**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.
* A képen képernyőkép, szöveg, Betűtípus, sor látható

  Automatikusan generált leírás
* 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!');**).
* A képen szöveg, képernyőkép, szoftver látható

  Automatikusan generált leírás

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');**).

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

* 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.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

* 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')**).
* A képen szöveg, képernyőkép, Betűtípus látható

  Automatikusan generált leírás
* 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)**).
* A képen szöveg, képernyőkép, képernyő, szoftver látható

  Automatikusan generált leírás
* 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()**).
* A képen szöveg, képernyőkép, Betűtípus látható

  Automatikusan generált leírás
* 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.

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás

**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.
* A képen szöveg, képernyőkép, szoftver látható

  Automatikusan generált leírás

**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.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

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.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**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.

A képen szöveg, képernyőkép, szoftver látható

Automatikusan generált leírás

**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.

A képen szöveg, képernyőkép, szoftver látható

Automatikusan generált leírás

**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.
* A képen szöveg, képernyőkép, szoftver látható

  Automatikusan generált leírás

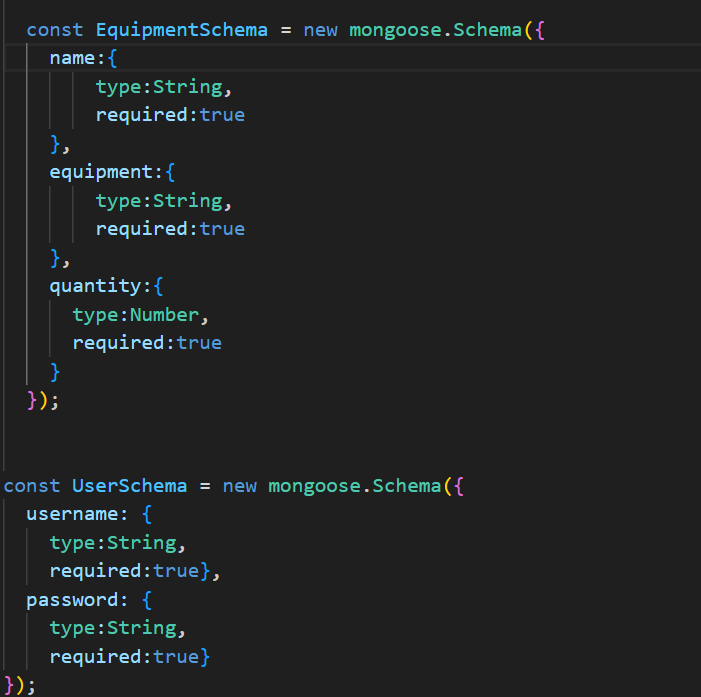
db.js

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.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

1. **Sémák Definiálása:**
   * A **mongoose.Schema** segítségével definiáltam az adatbázis sémákat (struktúrákat), amelyek meghatározzák az adatok struktúráját és validációját.



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

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

* 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).