Képgaléria létrehozása React-tal

# Projekt létrehozása

npx create-react-app appneve

cd appneve

dockerben: npm run start&   //nem dockerben: npm start

A böngészőben: localhost:3000

A képeket a public mappába mentsük.

Az App komponens indítja a programot, mindent ebbe írunk JSX-szel

JSX = html+javascript

Minden komponenshez saját css-t használhatunk.

A css-t is importáljuk!

Alakítsuk ki a képgaléria alapot!

A képen szöveg, képernyőkép, szám, dokumentum látható

Automatikusan generált leírás

Tegyük hozzá a css-t, de használhatunk bootstrapet is!

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

# Ismétlődő elemek és a komponensek

Hozzunk létre egy új komponenest. A komponens feladata lesz egyetlen kép megjelenítése.

Ehhez hozzunk létre az alábbi mapparendszert. A z új komponense a Kiskep.js fájlba kerül.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

Az App.js-be importáljuk be az új komponensünket!

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

 Az App.js-be importáljuk be az új komponensünket!

És cseréljük le az eddigi kiskép diveket a komponensre!

Érdemes a kisképre vonatkozó css-t külön css fájlba helyezni és a kisképhez importálni.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

Persze az adataink dinamikusan érkeznek majd egy JSON adathalmazból. Ezt most egy adatlista létrehozásával szimuláljuk.

Hozz létre egy adatok.js fájlt egy objektumokat tartalmazó listával.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

importáld be az App.js-be!

Annyi kiskép kell, ahány eleme van a listánknak, ezért a galeria-bn ezen a listán kell végigiterálnunk!

Ehhez a map metódust fogjuk használni, nem a forEach-et, mert a map metódusnak lehet visszatérési értéke, márpedig a komponenseknek van visszatérési értéke.

Minden JS kódot {} közé kell írni a komponensben!

A képen szöveg, képernyőkép, sor, Betűtípus látható

Automatikusan generált leírás

Így persze még mindenhol ugyanaz a kép jelenik meg, de már át tudtuk adni a lista értékét az adat props-ban. adat={elem}

Már csak az kell, hogy a Kiskep komponens az src-ben az elem props-ból kiolvassa az elérési utat.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

Most így néz ki a galériánk.

A képen szöveg, képernyőkép, kültéri, növény látható

Automatikusan generált leírás

# Tehát a beágyazott komponensek már tudnak fogadni adatot a szülőkomponenstől. De miképp küldenek vissza adatokat?

Most már tudjuk, hogy a szülőelem a props-on keresztül tud paramétereket átadni a gyereknek. De hogy tudja a gyerek tudatni a szülővel a saját állapotát?

Ugyancsak a props-on keresztül, mégpedig oly módon, hogy a szülőelem a props-ban átadja a saját függvényét, melyet a gyerekelemben meghívhatunk.

1. lépés: A szülőkomponensben hozzunk létre egy metódust, ami fogadni fogja a gyerektől érkező adatot:
2. lépés: A szülőkomponensben a gyerekkompnens prosaként átadjuk a fogadó metódus nevét.

A képen szöveg, képernyőkép, Betűtípus, Párhuzamos látható

Automatikusan generált leírás

1. lépés: A gyerek komponensben hozzunk létre egy kattintás eseménykezelőt, és a függvényt, ami lekezeli az eseményt.
2. lépés: a kattintás eseménykezelőhöz rendelt függvényben hívjuk meg a szülő komponens metódusát.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

# Hogy fog megváltozni a főkép?

A főképnek a kiválasztott kisképtől függően kell változnia

Minden komponensnek lehet egy saját állapota, melyet a state reprezentál. A state állapotértékeinek megváltoztatásával automatikusan újragenerálódik a komponens.

Függvénykomponensek esetén ehhez az úgynevezett useState hook-ot használjuk.

Importálni kell a a React függvénykönyvtárából a useState metódust!

import React, { useState } from "react";

A statek meagadásának lépései:

const [stateValtozo, setStateValtozotkezeloFuggveny] = useState("kezdoertek");

A fenti sor jelentése a következő:

A **stateValtozó** tetszőleges változónév lehet. Ha erre a változóra hivatkozunk valahol a programban, akkor az az érték automatikusan frissül az oldalon, ha a program futása során változik az értéke.

A **setStateValtozotkezeloFuggveny** az a függvény, aminek a feladata, hogy megváltoztassa a hozzárendelt state változó értékét. Csak ekkor fog frissülni az oldalon az adat, ha a változó értékét ezen keresztül változtatjuk.

tehát az App.js-ben a következő változtatásokat kell eszközölni:

1. lépés: importáljuk a useState függvényt
2. Létrehozunk egy state változót és hozzárendeljük a metódsut, amivel megváltoztathatjuk az értékét. Ezt a metódust nem kell megírnunk, csak használni kell!
3. A szulometódsub-ban meghívjuk a statet állító függvényt.
4. a nagykép propsaként a state-t adjuk át!

A képen szöveg, képernyőkép, Betűtípus, Párhuzamos látható

Automatikusan generált leírás

# Feladat: Készítsünk léptetőgombokat!

1. feladat: szervezzük át a kódot! A nagykép és a léptetőgombok és a megjelenő adatok kerüljenek külön komponensbe!