# REACT

Tartalom

[REACT 1](#_Toc121076279)

[Admin és Publikus felület kialakítása 3](#_Toc121076280)

[Adatok betöltése API végpontról 3](#_Toc121076281)

[Mik az a Hook? 3](#_Toc121076282)

[useEffect hook 4](#_Toc121076283)

[Fecth és a useEffect 4](#_Toc121076284)

[Admin felület kialakítása 7](#_Toc121076285)

[Rootolás 8](#_Toc121076286)

[Layout.js 8](#_Toc121076287)

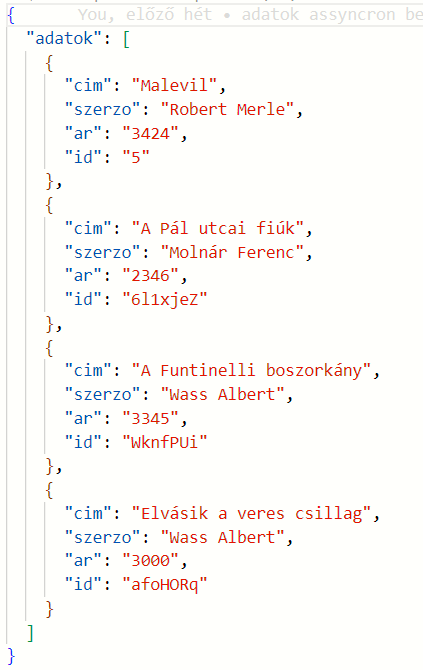
[BrowserRouter 9](#_Toc121076288)

[Admin felület 10](#_Toc121076289)

[Töröl gomb működése 11](#_Toc121076290)

# Admin és Publikus felület kialakítása

## Adatok betöltése API végpontról

Az adataink jelen pillanatban egy constans listából érkeznek. Most szeretnénk, ha egy adatbázisból, API végpontról kapnánk meg őket.

#### JSON szerver beüzemelése

Ehhez először egy virtuális szervert fogunk használni, amely egy json fájl alapján fogja nekünk szimulálni egy adatbázisszerver működését.

1. Ehhez a meglévő listánkat mentsük ki egy külön **adatok.json** fájlba az src mappába.
2. A JSON szerver indításához nyissunk egy **új terminált** és ..
3. Telepítsük a json servert: **npm install -g json-server**
4. Ellenőrizzük, hogy a projektünk mappájában állunk-e?
5. Indítsuk a szerverünket a json fájl helyének megadásával: json-server --watch adat.json   
   Amennyiben a 3000-es portunkat foglalja a react, úgy adjunk meg új portot:  
   **json-server --watch --port=3001 src/adatok.json**
6. Ellenőrizzük le a **localhost:3001** elérési úttal a böngészőben, hogy megkapjuk-e az adatainkat!

## Mik az a Hook?

Eddig a useState hook-ot használtuk egy komponens állapotának frissítéséhez, bár nem tudtuk, hogy ez egy hook.

A hook-ok segítségével a függvény komponensekben elérhetjük a react state-jét és egyéb életciklus metódusait.

1. Hook-ot csak függvény komponensben használhatunk, class komponensben nem. ű
2. Hook-ot csak a legfelső szülőkomponensben használhatunk.
3. Hook-ot ne tehetünk bele egy feltételbe.

## useEffect hook

A UseEffect hook segítségével asszinkron módon tudunk adatokat kezelni, ileltve frissíteni tudjuk a DOM-ot, időzítőket használhatunk. Két argumnetuma van.

useEffect(<function>, <dependency>)

useEffect(() => {

//Runs only on the first render

}, []);

Az első argumentum egy függvény.Amikor a függvény által meghatározott tartalom megváltozik, automatikusan újrarenderelődik az oldalon a vonatkozó tartalom a DOM-ban.

A második paraméter opcionális, arra használjuk, hogy függőségeket adjunk át a useEffectnek. A tömb eleme lehetnek props, vagy state elemek. A useEffect összehasonlítja a tömbben adott értékek előző és az aktuális állapotát, és csak akkor frissíti az oldalt, ha eltérés mutatkozik a két állapot között. Ezzel elkerülhetjük a végtelen hívásokat és frissítéseket.

## Fecth és a useEffect

Használjuk tehát a useEffect hookot az adataink asszinkron beolvasására!

Importáljuk a useEffect függvényt a react könyvtárból.

import { useEffect } from "react";

Hozzunk létre a Public függvénykomponensen belül egy statet a könyvek számára.

const [konyvek, setKonyvek] = useState([]);

Adjuk meg az előzőekben beüzemelt végpontot. Innen érhetőek el a könyveink.

let vegpont = "http://localhost:3001/adatok";

Használjuk a useEffect hook-ot:

 useEffect(() => {

        fetch(vegpont)

            .then((response) => response.json())

            .then((data) => setKonyvek(data));

    }, []);

Most így néz ki a komponensünk:



Persze használhatjuk a már JavaScriptben megszokott osztályunkat is az adatok beolvasásához. Ehhez a model mappába hozzuk létre egy fájlt (könyváruházModell.js, mely a CRUD műveleteket megvalósító osztályt tartalmazza, a már ismert módon.

class KonyvModel {

    #konyvekTomb = [];

    #responseClone;

    getKonyvek() {

        return this.#konyvekTomb;

    }

    adatBe(vegpont, callback) {

        fetch(vegpont, {

            method: "GET",

            headers: {

                "Content-Type": "application/json",

            },

        })

            .then((response) => {

                this.#responseClone = response.clone(); // 2

                return response.json();

            })

            .then((data) => {

                this.#konyvekTomb = data;

                callback(data);

            })

            .catch((rejectionReason) => {

                // 3

                console.log(

                    "Error parsing JSON from response:",

                    rejectionReason,

                    this.#responseClone

                ); // 4

                this.#responseClone

                    .text() // 5

                    .then(function (bodyText) {

                        console.log(

                            "Received the following instead of valid JSON:",

                            bodyText

                        ); // 6

                    });

            });

    }

}

Importáljuk, példányosítsuk, és a useEffect-ben hívjuk meg a megfelelő metódust. A callback függvény itt a könyvek state-t beállító függvény lesz.

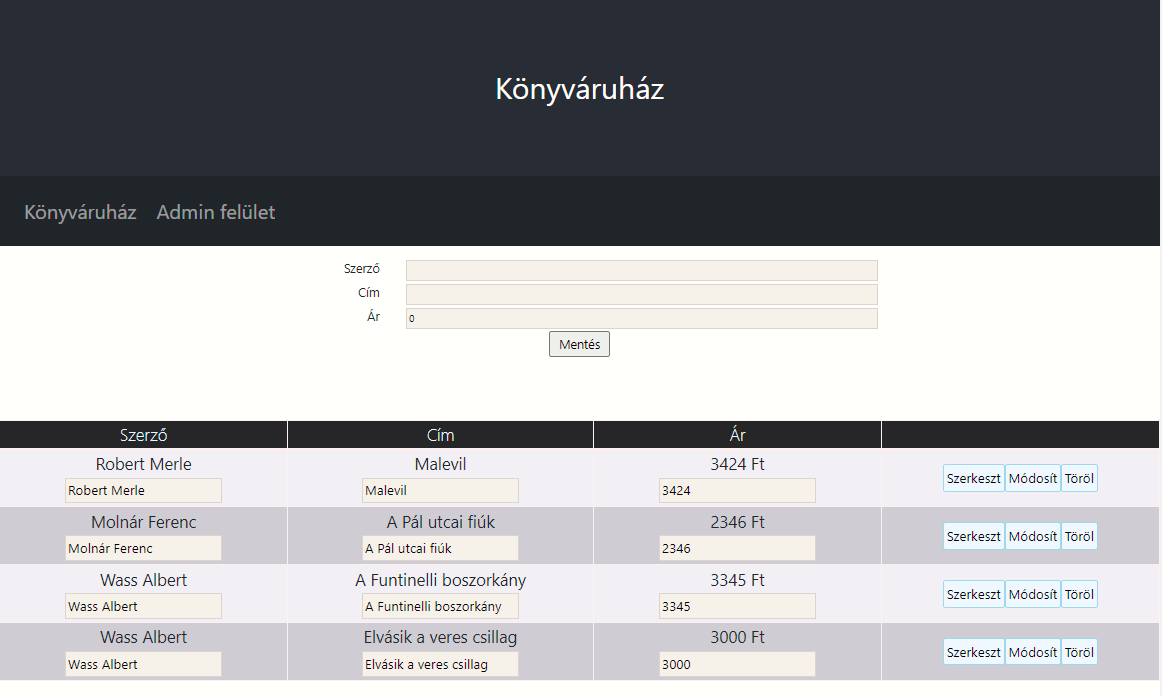


## Admin felület kialakítása

Fejlesszük tovább a könyvesboltunkat!

Az eddigi könyváruházunk a felhasználói oldalról mutatta meg a könyveket, ez volt a publikus felületünk. Most elkészítjük az admin felületet.

Az admin felületen lehetőség lesz az adatok törlésére, módosítására, illetve új adat felvitelére is.



## Rootolás

Első körben szervezzük át kicsit a meglévő fájljainkat, a mellékelt képen látható mintának megfelelően.

Hozzunk létre egy componens mappát, itt pedig egy public és egy admin mappát.

A public mappába másoljuk be az eddigi fájlokat az App.js és az index.js kivételével.

Hozzunk létre egy pges mappát, ide másoljuk be a a publikus felület induló komponensét: Public.js

Hozzunk létre egy Admin.js-t és benne egy egyszerű komponenst, a tartalom csak annyi,, hogy „Admin felület”. Ezt is másoljuk a pages mappába.

Kialakítunk egy menüt, ahol a linkekre kattintva az Admin , vagy a Publikus felületre érkezünk.

A Routoláshoz telepítenünk kell a React Routert. <https://www.w3schools.com/react/react_router.asp>

npm i -D react-router-dom

A React Router használatához szükség lesz egy Layout.js fájlra.

## Layout.js

Ez a fájl tartalmazz a navigációs linkeket. Ehhez importálni kell az Outlet és a Link react router dom függvényeket.

A Layout-ban alakítjuk ku a menüpontokat, az a tag helyett a Link komponenst használva. az Outlet komponensbe töltődik be a Link komponensben megadott tartalom.

import { Outlet, Link } from "react-router-dom";

const Layout = () => {

    return (

        <>

            <nav>

                <ul>

                    <li>

            {/\* A publikus tartalom linkje \*/}

                        <Link to="/">Könyváruház</Link>

                    </li>

                    <li>

            {/\* Az admin oldal linkjetartalom linkje \*/}

                        <Link to="/admin">Admin felület</Link>

                    </li>

                </ul>

            </nav>

            <article>

            {/\* Ide kerül majd az útvonalak/linkek által meghatározott tartalom \*/}

                <Outlet />

            </article>

        </>

    );

};

export default Layout;

A **NoPage.js** is tartalmaz egy egyszerű komponenst, amely a „Nem található az oldal” üzenetet jeleníti meg.

## BrowserRouter

Az APP.js-ben elhelyezzük a BrowserRouter komponensünket. Ehhez az App.js-ben importálni kell az oldalainkat:

import Public from "./components/public/Public";

import Admin from "./components/admin/Admin";

import Layout from "./pages/Layout";

import NoPage from "./pages/NoPage";

import { BrowserRouter, Routes, Route } from "react-router-dom";

és a függvénykomponens return metódusában meg kell adnunk az alábbi sorokat:

  <BrowserRouter>

                <Routes>

                    <Route path="/" element={<Layout />}>

                        <Route index element={<Public />} />

                        <Route path="admin" element={<Admin />} />

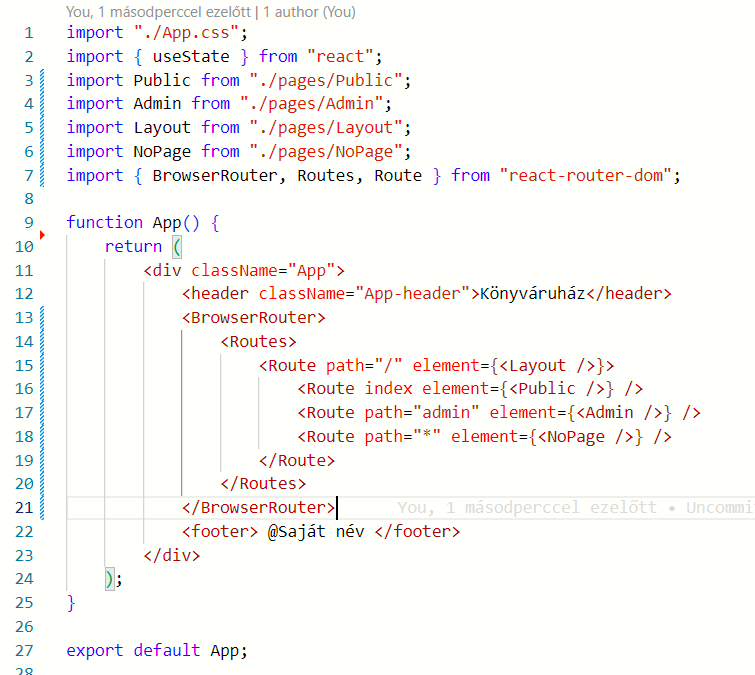
                        <Route path="\*" element={<NoPage />} />

                    </Route>

                </Routes>

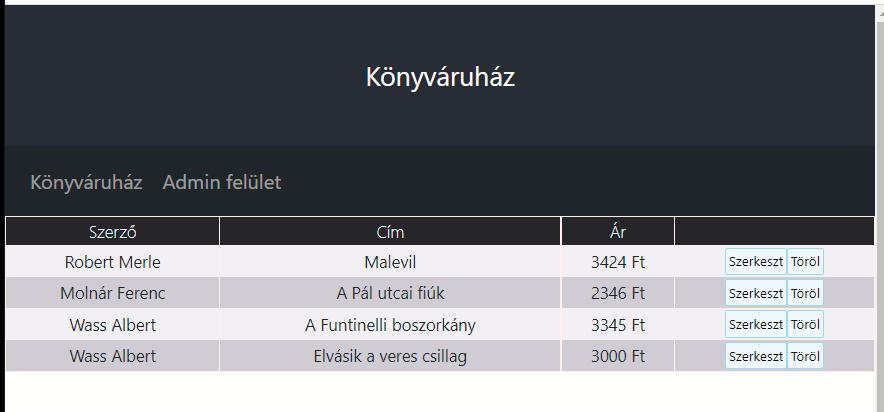
            </BrowserRouter>

Most így néz ki az App komponensünk:



## Admin felület

Az Admin.js a Public.js-hez hasonlóan fog felépülni, csak itt most az adatokat nem div-ekben, hanem egy táblázatban jelenítjük meg. Ehhez készítsük el a szükséges komponenseket!



## Töröl gomb működése

#### A konyvaruhazModel.js-ben a KonyvModel osztály töröl tagfüggvénye:

   adatTorol(vegpont, id) {

        console.log("TÖRÖLTEM: " + id);

        vegpont += "/" + id;

        console.log(vegpont);

        fetch(vegpont, {

            method: "DELETE",

        })

            .then()

            .then(() => {

                console.log("sikeres törlés");

            })

            .catch((error) => {

                console.error("Error:", error);

            });

    }

#### Admin.js-ben

 function torol(id) {

        konyvModel.adatTorol(vegpont, id);

        konyvModel.adatBe(vegpont, setKonyvek);

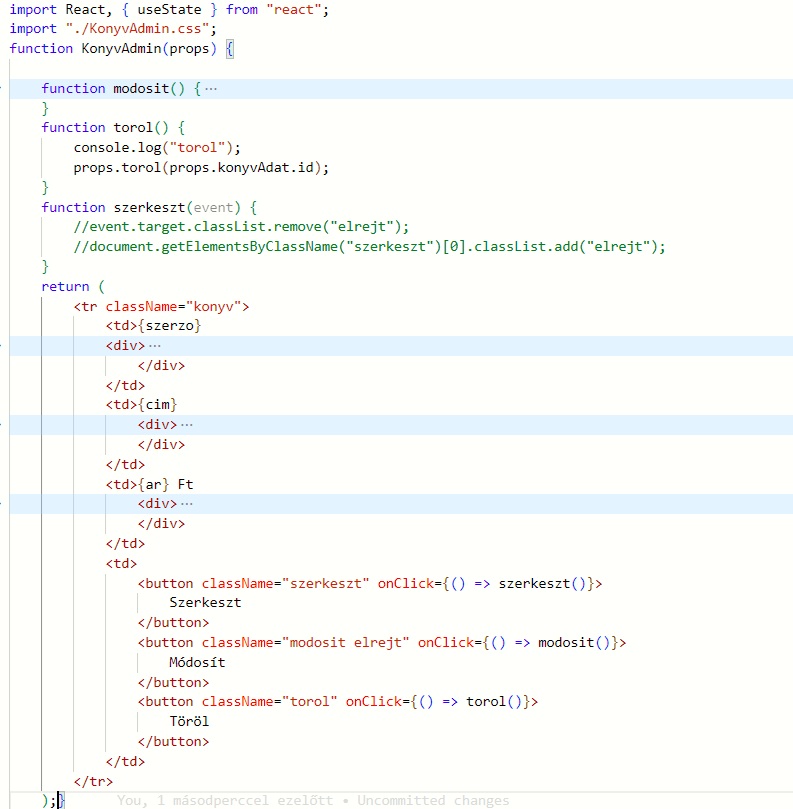
    }

A KonyvAdmin hívásakor át kell adni a props-on keresztül a torol metódust is. Valahogy így:



#### KonyvAdmin.js-ben

A táblázat egyetlen sorát reprezentálja, azaz egyetlen könyvet.



A következő fejezetben megismerkedünk az űrlapok kezelésével!