React 2

Segédlet

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

Utolsó módosítás: 2025-10-16

1 Bevezetés

A gyakorlat folytatása a React 1 gyakorlatnak.

Ha az nincs meg, akkor a hivatalos kiinduló projekttel kell kezdeni. Ebben az esetben a zip kitömörítése után ki kell adni az **npm install** 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.

2 Feladatok

Ezeket a komponenseket fogjuk létrehozni (App és Login már készen vannak).

App
- Login
- Main
  - LeftPane
    - TextInputAndButton
    - ConversationCard
  - RightPane
    - MessageCard
    - TextInputAndButton

Az alkalmazás felületét a Main komponens kezeli, ha be vagyunk lépve, különben a Login komponens látszik.

2.1 Gyerek komponens funkcionalitásának publikálása

Készítsünk egy többször is felhasználható input+button komponenst TextInputAndButton néven. Használjuk fel a már létező TextInput komponenst, és publikáljuk ki a teljes funkcionalitását.

2.1.1 LÉTEZŐ PROPS BŐVÍTÉSE

Hozzunk létre egy fájlt TextInputAndButton.tsx néven. Bővítsük a már létező TextInputProps-ot két mezővel, és hozzuk létre a komponenst.

```
import { TextInput, TextInputProps } from "./TextInput";
export type TextInputAndButtonProps = TextInputProps & {
    buttonContent?: string;
    onClick?: () => void;
}
export function TextInputAndButton( { buttonContent, onClick, ...textInputProps }:
TextInputAndButtonProps )
{
    return <div class="TextInputAndButton">
        <TextInput { ...textInputProps } />
        <button type="button" />
    </div>
}
```

Figyeljük meg a spread operátor használatát mindkét helyen.

- Az első (a függvény paramétereinél) összegyűjti egy objektumba az összes maradék tulajdonságot, amit a függvény kap, tehát a buttonContent és onClick props-on kívül mindet, ami pont a TextInput props-a.
- A második (a TextInput hívásakor) az objektum tulajdonságait, mint attribútumokat adja át.

2.1.2 PROPS MÓDOSÍTÁSA GYEREK FELÉ

Kezeljük le az onEnter-t a TextInput-on, illetve a gombot is implementáljuk megfelelően.

```
<TextInput { ...textInputProps } onEnter={ onClick } />
<button type="button" onClick={ onClick }>
    { buttonContent }
</button>
```

Figyeljük meg, hogy az onEnter nem működik többé ezen a vezérlőn, hiába adnánk meg kívülről, mert felül van írva az onClick függvénnyel. Mostantól csak az onClick működik, ami pontosan az, amire szükségünk van, mert nem akarunk két eseményt (onEnter és onClick) is lekezelni ugyanúgy.

TextInputAndButton.css fájl:

```
.TextInputAndButton {
    grid-template-columns: 1fr auto; /* Gyerekek elrendezése úgy, hogy TextInput kapja a maximális helyet, button pedig a lehető legkisebbet */
    gap: 4px; /* legyen köztük némi távolság*/
    background: black; /* Az egész háttér legyen fekete, mintha az egész a TextInput lenne*/
    align-items: center; /* Függőlegesen középre igazítjuk a gyerekeket*/
    padding-right: 4px; /* A gomb után hagyunk pici helyet, hogy ne érjen ki a szélére*/
}
```

Importáljuk be a TextInputAndButton.css fájlt a tsx fájlban.

2.2 LeftPane

A bal oldali rész a LeftPane. Itt helyezkedik el a meghívó gomb, illetve a már meghívottak listája. Hozzuk létre a LeftPane.tsx és css fájlokat.

```
export function LeftPane()
{
    let [ invite, setInvite ] = useState( "" );
    return <div class="LeftPane">
        <TextInputAndButton value={ invite } onChange={ setInvite } buttonContent="Invite"
            placeholder="Tag" />
        <div />
    </div>
}
```

Tegyük be az importokat, hogy forduljon, illetve a css fájlt is importáljuk.

Egyelőre csak kitesszük a meghívó gombot, amivel új beszélgetést tudunk nyitni valakivel.

A hozzá tartozó CSS a LeftPane.css:

```
.LeftPane {
    grid-template-rows: auto 1fr; /* Függőlegesen két részre osztjuk, a felső csak egy soros, az alsó a maradék helyet veszi fel */
    border-right: 1px solid gray; /* Elválasztó keret a jobb oldalon */
}
```

2.3 RightPane

A felső rész lesz az üzenetek, az alsó az üzenet küldés.

Hozzuk létre a RightPane.tsx és css fájlokat.

```
export function RightPane()
{
    let [ message, setMessage ] = useState( "" );
    return <div class="RightPane">
        <div />
        <TextInputAndButton value={ message } onChange={ setMessage } buttonContent="Send"
            placeholder="Write a message..."/>
    </div>
}
```

Tegyük be az importokat, hogy forduljon, illetve a css fájlt is importáljuk.

A hozzá tartozó CSS a RightPane.css:

```
.RightPane {
    grid-template-rows: 1fr auto; /* Függőlegesen két részre osztjuk, az alsó csak egy soros, a felső a
maradék helyet veszi fel */
}
```

2.4 Main

A Main simán csak a bal és jobb oldal egymás mellett.

Hozzuk létre a Main.tsx és css fájlokat.

```
export function Main()
{
    return <div class="Main">
        <LeftPane />
        <RightPane />
    </div>
}
```

Tegyük be az importokat, hogy forduljon, illetve a css fájlt is importáljuk.

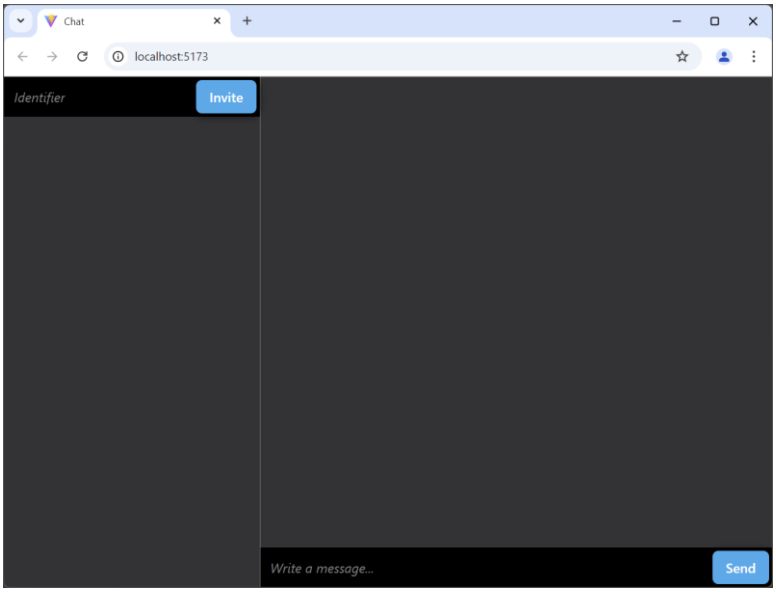
A hozzá tartozó Main.css:

```
.Main {
    grid-template-columns: 1fr 2fr; /* Vízszintesen két részre osztjuk 1/3 - 2/3 arányban */
}
```

Az App-ban a tesztelés kedvéért tegyük ki a Main-t a Login helyett.

```
function App()
{
    return <Main />
}
```

Jelenleg így néz ki az alkalmazás.



2.5 Külső adatbázishoz kapcsolódás

A chat alkalmazás adatbázisa websocketen keresztül érhető el, és ezen keresztük kapjuk az értesítéseket is. A kommunikáció nem kérés-válasz alapú, hanem a szerver folyamatosan frissíti a kliens állapotot, a kliens pedig parancsokat küldhet a szervernek.

Bár nem szükséges típusokat definiálni a működéshez, a könnyebb hibakeresés és fejlesztés kedvéért hozzuk létre a típusokat. Általában ezt a szerver fejlesztőitől kapjuk, kézzel ritkán írjuk. Egy új fájlba, a ChatService.ts-be dolgozzunk.

```
export type MessageDto = {
    id: number;
    timeStamp: string;
    referenceTo: number; // 0: normal message, +: update, -: delete
    senderId: string;
    contentType: number;
    content: string;
}
```

```
export type UserDto = {
    id: string;
    displayName: string;
    tag: string;
    lastSeen: string;
}
```

```
export type ConversationDto = {
    channelId: string;
    parentChannelId: string;
    name: string;
    description: string;
    data: string;
    state: number; // disconnected, outgoingRequest, incomingRequest, accepted, group
    access: number; // none, read, write, admin
    notificationLevel: number; // none, gray, push
    unreadCount: number;
    memberIds: string[];
    lastMessages: MessageDto[];
}
```

```
export type InboxDto = {
    user: UserDto;
    contacts: UserDto[];
    conversations: ConversationDto[];
}
```

```
export type OutgoingPacket =
    { type: "login", email: string, password: string, staySignedIn: boolean } |
    { type: "loginWithToken", token: string } |
    { type: "register", email: string, password: string, displayName: string, staySignedIn: boolean } |
    { type: "contactRequest", email: string, firstMessage: string } |
    { type: "message", channelId: string, referenceTo: number, contentType: number, content: string };

export type IncomingPacket =
    { type: "error", message: string } |
    { type: "login", query: string, token: string, inbox: InboxDto } |
    { type: "message", channelId: string, message: MessageDto } |
    { type: "conversationAdded", conversation: ConversationDto } |
    { type: "conversationRemoved", channelId: string } |
    { type: "user", user: UserDto };
```

Ezek kódot nem generálnak, csak a típusellenőrzéshez és kódkiegészítéshez kellenek.

A szerver irányába kimenő csomag OutgoingPacket formátumban megy, míg a szerverről bejövő IncomingPacket formájú. Mindkettő JSON-ként közlekedik.

2.6 HELYI ADATBÁZIS

Hozzunk létre egy osztályt ChatService néven a ChatService.ts-be, ami a websocket kapcsolatot kezeli. És hozzunk létre egy globális példányt belőle chatService néven.

```
class ChatService
{
    #ws = new WebSocket( "wss://kliensoldali.azurewebsites.net/" );
    constructor()
    {
    }
}
export const chatService = new ChatService();
```

Írjuk meg a küldő függvényt, ami a megfelelő formátumban (JSON) elküldi a csomagot a szervernek.

```
send ( packet: OutgoingPacket )
{
    this.#ws.send( JSON.stringify( packet ) );
}
```

A helyi tár a szerverről bejött adatok tárolására való. Ezt később lementhetjük, hogy offline is lehessen nézni az üzeneteket. Vegyük fel az inbox tulajdonságot az osztályba.

```
inbox?: InboxDto;
```

Majd kezeljük a bejövő csomagokat a konstruktorban.

```
this.#ws.addEventListener( "error", () => alert( "WebSocket hiba, zárd be a felesleges tabokat, és
frissítsd az oldalt." ) );
this.#ws.addEventListener( "message", e =>
{
   let p = JSON.parse( e.data ) as IncomingPacket;
   console.log( p ); // DEBUG
   switch ( p.type )
   {
      case "error":
         alert( p.message );
         break;
      case "login":
         this.inbox = p.inbox;
         break;
      case "message":
         let cid = p.channelId;
         this.inbox!.conversations.find( x => x.channelId === cid )?.lastMessages.push( p.message );
         break;
      case "conversationAdded":
         this.inbox!.conversations.push( p.conversation );
         break;
   }
} );
```

2.7 REACT KOMPONENSEK ÉRTESÍTÉSE

Ha bejön egy üzenet, akkor értesíteni kell a felhasználói felületet, hogy megjelenjen az új adat.

A komponensek értesítéséhez addListener és removeListener párost vezetünk be, amihez el kell tárolni a feliratkozott függvényeket. Vegyünk fel ezeknek egy tömböt.

```
#listeners: ( () => void )[] = [];
```

A végigiterálást tegyük be message kezelésének végére (a switch után), a konstruktorban.

```
for ( let listener of this.#listeners )
   listener();
```

Az add és remove pedig így néz ki.

```
addListener( listener: () => void )
{
    this.#listeners = [ ...this.#listeners, listener ];
}

removeListener( listener: () => void )
{
    this.#listeners = this.#listeners.filter( x => x !== listener );
}
```

Látható, hogy az implementáció mindig új tömböt hoz létre, így az add és remove nem fog ütközni az iterációval. A megoldás nem annyira pazarló, mint amilyennek látszik, mert a fel- és leiratkozás ritka esemény.

2.8 useEffect

Függvénykomponensek értesítéséhez a feliratkozást a useEffect teszi lehetővé. Iratkozzunk fel az App komponensben (index.tsx) a bejövő csomag eseményre, majd az alapján tegyük ki a Login, vagy a Main komponenst, hogy be vagyunk-e lépve.

Első lépés egy állapot felvétele, hogy frissíteni tudjuk a komponenst.

```
let [ loggedIn, setLoggedIn ] = useState( false );
```

A useEffect használata a fel és leiratkozásra.

```
useEffect( () =>
{
   const listener = () => setLoggedIn( !!chatService.inbox );
   chatService.addListener( listener );
   return () => chatService.removeListener( listener );
}, [] );
```

Létrehozzuk a listenert, ami az állapotot állítja, majd feliratkozunk és leiratkozunk.

Tegyük be az importot a useState-nek, a useEffect-nek és a chatService-nek a szokásos módon.

Végül az állapot alapján tesszük ki a felületet.

```
return loggedIn ? <Main /> : <Login />
```

Vizsgáljuk meg a websocket csatornát a Network tabon a DevTools-ban (frissíteni kell az alkalmazást, hogy a DevTools elkapja a kapcsolódást). Egy új websocketnek meg kell jelennie, de üzenetek nem közlekednek egyelőre: 101 Pending státusz, nincs konzol hiba.

2.9 Login komponens logikája

Kezdjük el használni a ChatService osztályt, hogy be tudjunk regisztrálni és lépni a szerverre.

A Login.tsx-ben a useState-ek alá, de még a return elé vegyünk fel egy új függvényt.

```
function loginRegister()
{
   if ( register )
      chatService.send( { type: "register", email, password, displayName, staySignedIn: true } );
   else
      chatService.send( { type: "login", email, password, staySignedIn: true } );
}
```

A függvény a register állapot függvényében meghívja vagy a login-t, vagy a register-t.

Ezt a függvényt adjuk meg button onClick-nek, illetve a TextInput onEnter-nek.

Ezzel készen is vagyunk a Login komponenssel.

Próbáljuk ki, regisztráljunk be valami egyedi kóddal (ne valós adat legyen, mert másokat ez alapján lehet majd meghívni – a szerver nem ellenőrzi az email cím helyességét)! Nézzük meg, hogy jön-e login csomag (a konzolon és a DevTools-ban látni)!

Előfordulhat, hogy a belső állapotok a kód szerkesztése közben beálltak valami véletlenszerűre, így, ha a Login nem vált át a Main-re, akkor frissítsük az alkalmazást előbb (F5), majd próbáljunk újra belépni.

3 Önálló feladatok

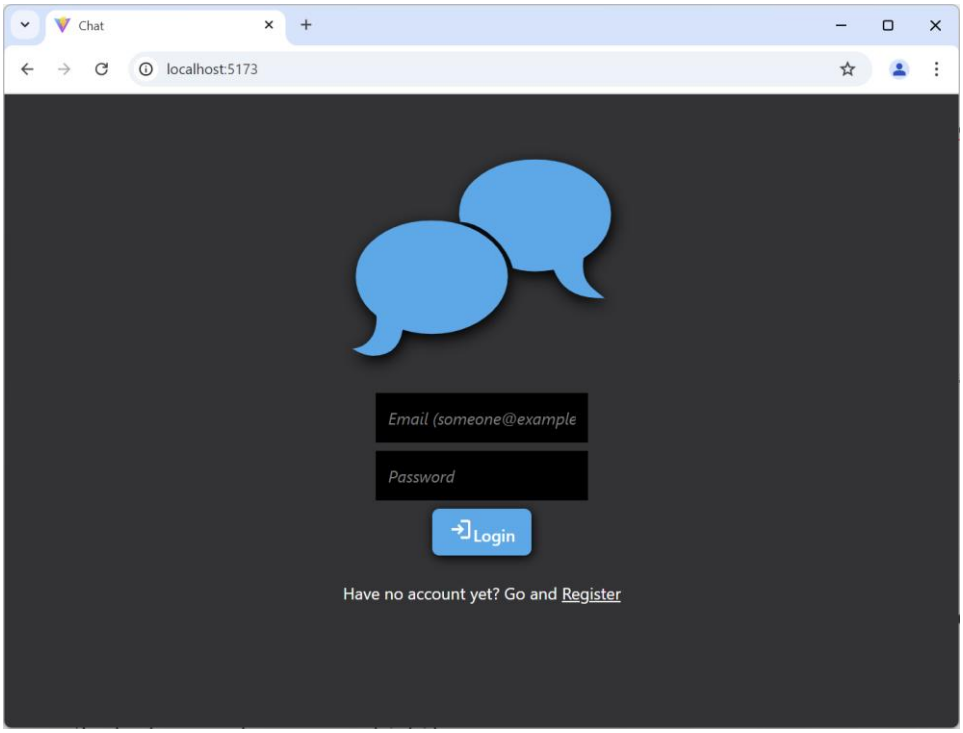
3.1 Ikonok

Vezessünk be mindenhova ikonokat, hogy jobban nézzek ki az alkalmazásunk.

Több lehetőségünk is van (fontawesome, google, …), mi most használjuk a Google Material Icons készletet, mert ezt a legegyszerűbb hozzáadni a projekthez.

1. Menjünk fel a https://fonts.google.com/icons oldalra.
2. Keressünk rá egy ikonra (pl. Login), válasszuk ki, és jobb oldalt megjelennek a telepítési utasítások.
3. Kövessük az utasításokat:
    a. Tegyük be az index.html-be a link-et a **Variable icon font** részből.
    b. Tegyük be a css style-t a HTML-be, vagy az index.css-be a **Variable icon font** részből.
    c. A login gombba a szöveg elé tegyük be a span-t az **Inserting the icon** részből.

Ezek után az alkalmazás így néz ki. (az ikon nem jó helyen van még)



3.2 IconButton komponens

Bár CSS-sel mindenhol meg tudnánk oldali, hogy jól nézzen ki az ikon, egy jobb megoldás, ha készítünk hozzá egy komponenst IconButton néven.

1. Hozzuk létre a fájlokat: IconButton.tsx és IconButton.css
2. Írjuk meg a komponens kódját.
    a. props-ként kapja meg az ikon nevét, a szöveget és az onClick eseményt.
    b. Generáljon egy button-t a megadott paraméterekkel.
3. CSS-ben rendezzük el az ikont és a szöveget szépen (display: flex, gap: 4px, align-items: center)
4. Használjuk fel az IconButton-t a Login-ban.

3.3 Bónusz feladat: TextInputAndButton komponens

Használjuk fel az IconButton-t a TextInputAndButton-ban is.

1. Vegyük át az ikon nevét is a props-ban.
2. Cseréljük le a button-t az IconButton-ra.
3. Adjuk át az ikon nevét
4. Adjunk meg ikonokat a LeftPane-en az invite gombra (például add ikon) és a RightPane-en a küldés gombra (például send ikon).