Context API i Portali

Proširen je primjer klijentske aplikacije sa prethodnog predavanja. Dodane su još dvije komponente ThemeContext i Modal.

1. **ThemeContext.js**

U primjeru je korišten Context API. U prethodnim primjerima je vidljivo da se podaci u komponente prosljeđuju preko propertija (sa roditelja na dijete). Pristup prosljeđivanja podataka na taj način je dozvoljeno i opravdano zaobići **u iznimnim situacijama**. Primjer su podaci koji su uvijek isti i traženi su na nivou cijele ili većine aplikacije (poput UI teme ili recimo odabranog jezika). Za takve potrebe postoji mehanizam koji se zove Context. Kako se koristi pogledati u dokumentaciji. (https://reactjs.org/docs/context.html)

Što je točno createConetxt metoda koja je korištena u primjeru sa predavanja (u datoteci ThemeContext) i kako se koristi pogledati detaljnije u https://reactjs.org/docs/context.html#reactcreatecontext.

Primjer korištenja komponente Context.Provider je pokazan u datoteci App.js, a više o njoj pročitajte na linku - https://reactjs.org/docs/context.html#contextprovider.

Primjer korištenja komponente Context.Consumer je pokazan u datoteci Details.js. Više o tome kako se koristi pročitajte na https://reactjs.org/docs/context.html#contextconsumer. Primjetit ćete da se kao dijete očekuje funkcija koja kao argument ima vrijednost (value) konteksta Provider komponente na koju je preplaćena, a kao return se očekuje komponenta (markup)-odnosno čvor u React DOM-u.

```
<ThemeContext.Consumer>
                {([theme])=>(
            <button style={{backgroundColor:theme}} onClick={this.toggleMo
dal}>
                Movie: {name}
                </button>
                )}
</ThemeContext.Consumer>
```

Primjer korištenja hook-a useContext je dan u datoteci SearchParams.js. Hook useContext radi sličnu stvar kao komponenta Context.Consumer.
useContext – kao argument prima Context objekt. Samim time je napravljena preplata na Context.Provider. Kada se neke komponente želi pristupiti vrijednosti (ili funkciji za mijenjanje) koja je u Context.Provider komponenti, pozove se funkcija useContext (`const [theme, setTheme] = useContext(ThemeContext);`). Više o useContext pročitati na useContext - https://reactjs.org/docs/hooks-reference.html#usecontext .
Kad se želi promijeniti vrijednost (kao npr. promjena boje botuna, u primjeru sa predavanja) tada se

poziva funkcija za promjenu vrijednosti stanja setTheme.

Pripaziti! Kontekst se koristi isključivo kada se nekom podatku želi pristupiti iz više komponenti koje su na različitim nivoima hijerarhije stabla. Neoprezno korištenje Context API-a dovodi do toga da se komponenta ne može iznova iskoristiti (eng. *reuse*), a time se narušava cijela ideja React paketa.

Portals

Portal je način da se u DOM ubaci čvor za kojeg ne želimo da bude renderan unutar roditeljske komponente, a opet želimo da bude pod punom kontrolom React-a, odnosno da ima sve mogućnosti i saznanja kao i standardna djeca roditeljske komponente.
Službena dokumentacija preporučuje da se Portali renderaju izvan app (root) elementa.
Detaljnije o tome što je portal možete pročitati na sljedećem linku https://reactjs.org/docs/portals.html.
U našem primjeru je korišten za prikaz modalnog prozora klikom na botun.

Prilikom kreiranja modala kreira se markap koji se ubacuje u DOM. Nakon zatvaranja modala cilj je taj makrap uništiti i maknuti iz doma. Zbog toga je korišten hook UseRef.

Modal.js

U kodu koji je prikazan niže, koristi se hook useRef - dokumentaciju pogledajte na sljedećem linku https://reactjs.org/docs/hooks-reference.html#useref . Ono što useRef vraća je promjenjivi objekt. Jedan od property-a tog objekta je property *current*, te se on inicijalizira na vrijednost prosljeđenu u useRef hook-u (u našem slučaju null). U biti, useRef se može zamisliti kao kutija koja u svom current property-u može sadržavati vrijednost koja se mijenja. Razlika u kreiranju vlastitog objekta {current:...} i putem useRef hook-a je u tome što useRef prilikom renderanja svaki put referencira na isti objekt (ne kreira se novi markup). UseRef nas neće obavijestiti kad se njegov sadržaj promijeni, što znači da promjena sadržaja u current property-u neće uzrokovati re-renderanje DOM-a.
{

```
  const elRef = useRef(null);
  if (!elRef.current) {
    const div = document.createElement("div");
    elRef.current = div;
  }
```

U UseEffect() funkciji:
Hvatanje Modal-a u DOM-u (index.html):

```
const modalRoot = document.getElementById("modal");
```

I onda se na njega doda div

```
modalRoot.appendChild(elRef.current)
```

Na kraju se uklanja modal:

```
return () => modalMovieRoot.removeChild(elRef.current);
```

Povrat iz komponente:

```
return createPortal(<div>{children}</div>, elRef.current);
```

CreatePortal – children: prvi argument predstavlja komponente koje ce se renderati u container: drugi argument.

Details.js
Dodane su još dvije funkcije:

`changeModalState = () => this.setState({ showModal: !this.state.showModal });` - služi za izmjenu stanja modal-a i poziva se klikom na botun "Klik on movie"

```
movieImdb = () => (window.location.href = "https://www.imdb.com/search/");
```

```
{showModal ? (
          <Modal>
            <div>
              <h1>Would you like to see movie {title} on IMDB?</h1>
              <div className="buttons">
                <button onClick={this.movieImdb}>Yes</button>
                <button onClick={this.changeModalState}>No</button>
              </div>
            </div>
          </Modal>
        ) : null}
```

Učitavanje modala u kojem se poziva jedna od gore navedenih funkcija, u ovisnosti o tome na koji botun se klikne (u jednom slučaju se uklanja modal, a u drugom se korisnika vodi (navigate) na hardkodirani url.)
Realizacija modala se mogla pojednostavniti bez korištenja useRef i useEffect, no tu je da se pokažu i te funkcionalnosti Reacta.