React 4

Segédlet

Készítette: Rajacsics Tamás (rajacsics.tamas@aut.bme.hu)

1 Bevezetés

A gyakorlat folytatása a React 3 gyakorlatnak.

Ha az nincs meg, akkor a hivatalos kiinduló projekttel kell kezdeni. Ebben az esetben a zip kitömörítése után ki kell adni az **npm update** parancsot, hogy leszedje a node_modules mappa tartalmát. Figyelmeztetés: a kiinduló projekt nem tartalmazza az önálló feladatok megoldását.

1.1 Beadás

A gyakorlat végén az elkészített kódot a Moodle-be fel kell tölteni. A beadandó a teljes könyvtár a node_modules mappa kivételével egy zip fájlban.

1.2 Progressive Web App (PWA)

Az alkalmazásunkat telepíteni fogjuk a mobilunkra/számítógépünkre, ami a jelenlegi támogatottság mellett szinte teljes értékű app lehet.

Az előfeltételeket nagyjából teljesítettük: az alkalmazás jól működik kicsi ablakban is.

A telepíthetőség minimum feltételei (PWA):

- Manifest.json
- Service Worker
- https host

2 Feladatok

2.1 manifest.json

A manifest.json fájl a telepített PWA konfigurációs fájlja, bár bizonyos beállítások böngészőben is érvényesülnek. A teljes dokumentáció például itt található:

https://developer.mozilla.org/en-US/docs/Web/Manifest

Hozzuk létre a /public/manifest.json fájlt a következő tartalommal.

```
{
    "name": "Kliensoldali Rendszerek Chat App",
    "short_name": "Chat"
}
```

Majd vegyük fel az index.html-be.

```
<link rel="manifest" href="manifest.json" />
```

Nézzük meg, hogy sikerült-e betölteni és mi a státuszunk PWA tekintetben.

Nyissuk meg a DevTools (F12) / Application / Manifest oldalt, és vizsgáljuk meg, hogy mit sikerült és mit nem sikerült megoldjunk.

Bizonyos hiányosságok javítása egyszerű lesz, másokhoz picit többet kell dolgozzunk.

2.1.1 START URL

Adjuk meg a kötelező start_url-t. Ezt fogja indítani a rendszer, ha fel van telepítve az alkalmazás.

```
"start_url": "./?mode=pwa"
```

Bármit beírhatunk ide. A mode és pwa szavak semmit sem jelentenek a böngészőnek, csak számunkra adnak információt.

2.1.2 DISPLAY

Adjuk meg a másik kötelező elemet a display-t. Ezt állítsuk standalone-ra, hogy teljes app élményt adjunk a felhasználónak.

```
"display": "standalone"
```

Nézzük meg, hogy eltűntek-e ezek a hibák a DevTools-ban!

2.1.3 IKONOK

Kövessük a minimum követelmények teljesítését a DevTools-ban az ikonnal.

Hozzunk létre egy ikont PNG, vagy WebP formátumban, ami teljesíti a minimum követelményt. Legyen 192 pixel méretű. Mentsük el logo192.png (vagy .webp) néven, és tegyük be a /public/ mappába.

Egy lehetőség, hogy screenshotoljuk az alkalmazásunkat, majd kivágjuk belőle a logót, és átméretezzük 192x192 pixelre. Például ez jó is lehet, de egy bármilyen kép megteszi, amit találunk a neten.

Hivatkozzunk rá a manifest.json-ben.

```
"icons": [
  {
    "src": "logo192.png",
    "sizes": "192x192",
    "type": "image/png"
  }
]
```

És bár nem kell azonos ikonnak lennie böngészőben és telepítve, érdemes ugyanazt használni, hogy ne zavarjuk össze a felhasználót. Az index.html-ben cseréljük le az ikont:

```
<link rel="icon" type="image/png" href="logo192.png" />
```

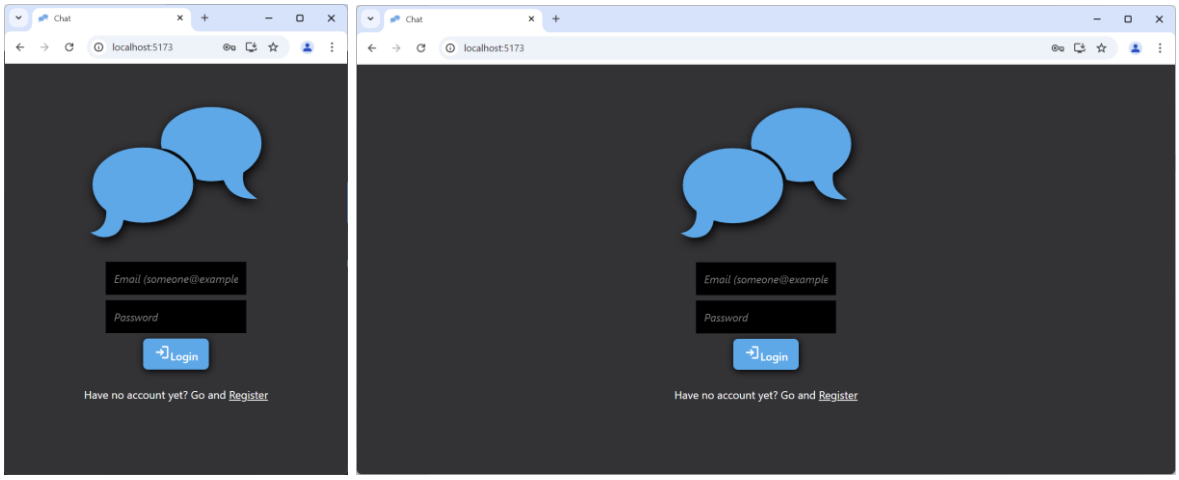
Nézzük meg a DevTools-ban, hogy sikerült-e haladnunk!

2.1.4 SCREENSHOTS

Szükség van a telepíthetőséghez screenshotokra. Desktop telepíthetőséghez kell széles, míg mobil telepíthetőséghez mobil formátumúra van szükség.

Csináljunk két screenshotot az alkalmazásról: egy széleset, és egy keskenyt. Mentsük el őket screenshot-desktop.png és screenshot-mobile.png néven a /public/ mappába.

Egy-egy példa screenshot:



Hivatkozzunk rájuk a manifest.json fájlban. Figyeljünk arra, hogy a méretét pontosan adjuk meg.

```
"screenshots": [
  {
    "src": "screenshot-desktop.png",
    "sizes": "1804x1032",
    "type": "image/png",
    "form_factor": "wide",
    "label": "Chat App Login Screen"
  },
  {
    "src": "screenshot-mobile.png",
    "sizes": "754x1032",
    "type": "image/png",
    "form_factor": "narrow",
    "label": "Chat App Login Screen"
  }
]
```

Vizsgáljuk meg, hogy maradt-e még hibánk a DevTools szerint.

2.2 SERVICE WORKER

A service worker feladata a cache és offline működés kezelése. A mi alkalmazásunk első pillantásra haszontalan offline módban, mert nem tudunk chatelni, azonban van értéke annak, ha mégis működik alapszinten, és meg tudjuk nézni a korábban kapott/küldött üzeneteinket. Tehát nem felesleges az offline mód támogatása.

2.2.1 CACHE-FIRST IMPLEMENTÁCIÓ

Az előadáson elhangzott cache stratégiák közül a cache-first megoldást választjuk, mert egyszerű, és majdnem tökéletes számunkra (általában is és itt is a legjobb megoldás a párhuzamos lekérés app értesítéssel, de az meghaladja a gyakorlat kereteit).

Hozzunk létre egy fájlt Service Workernek /public/sw.js néven a következő tartalommal.

```
const cacheName = "v1";
async function impl(e) {
   let cache = await caches.open(cacheName); // Cache megnyitása, async
   let cacheResponse = await cache.match(e.request); // Lookup
   if (cacheResponse) // Ha megvan
      return cacheResponse // Visszadjuk
   else {
      let networkResponse = await fetch(e.request); // Ha nincs meg, akkor elindítjuk a tényleges
hálózati lekérdezést
      cache.put(e.request, networkResponse.clone()) // Eltároljuk
      return networkResponse; // Visszadjuk
   }
}
self.addEventListener("fetch", e => e.respondWith(impl(e))); // Eseményre feliratkozás
```

Minden fetch jellegű kérésre (de nem websocket kérésre) megnézzük, hogy megvan-e, és ha igen, akkor visszaadjuk. Ha nincs meg, akkor lekérjük, eltároljuk és visszaadjuk.

2.2.2 TypeScript fordító konfigurálása

Ha a TypeScript fordítóban nincs letiltva a js fájlok elemzése, akkor hibákat jelezhet. Oldjuk meg, hogy csak a /src/ könyvtárban dolgozzon.

Módosítsuk a /tsconfig.json-t.

```
"include": ["src"]
```

(Lehetséges, hogy újra kell indítani az app-ot, a debuggert és a webszervert is, hogy ez a változás érvényesüljön. Ha igen, akkor le kell állítani a web szervert ctrl+c-vel, majd újraindítani npm start-tal)

2.2.3 Server worker telepítése

A server worker telepítése kódból lehetséges. Hozzuk létre a /src/Pwa.ts fájlt, amiben a PWA-val kapcsolatos kódot tesszük az alábbi tartalommal.

```
class Pwa
{
   #serviceWorkerRegistration?: ServiceWorkerRegistration;
   constructor()
   {
      if ( isSecureContext )
      {
         ( async () =>
         {
            this.#serviceWorkerRegistration = await navigator.serviceWorker.register( "sw.js" );
         } )();
      }
   }
}
export const pwa = new Pwa();
```

A kód lényege, hogy regisztrálja a service workert, de csak akkor, ha az lehetséges (isSecureContext).

Importáljuk be ezt a fájlt az index.tsx-ben, hogy lefusson a kódja.

```
import "./Pwa"
```

Vizsgáljuk meg, hogy sikerült-e regisztráljuk a service workert: DevTools (F12) / Application / Service workers.

Ha sikerült, akkor a cache-nek is működnie kell.

Vizsgáljuk meg, hogy a cache-ban vannak-e elmentett elemek: DevTools (F12) / Application / Cache storage / v1 - http://localhost:5173/

A cache-t és a server worker-t tudjuk törölni, ha szükségünk lesz egy üres indításra.

Figyeljük meg, hogy megjelent a böngészőben az install gomb a címsorban a jobb oldalon.

Ezzel teljesítettük az összes követelményt, hogy telepíthető legyen az alkalmazásunk.

2.3 BUILDELÉS

2.3.1 FELKÉSZÜLÉS ALOLDALRA

Ha az alkalmazásunk nem a domain root-on fut, hanem a path nem üres, akkor arra külön készülnünk kell. Jelenleg minden link abszolút path-t használ.

A vite.config.ts fájlban adjuk hozzá a következőt a defineConfig-ban.

```
base: "",
```

Ez relatívra állítja az összes linket.

2.3.2 BUILD

Az alkalmazás jelenleg csak fejlesztői környezetben működik Vite segítségével. Ennek az oka, hogy a build nélküli eszközök nem hoznak létre egy csomagot az alkalmazásból, hogy kényelmesen és gyorsan lehessen fejleszteni, kicserélni kódot futás közben, stb.

Hozzuk létre a csomagot. Ehhez vagy nyissunk egy új terminál ablakot, vagy csak állítsuk le az npm start futását.

```
npm run build
```

Ez létrehozza a csomagot a /dist/ mappában.

Nézzük meg, hogy mi jött létre a /dist/ mappában.
Itt meg is nézhetjük, hogy sikerült-e helyes, relatív linkeket generálni. A generált /dist/index.html-ben relatív linken kell legyen a stylesheet és a module.

2.4 GITHUB PAGES – OPCIÓ 1

2.4.1 HELYI REPOSITORY

Tegyük fel a forrást GitHub-ra.

VSCode-ban menjünk a source control tabra (3. bal oldalon) és nyomjuk meg az initialize repository gombot.

Ez létrehozza a .git mappát, ami a git konfigurációt és minden adatot tartalmaz.

Konfiguráljuk be a felhasználónév és név párost. Írjuk be a terminálba a következőket.

```
git config user.email "valamilyen@email"
git config user.name "Egy név"
```

Ezzel készen vagyunk a konfigurációval.

Végül commit-oljuk a kódot: adjunk egy nevet (pl. first commit) és nyomjunk rá a commit gombra. (ez csak egy helyi csomagot hoz létre, ténylegesen nem tesz fel semmit)

Ha sikeresek voltunk, akkor a gomb Publish Branch-re változik.

2.4.2 GITHUB

Szükségünk van egy https kiszolgálóra, ahova feltehetjük az alkalmazásunkat. Legyen ez most a GitHub.

Regisztráljunk be GitHub-ra, ha még nem vagyunk. https://github.com/

GitHub csak akkor működik statikus oldalak és alkalmazások hostolására, ha a forrást feltesszük.

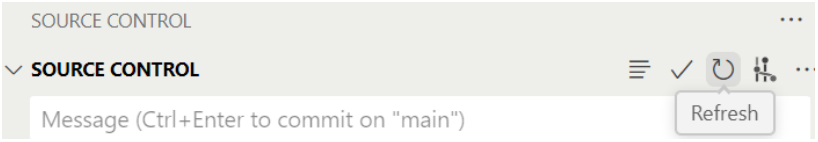
Hozzunk létre egy repository-t krchat néven. Minden mást hagyjunk alapértelmezetten (public kell legyen). Ezt gyakorlat után törölhetjük majd.

A létrehozás után a Code fülön (ahova alapban kerülünk) látjuk a szükséges kódot, ami összeköti a helyi repository-t a szerveren lévővel.

Adjuk ki a következő parancsot a terminálban, hogy a git tudja, hova kell feltölteni a kódot. Ne felejtsük el lecserélni az ACCOUNT-ot arra, amit GitHub kiírt nekünk.

```
git remote add origin https://github.com/ACCOUNT/kchat.git
```

Frissítsük VSCode source control részét.



Majd nyomjuk meg a Publish gombot.

A böngészőben frissítés után látnunk kell a kódunkat GitHub-on.

2.4.3 DIST PUBLIKÁLÁSA

Jelenleg a buid kimenet csak helyileg van meg, de nekünk ezt is fel kell tenni, hogy tudjuk hostolni az alkalmazást GitHub-ról.

Töröljük a dist sort a .gitignore fájlból.

A source control tab frissítés után mutatja, hogy megint vannak fájlok, amit fel lehet tenni.

Commitoljuk és push-oljuk ezeket a fájlokat is: Message-be írjunk be valamit, majd Commit & Push.

Ellenőrizzük, hogy ott van-e minden GitHub-on.

2.4.4 GITHUB PAGES

GitHub képes hostolni statikus oldalakat.

A projekt oldalán menjünk be a Settings/Pages fülre, majd engedélyezzük a GitHub Pages-t a Branch állításával: main, majd Save.

Ezek után működik az oldalunk a megadott linken, csak hozzá kell tenni a /dist-et a végére.

https://ACCOUNT.github.io/krchat/dist/

2.5 EGYÉB STATIKUS SZOLGÁLTATÓK – OPCIÓ 2

Ha Github nem működik, vagy máshova szeretnénk feltenni az alkalmazásunkat, akkor választhatunk egy másik tetszőleges szolgáltatót (regisztrálni kell mindegyikhez, ami ingyenes).

Példák:

https://tiiny.host/

https://static.app/

2.6 HELYI TELEPÍTÉS

Telepítsük fel az alkalmazást az install gomb megnyomásával a böngésző címsorában jobb oldalon. Zárjuk be a böngészőt, majd indítsuk el a desktopról, ha magától nem indult el.

3 ÖNÁLLÓ FELADATOK

A feladat Push Notification kezelés kliens oldalon.

Mivel a böngésző segítségével tudunk teszt üzenetet küldeni, újra indítsuk el debug módban az alkalmazást, ha nem futna (npm start).

3.1 FELKÉSZÜLÉS PUSH NOTIFICATION ÉRKEZÉSÉRE

Kérjünk engedélyt a böngészőtől push notification megjelenítésre a **Notification.requestPermission** függvénnyel. Ezt elvileg egy gomb megnyomására kéne meghívni, de most csak tegyük be az index.tsx-be valahova globálisan (nem gond, ha minden indításra meghívódik, mert nem fogja újra és újra megkérdezni a felhasználót).

A böngészőben fel fog jönni egy popup, hogy engedélyezzük-e az értesítéseket. Erre nyomjunk Allow-t.

3.2 PUSH NOTIFICATION KEZELÉSE

A Service Workerben (sw.js) kezeljük le a beérkező értesítéseket.

1. Iratkozzunk fel a push eseményre. Hasonló a fetch eseményre feliratkozáshoz.
2. Szerezzük meg az üzenet tartalmát, ami az **e.data?.text()** hívással kérhető le, ahol e az esemény paramétere.
    a. Null-t ad vissza, ha üres.
3. Jelenítsük meg a felhasználónak az üzenetet a **self.registration.showNotification** hívással.
    a. Az első paramétere lesz a címsor, ez bármi lehet (pl. "Chat Notification")
    b. A második paramétere egy objektum, amiben ki lehet tölteni egyéb paramétereket. A body paraméterbe tegyük be, amit kaptunk a 2. pontban.
    c. A hívás visszaad egy Promise-t, amire az **e.waitUntil** hívással várhatunk.
    d. Doksi itt: https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerRegistration/showNotification

3.3 KÜLDJÜNK TESZT PUSH NOTIFICATIONT

A DevTools (F12) / Application / Service workers oldalon a Push gombot nyomjuk meg. Figyeljük meg, hogy meg jön-e a push notification.

Ha nem jön, akkor nézzük meg, hogy

- Elmentettünk-e mindent és újraindítottuk-e az alkalmazást (F5).
- Lekezeltük-e az üzenetet az sw.js-ben

- Van-e jogunk megjeleníteni üzenetet. Ezt látjuk a konzolban, mert kiír egy hibát, ha nincs jogunk.