" value={email} onChange={(e) => setEmail(e.target.value)} required />
46
47
48
49
50
                <label>Lozinka:</label>
                <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} required />
51
52
              <button type="submit">Registruj se</button>
53
54
55
            {message && {message}}
        );
56
57
      export default RegisterForm;
```

Docker-compose.yml

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

- -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 healtcheck 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 kondu 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 healty
- -Za frontend servis imamo jedan bind mount volumen koji omogućava da uživo vidimo sve pormene 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 odredjenog uslova onda se samo oznacava da docker compose prvo pokusa 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

```
:~/projekatcloud/v3/probacloud$ sudo docker compose up
  N|[9990] /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
miniweb_frontend
                            > frontend@0.1.0 start
 miniweb_frontend
                           > react-scripts start
 miniweb_frontend
 miniweb backend
                           [nodemon] 3.1.10
                            [nodemon] to restart at any time, enter `rs`
 miniweb backend
 miniweb_backend
                            [nodemon] watching path(s): *.*
                            [nodemon] watching extensions: js,mjs,cjs,json
 miniweb backend
                            [nodemon] starting `node server
 miniweb_backend
                            [dotenv@17.2.3] injecting env (0) from .env -- tip: <a> add access controls to secret</a>
 miniweb_backend
 miniweb backend
                            Server je pokrenut na http://localhost:3000
 miniweb backend
                           Uspešno 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
                                                      http://localhost:3000
                              Local:
 miniweb_frontend
miniweb_frontend
                              On Your Network: http://172.18.0.4:3000
 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
 ihajlo@DESKTOP-KLFADRV:~/projekatcloud/v3/probacloud$ sudo docker ps
[sudo] password for mihajlo:
CONTAINER ID IMAGE
                                             CREATED
                                                         STATUS
                          COMMAND
                                                                                                                          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-KLFADRV:~/projekatcloud/v3/probacloud$ sudo docker image ls
                                 CREATED
REPOSITORY
              TAG
                      IMAGE ID
                                              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
                      6fe7ca243375 3 weeks ago
                                              1.16GB
app
                      94753e67a0a9
              8.0
                                 3 weeks ago
mvsal
hello-world
              latest
                      1b44b5a3e06a 2 months ago
mihajlo@DESKTOP-KLFADRV:~/projekatcloud/v3/probacloud$ sudo docker volume ls
local
       aabb34b02f38b9b3cd5d60c9a88ac63108ffc8a50a4e8ea61307021ebaef5e0b
local
       f5a70129450c7966dd86c3e96d2ab78d68f47d32bf336825e3a378362f50848a
local
      webshop_db_data
     @DESKTOP-KLFADRV:~/projekatcloud/v3/probacloud$ sudo docker netwoek ls
docker: unknown command: docker netwoek
Run 'docker --help' for more information
nihajlo@DESKTOP-KLFADRV:~/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
 ihajlo@DESKTOP-KLFADRV:~/projekatcloud/v3/probacloud$
```