Zsigó Bence AGQU01

Webtechnológiák 2.
beadandó
2024.

Frontend

App.js

Login Component

- A `Login` komponens egy funkcionális komponens, ami a bejelentkezési űrlapot és hozzá tartozó logikát kezeli.
- A `useState` hook-al követi a `username` és `password` állapotokat.

```
const [username, setUsername] = useState('');
const [password, setPassword] = useState('');
```

- Az `axios` könyvtár segítségével küld HTTP POST kérést a szerver `http://localhost:3001/login` végpontjára a felhasználónév és jelszó megadásával.
- Sikeres bejelentkezés esetén beállítja a `loggedIn` állapotot `true`-ra, eltárolja a bejelentkezett felhasználó nevét a `localStorage`-ban (`localStorage.setItem('user', username)`), majd átirányít a `Mainpage` komponensre (`window.location.href = '/Mainpage';`).
- Hiba esetén megjeleníti az alertet a megfelelő hibaüzenettel (`alert('Hibás felhasználónév vagy jelszó');` vagy `alert('Hiba történt a bejelentkezés során!');`).

```
const handleLogin = async () => {
  try {
    await axios.post('http://localhost:3001/login', { username, password });
    setLoggedIn(true);
    localStorage.setItem('loggedIn', 'true');
    localStorage.setItem('user', username);
    window.location.href = '/Mainpage';
  } catch (error) {
    if (error.response && error.response.status === 401) {
      alert('Hibás felhasználónév vagy jelszó');
    } else {
    console.error('Hiba:', error);
    alert('Hiba történt a bejelentkezés során!');
    }
  }
};
```

App Component

- Az `App` komponens a fő komponens, amely az alkalmazás útvonalait definiálja a `Routes` komponens segítségével.
- A `loggedIn` állapotot `useState` hook-al kezeli, alapértelmezetten `false` értékkel.
- A `useEffect` hook segítségével ellenőrzi a `localStorage`-ban tárolt `loggedIn` állapotot az alkalmazás betöltésekor.
    - Ha be van jelentkezve (`loggedInStatus === 'true'`), akkor beállítja a `loggedIn` állapotot `true`-ra.
    - Ha nincs bejelentkezve, de az URL `/Mainpage`-re mutat, akkor visszairányít a bejelentkező oldalra (`navigate('/Login');`).

```
useEffect(() => {
  const loggedInStatus = localStorage.getItem('loggedIn');
  if (loggedInStatus === 'true') {
    setLoggedIn(true);
  } else if(window.location.pathname == "/Mainpage"){
    navigate('/Login');
  }
}, [navigate]);
```

- A `Routes` komponensben definiálja az alkalmazás útvonalait és hozzárendeli a megfelelő komponenseket:
    - `/` vagy `/Login`: `Login` komponens jelenik meg.
    - `/Mainpage`: `Mainpage` komponens jelenik meg, ha be van jelentkezve (`loggedIn === true`).
    - `/Register`: `Register` komponens jelenik meg.

```
<Routes>
  <Route path="/" element={<Login setLoggedIn={setLoggedIn} />} />
  <Route path="/Login" element={<Login setLoggedIn={setLoggedIn} />} />
  <Route path="/Mainpage" element={loggedIn ? <Mainpage /> : null} />
  <Route path="/Register" element={<Register />} />
</Routes>
```

- Az `onClick` eseménykezelő hívja a `handleRegister` függvényt, ami elvégzi a regisztrációs kérést a szerver felé.
- Az állapotok (`username`, `password`, `password2`) `input` mezőkön keresztül frissülnek az `onChange` események segítségével.
- A komponens `class` attribútumokat használ a stílusok és CSS osztályok beállítására.

Mainpage.js

- A **`Mainpage`** komponens egy funkcionális komponens, amely a felszerelések kezeléséhez kapcsolódó funkcionalitást valósítja meg.
- A komponens több állapotot is követ (**`useState`** hook), például **`products`** az elérhető felszerelések tárolására, **`newProduct`** az új felszerelés adatainak tárolására, valamint **`errors`** az érvényesítési hibák kezelésére.
- Az **`useEffect`** hook segítségével az oldal betöltésekor lefuttatja a **`fetchProducts`** függvényt, amely lekéri az elérhető felszereléseket a szerverről (**`axios.get('http://localhost:3001/equipments')`**).

```
useEffect(() => {
  fetchProducts();
}, []);

const fetchProducts = async () => {
  try {
    const response = await axios.get('http://localhost:3001/equipments');
    setProducts(response.data.data);
    console.log(response);
  } catch (error) {
    console.error('Error fetching products:', error);
  }
};
```

- A **`handleInputChange`** függvény figyeli az input mezők változásait, és frissíti az **`newProduct`** állapotot.
- A **`validateProduct`** függvény érvényesíti az új felszerelés adatait, és visszaadja a validációs hibákat.
- Az **`addProduct`** függvény hozzáadja az új felszerelést a szerverhez, először validálja az adatokat (**`validateProduct`**), majd küldi a kérést (**`axios.post('http://localhost:3001/equipments', newProduct)`**).

```
const addProduct = async () => {
  const validationErrors = validateProduct();
  if (Object.keys(validationErrors).length === 0) {
    try {
      await axios.post('http://localhost:3001/equipments', newProduct);

      setNewProduct({
        name: '',
        equipment: '',
        quantity: ''
      });
      fetchProducts();
    } catch (error) {
      console.error('Error adding product:', error);
    }
  } else {
    setErrors(validationErrors);
  }
};
```

- A `handleDelete` függvény törli a kiválasztott felszerelést a szerverről az ID alapján (`axios.post('http://localhost:3001/delete', { userid: id })`), majd frissíti a felszerelések listáját (`fetchProducts()`).

```
const handleDelete = async (id) => {
  if (window.confirm(`Biztosan törölni szeretnéd?`)) {
    await axios.post('http://localhost:3001/delete', {userid: id});
  }
    fetchProducts();

};
```

- A `handleLogout` függvény kijelentkezteti a felhasználót (`localStorage.setItem('loggedIn', 'false')`).
- Az állapotok (`useState` hook) segítségével követi a komponens adatait és frissíti azokat.
- Az `axios` modult használja az aszinkron adatkérésekhez a szerverrel történő kommunikációra (`axios.get`, `axios.post`).
- A `Link` komponens (`react-router-dom` modul) lehetővé teszi a navigációt más oldalakra.
- Az `onClick` eseménykezelők hívják a megfelelő függvényeket (`handleInputChange`, `addProduct`, `handleDelete`, `handleLogout`) a felhasználói interakciók kezelésére.
- A komponens rendereli a felszerelések listáját (`products.map`), és lehetővé teszi az új felszerelés hozzáadását és a meglévők törlését.

Register.js

Állapotok inicializálása:

- `useState` hook-ot használunk az állapotok (`username`, `password`, `password2`) kezelésére, amelyek a felhasználói input mezők értékeit tárolják.

```
const navigate = useNavigate();
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [password2, setPassword2] = useState('');
```

Regisztráció kezelése (`handleRegister` függvény):

- Az `axios.post` függvény segítségével küldünk egy POST kérést a szervernek a regisztrációhoz.
- A kérés tartalmazza a felhasználónév (`username`), jelszó (`password`) és megerősítő jelszó (`password2`) adatokat.
- Az aszinkron kérés eredményére reagálunk: sikeres regisztráció esetén visszairányítjuk a felhasználót a bejelentkezési oldalra (`navigate('/Login')`), különben megjelenítjük a hibát.

```
const handleRegister = async () => {
    try {
        await axios.post('http://localhost:3001/register', { username, password, password2 });
        alert('Sikeres regisztráció!');
        navigate('/Login');
      } catch (error) {
        if (error.response && error.response.status === 400) {
          alert('A megadott jelszavak nem egyeznek!');
        } else if (error.response && error.response.status === 409) {
          alert('A felhasználónév már foglalt!');
        } else {
          console.error('Hiba:', error);
          alert('Hiba történt a regisztráció során!');
        }
      }
};
```

Input mezők kezelése (`onChange` esemény):

- Az `onChange` eseményekre (amikor az input mező tartalma változik) frissítjük az állapotokat (`setUsername`, `setPassword`, `setPassword2`) az új értékekkel.

Felhasználói felület (`JSX`):

- A `return` részben JSX-et használunk a komponens felhasználói felületének definiálására.
- Megjelenítjük a regisztrációs űrlapot, amely tartalmazza a felhasználónév, jelszó és jelszó megerősítő mezőket, valamint a regisztráció és vissza a bejelentkezéshez gombot (`<button>`).

- A `Link` komponens segítségével a vissza a bejelentkezéshez gombot beágyazzuk a React Router `Link`-jébe, amely a bejelentkezési oldalra irányít vissza.

index.js

- A `BrowserRouter` használata biztosítja a React Router működését az alkalmazásban.
- A `Login` komponens a gyökérkomponens, amely a `Router`-en belül rendeli az útvonalakhoz.
- A `createRoot` segítségével létrehoztam a gyökér renderelési kontextust a megadott HTML elem (`rootElement`) alapján.
- A `root.render()` függvényével rendereltem a gyökér elemre a `<Router>` komponenssel körbevett `<Login>` komponenst.
- A `<React.StrictMode>` biztosítja a szigorú üzemmódot az alkalmazás számára, ami segít kiemelni a lehetséges problémákat és hibákat a komponensekben.

```
const rootElement = document.getElementById('root');
const root = createRoot(rootElement);

root.render(
  <Router>
    <Login />
  </Router>
);


reportWebVitals();
```

Backend

server.js

Ez a Node.js alkalmazás egy egyszerű szerveroldali API-t valósít meg, amely lehetővé teszi felhasználók bejelentkezését, regisztrációját, valamint különböző felszerelések (equipments) kezelését adatbázis segítségével. Az alkalmazás a `express`, `body-parser`, `cors` és egy adatbáziskezelő modul (pl. MongoDB) segítségével működik.

Alapvető beállítások és modulok importálása:

- `express`: HTTP kérések kezelésére.
- `body-parser`: HTTP kérés testének feldolgozására.
- `cors`: Kérési eredet szabályozására.

Alapvető konfiguráció:

- A szerver az `express()` függvényének meghívásával inicializálódik.
- Beállításra kerül a `body-parser` JSON adatok kezelésére.
- Engedélyezi a CORS-t a kliens oldali kérések fogadására.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const cors = require('cors');
const port = 3001;
const { EquipmentCollection, UserCollection } = require("./db");
app.use(cors());
app.use(bodyParser.json());
```

Bejelentkezés (`/login` végpont):

- A felhasználónevet és jelszót az adatküldésből kinyerjük (`req.body`).
- Megkeressük az adatbázisban a felhasználót a megadott felhasználónév alapján.
- Ellenőrizzük a jelszót.
- Ha a bejelentkezés sikeres, visszaadjuk a sikeres üzenetet (`{ success: true, message: 'Sikeres bejelentkezés' }`).
- Hibás felhasználónév vagy jelszó esetén megfelelő hibakóddal és üzenettel válaszolunk.

```
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {

    const user = await UserCollection.findOne({ username: username });
    if (!user) {
      return res.status(401).json({ success: false, message: 'Hibás felhasználónév vagy jelszó' });
    }

    if (user.password != password) {
      return res.status(401).json({ success: false, message: 'Hibás felhasználónév vagy jelszó' });
    }

    res.json({ success: true, message: 'Sikeres bejelentkezés' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ success: false, message: 'Valami hiba történt a szerver oldalán' });
  }
});
```

Regisztráció (`/register` végpont):

- Fogadjuk a regisztrációs adatokat (`username`, `password`, `password2`).
- Ellenőrizzük a jelszavak egyezőségét.
- Ellenőrizzük, hogy a felhasználónév már létezik-e az adatbázisban.
- Ha minden ellenőrzés sikeres, létrehozunk egy új felhasználót az adatbázisban és visszaadjuk a sikeres regisztrációs üzenetet.

```
app.post('/register', async (req, res) => {

  try {
    const { username, password, password2 } = req.body;

    if (password !== password2) {
      return res.status(400).json({ message: 'A megadott jelszavak nem egyeznek!' });
    }

    const user = await UserCollection.findOne({ username });
    if (user) {
      return res.status(400).json({ message: 'A felhasználónév már foglalt!' });
    }

    const newUser = new UserCollection({ username, password });
    await newUser.save();
    res.status(201).json({ message: 'Sikeres regisztráció!' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Valami hiba történt a szerver oldalán.' });
  }
});
```

Felszerelések kezelése (`/equipments` végpontok):

- **`POST /equipments`**: Fogadja az új felszerelés adatait, majd menti azokat az adatbázisba.
- **`GET /equipments`**: Visszaadja az összes felszerelést az adatbázisból.

Felszerelés törlése (`/delete` végpont):

- **`POST /delete`**: Törli a megadott felszerelést az azonosítója alapján az adatbázisból.

```
app.post('/delete', async (req, res) => {
  const { userid } = req.body;
  try {
    const result = await EquipmentCollection.deleteOne({ _id: userid });
    if (result.deletedCount === 1) {
      res.json({ success: true, message: 'Termék sikeresen törölve.' });
    } else {
      res.status(404).json({ success: false, message: 'A termék nem található.' });}
  } catch (error) {
    console.error('Hiba történt a törlés során:', error);
    res.status(500).json({ error: 'Hiba történt a törlés során.' });
  }
});
```

db.js

1. **Adatbázis Kapcsolat létrehozása:**
   - A **`mongoose.createConnection`** segítségével hozol létre kapcsolatot a MongoDB adatbázisokkal.
   - Az első kapcsolat létrehozása a **`equipments`** adatbázissal, a második a **`user`** adatbázissal.

```
const mongoose = require('mongoose');



const equipments = mongoose.createConnection("mongodb://localhost:27017/equipments");
equipments.on('error', console.error.bind(console, 'equipments connection error:'));
equipments.once('open', () => {
  console.log('equipments connected');
});

const userDB = mongoose.createConnection("mongodb://localhost:27017/user");
userDB.on('error', console.error.bind(console, 'userDB connection error:'));
userDB.once('open', () => {
  console.log('userDB connected');
});
```

2. **Sémák Definiálása:**

- A `mongoose.Schema` segítségével definiáltam az adatbázis sémákat (struktúrákat), amelyek meghatározzák az adatok struktúráját és validációját.

```
const EquipmentSchema = new mongoose.Schema({
    name:{
        type:String,
        required:true
    },
    equipment:{
        type:String,
        required:true
    },
    quantity:{
      type:Number,
      required:true
    }
  });


const UserSchema = new mongoose.Schema({
  username: {
    type:String,
    required:true},
  password: {
    type:String,
    required:true}
});
```

3. **Modellek létrehozása:**
    - A `mongoose.model` segítségével definiáltam az adatbázis modelleket (kollekciókat), amelyek az adatokhoz tartozó műveleteket biztosítják.

```
const EquipmentCollection=equipments.model('EquipmentCollection',EquipmentSchema)
const UserCollection=userDB.model('UserCollection',UserSchema)



module.exports={ EquipmentCollection, UserCollection }
```

- A kód létrehozza a `mongoose` kapcsolatokat két különböző MongoDB adatbázissal.
- Minden adatbáziskapcsolat esetén kezeljük az esetleges hibákat (`equipments.on('error', ...)`), és a kapcsolat megnyitásakor végrehajtott műveleteket (`equipments.once('open', ...)`).
- Az adatbázis sémák (`EquipmentSchema`, `UserSchema`) definiálják az adatok struktúráját és validációját.
- A `mongoose.model` segítségével definiáljuk az adatbázis modelleket a sémák alapján.
- A `module.exports` segítségével exportáljuk a létrehozott modelleket, hogy más fájlok is használhassák őket (pl. a szerver kéréskezelő függvényeiben).