REACT

4. forduló

A kategória támogatója: TCS - Tata Consultancy Services

Ismertető a feladathoz

A 4. forduló után elérhetőek lesznek a helyezések %-os formában: azaz kiderül, hogy a kategóriában a versenyzők TOP 20% - 40% -60% -ához tartozol-e!

Szeretnénk rá felhívni figyelmedet, hogy a játék nem Forma-1-es verseny! Ha a gyorsaságod miatt kilököd a rendesen haladó versenyzőket, kizárást vonhat maga után!

Felhasznált idő: 00:00/15:00 **Elért pontszám:** 0/7

1. feladat 0/2 pont

Vizsgáljuk meg az alábbi egyszerű App-ot:

```
export const UserContext = React.createContext();

export default function App() {
  const [kind, setKind] = React.useState('rgb');
  const colors = ['red', 'green', 'blue'];

  return (
    <UserContext.Provider
      value={{
        kind,
        onChangeKind: setKind,
      }}
    >
      <Message />
      <ColorList colors={colors} />
    </UserContext.Provider>
  );
}

function Message() {
  const getColor = () => Math.floor(Math.random() * 255);
  const style = { color: `rgb(${getColor()},${getColor()},${getColor()})` };
  return <h4 style={style}>Random</h4>;
}
```

```
function ColorList({ colors }) {
  return (
    <>
      <Message />
      {colors.map((m) => (
        <Color name={m} />
      ))}
    </>
  );
}

function Color(props) {
  const ctx = React.useContext(UserContext);

  return (
    <div>
      {ctx.kind} : {props.name}{' '}
      <button onClick={() => ctx.onChangeKind(props.name)}>ch</button>
    </div>
  );
}
```

Miért változik meg minden alkalommal, amikor "ch"-t nyomunk a szöveg (Message) komponens által kiírt szöveg színe?

Válasz

- ○ Csak az egyik Message komponens színe változik
- ○ Nem változik meg
- ○ Csak annak a Message komponens színe változik meg, amelyik a ColorList-ben van
- ● Ez a ContextAPI "dirty little secret"-je: bármi, ami használja a context-et újra renderelődik, ha a contextben változik valami
  Ez a válasz helyes, de nem jelölted meg.
- ○ Ezt a viselkedést ContextAPI-ban ki lehet "kapcsolni"

Magyarázat

Az elvárt viselkedés az alábbi dokumentációban található meg: https://reactjs.org/docs/context.html#caveats, azaz amennyiben a context értéke megváltozik akkor bármi, ami használja a context-et újra renderelődik. Tehát mindegyik **Message** komponens szine megváltozik a ch gomb megnyomásakor.

Ezért a "Ez a ContextAPI "dirty little secret"-je: bármi, ami használja a context-et újra renderelődik, ha a contextben változik valami" válasz **helyes.**

2. feladat 0/2 pont

Vizsgáljuk meg alábbi app-ot:

```
export default function AppA() {
    const [counter, setCounter] = useState(0);

    const inc = () => {
        for (let i = 0; i < 3; ++i) {
            setCounter(counter + 1);
        }
```

```
            alert(counter);
        }

        return (
            <div>
                <div>{counter}</div>
                <button onClick={inc}>inc</button>
            </div>
        );
    }
```

Mi történik ha az "inc" gomb-ra 2x kattintunk az eredeti implementációban illetve ha kicseréljük a **setCounter(counter + 1);** sort -> **setCounter(prev => prev + 1);-**re. Az Alert-et minden alkalommal "okézzuk" ?

Válasz

- ○ eredetiben [alert=2, counter=2], modosítás után [alert=3, counter=6]
- ○ eredetiben [alert=1, counter=2], modosítás után [alert=3, counter=9]
- ○ eredetiben [alert=4, counter=4], modosítás után [alert=1, counter=3]
- ● eredetiben [alert=1, counter=2], modosítás után [alert=3, counter=6]
  Ez a válasz helyes, de nem jelölted meg.
- ○ eredetiben [alert=5, counter=2], modosítás után [alert=6, counter=12]
- ○ eredetiben [alert=1, counter=3], modosítás után [alert=6, counter=6]

Magyarázat

A ***setCounter*** működése *async* azaz egy *for loop*-ban használva az érték amit változtat nem updatelődik azonnal a loop után.

Ezért az eredeti implementációban az első kattintás lefutásakor a **counter** értéke még 0, csak a következő kattintásnál változik meg 1-re, ezért lesz az **alert=1.**

A **counter** értéke hiába van növelve 3-szor egy ciklusban, mivel a **counter** változó az első kattintás után végig 0-t tartalmaz ezért az első inc hivás végén 1 lesz, amit aztán a második kattintás növel meg 2-re, ezért lesz a **counter=2**.

A második implementációban a **setCounter(prev => prev + 1);** használatával mindig megkapjuk a counter state "belső" aktuális értékét és így az ismételt cikluson belüli hívás növelni fogja azt.

Az alert esetében az első kattintás lefutásakor a **counter** értéke itt is még 0 (mivel a setCounter továbbra is async), de a következő kattintásnál ez már megváltozik 3-ra, ezért lesz az **alert=3.**

A **prev** használatával mindig a növelt értéket állítjuk be, ezért a két lefutás alatt 6-szor növeljük meg a coutner-t ezért lesz a **counter=6.**

Ezért a **eredetiben [alert=1, counter=2], modosítás után [alert=3, counter=6]** válasz **helyes**.

3. feladat 0/3 pont

Adott az alábbi kötséges függvény:

```
function calcExpensive(numbers) {
    let res = 0;
    numbers.forEach(num => {
        for (let i = Math.pow(num, 7); i >= 0; i--) {
            res += Math.atan(i) * Math.tan(i);
        };
    });
```

```
        return res;
    }
```

Hogyan tudjuk elkerülni ennek a függvénynek az ismételt lefutását amennyiben a numbers értéke nem változik?

Válasz

- [ ]
```
const sum = useMemo(() => calcExpensive(numbers));
```

- [ ]
```
const sum = useMemo(() => calcExpensive(numbers), []);
```

- [ ]
```
const sum = useRef(() => calcExpensive(numbers));
```

- [ ]
```
const sum = useMemoize(() => calcExpensive(numbers), [numbers]);
```

- [ ]
```
const sum = useRef(() => calcExpensive(numbers), [numbers]);
```

- [x]
```
const sum = useMemo(() => calcExpensive(numbers), [numbers]);
```

Ez a válasz helyes, de nem jelölted meg.

Magyarázat

A **useMemo** horog alkalmas arra, hogy egy számítás értékét "megjegyezzük" lásd https://reactjs.org/docs/hooks-reference.html#usememo

A használata hasonló az **useEffect**-ez azaz a második paramétere a függőségi tömb: amikor ennek értéke megváltozik akkor a megadott függvény újra kiértékelődik.

Ezért a **const sum = useMemo(() => calcExpensive(numbers));** válasz **helytelen**, mivel ha nem adunk meg függőségi tömböt, akkor a megadott függvény minden render esetén kiértékelődik.

A **const sum = useMemo(() => calcExpensive(numbers), []);** válasz **helytelen**, mivel az üres függőségi tömb nem fog sosem megváltozni, ezért hiába változik a numbers értéke a sum mindig ugyanaz marad.

A **const sum = useRef(() => calcExpensive(numbers));** válasz **helytelen** mivel az useRef horog egy változtatható objektumot ad vissza nem pedig egy számítás eredményét tárolja el.

A **const sum = useMemoize(() => calcExpensive(numbers), [numbers]);** válasz **helytelen** mivel nincsen **useMemoize** nevő beépített horog a React-ban.

A **const sum = useRef(() => calcExpensive(numbers), [numbers]);** válasz **helytelen** lásd fent.

A **const sum = useMemo(() => calcExpensive(numbers), [numbers]);** válsz **helyes** mivel a függőségi tömb helyesen tartalmazza a numbers paramétert.

KÉSZÍTETTE cone

Megjelenés

Világos