REACT.JS

4. forduló

accenture

A kategória támogatója: Accenture

RENDELKEZÉSRE ÁLLÓ IDŐ:

15:00

Ismertető a feladathoz

Fontos információk

A forduló után a megoldások publikálásával együtt iránymutatásként elérhetőek lesznek a **helyezéssel kapcsolatos információk**, látni fogod, hogy a kategóriában a játékosok 20%, 40% vagy 60%-a közé tartozol-e épp.

Felhívjuk figyelmedet, hogy a következő, **5. fordulótól az egyes kategóriák csak a kijelölt napokon lesznek megoldhatóak 7-22 óra között**, érdemes letöltened a naptárat a Kategóriáim menüpontban.

Felhasznált idő: 01:49/15:00 **Elért pontszám:** 0/7

1. feladat 0/2 pont

A React a Virtual DOM összehasonlítása során az alábbi esetek közül mikor bont le és épít újra (umount/mount) egy részfát a DOM-ban is?

Válaszok

- [x] Ha két összehasonlított részfa gyökér komponensének a típusa megváltozik pl. div -> span
- [ ] Ha két összehasonlított DOM element (pl. div) túl sok tulajdonsága változik meg, akkor a React dönthet úgy, hogy újraépíti a részfát a DOM-ban
- [x] Ha két összehasonlított részfa **key** tulajdonsága nem egyezik
- [ ] Ha két összehasonlított saját komponens túl sok tulajdonsága változik meg, akkor a React dönthet úgy, hogy újraépíti a részfát a DOM-ban
- [ ] Ha két összehasonlított részfa gyermek komponenseinek száma megváltozik

Magyarázat

A "Ha két összehasonlított részfa gyökér komponensének a típusa megváltozik pl. div -> span" válasz helyes, mert ha egy részfa gyökér komponens típusa megváltozik, akkor az a részfa mindenképpen újraépül a DOM-ban.

A "Ha két összehasonlított DOM element (pl. **div**) túl sok tulajdonsága változik meg" válasz helytelen, mert ha a DOM típusa nem változik meg, akkor a React nem fogja lebontani a DOM element-et csak a megváltozott tulajdonságokat aktualizálja, függetlenül attól, hogy azokból mennyi változott meg.

A "Ha két összehasonlított részfa **key** tulajdonsága nem egyezik" válasz helyes, mert a key tulajdonság megváltozása mindenképpen újraépítést jelent. Ezt specális esetekben ki is lehet használni: https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html#recommendation-fully-uncontrolled-component-with-a-key

A "Ha két összehasonlított saját komponens túl sok tulajdonsága változik meg" válasz helytelen, mert hasonlóan a DOM tulajdonságok megváltozásához ebben az esetben sem bontja le a React a komponens részfát.

A "Ha két összehasonlított részfa gyermek komponenseinek száma megváltozik" válasz helytelen, mert a gyermek komponensek számának változása nem eredményezi egy egész részfa újraépítését, csak a megváltozott gyermekek (hozzáadott, kitörölt) módosulnak a DOM-ban.

2. feladat 0/2 pont

Melyik állítások igazak az alábbi magasabb rendű komponensre?

```
function withLogging(WrappedComponent) {
  WrappedComponent.prototype.componentDidUpdate = function (prevProps) {
    console.log("Current props: ", this.props);
    console.log("Previous props: ", prevProps);
  };

  ...
  return class extends React.Component() {
    async componentDidMount() {
      await logExternally("ComponentLoaded", WrappedComponent);
      console.log("component mounted");
    }

    async log(message) {
      await logExternally(message);
      console.log("message logged");
    }

    render() {
      return <WrappedComponent {...this.props} logger={this} />;
    }
  };
}
```

Válaszok

- [ ] **withLogging** helyesen átadja az *összes* kapott tulajdonságot a beburkolt komponensnek
- [x] A burkolt komponens eléri a **log** metódust a **props.logger** tulajdonságon keresztül

- [ ] A **withLogging** helyesen burkolja be a **componentDidUpdate**-et
- [ ] A **withLogging** helytelenül burkolja be **componentDidMount**-et, mert az nem lehet **async**
- [x] A **withLogging** nem támogatja olyan komponensek helyes beburkolását, amik osztályszintű statikus metódusokkal rendelkeznek

Magyarázat

Az "**withLogging** helyesen átadja az összes kapott tulajdonságot" válasz helytelen, mivel a ref tulajdonságot nem adja át a burkolt komponensnek, ehhez a **React.ForwardRef** használata lenne szükséges.

A "A burkolt komponens eléri a **log** metódust a **props.logger** tulajdonságon keresztül" válasz helyes, mert a burkolt komponensek kaphatnak új tulajdonságokat a beburkolás során.

A "A **withLogging** helyesen burkolja be a **componentDidUpdate**-et" válasz helytelen, mert a withLogging helytelenül felülírja a burkolt komponens componentDidUpdate-et, ezzel elveszítve annak eredeti működését.

A "A **withLogging** helytelenül burkolja be **componentDidMount**-et, mert az nem lehet **async**" válasz helytelen, mert az **async** nem okoz problémát, a componentDidMount helyesen lefut és az eredeti működése megmarad a burkolt komponensnek.

Az "A **withLogging** nem támogatja olyan komponensek helyes beburkolását, amik osztályszintű statikus metódusokkal rendelkeznek" válasz helyes, mert az előre ismert statikus metódusokat kézzel vagy segédkönyvtárakkal, mint a hoist-non-react-statics kell átmásolni.

3. feladat 0/2 pont

Adott az alábbi két komponens, ahol azt figyelhetjük meg, hogy minden alkalommal, amikor a Parent komponens újra render-elődik, akkor a Children komponens is újra render-elődik, amit el szeretnénk kerülni. Az alábbi lehetőségek közül hogyan érhetjük el ezt?

```
function Parent() {
  let [value, setValue] = React.useState(1);
  return (
    <div>
      <div>Parent value: {value}</div>
      <button onClick={() => setValue((v) => v + 1)}>Add to parent</button>
      <Children setParent={(value) => setValue(value)} />
    </div>
  );
}

function Children(props) {
  let [value, setValue] = React.useState(1);
  return (
    <div>
      <div>Children value: {value}</div>
      <button onClick={() => setValue((v) => v + 1)}>Add to children</button>
      <button onClick={() => props.setParent(value)}>Set from children</button>
    </div>
  );
}
```

Válaszok

- [ ] A **Children** komponenst osztállyá alakítjuk és megvalósítjuk a **shouldComponentUpdate** metódust, ahol vizsgáljuk, hogy a **setParent** nem változik. Ebben az esetben a Parent komponenst nem kell módosítani.
- [ ] A **Children** komponenst **React.memo** csomagoljuk és a **setParent** értékét kiemjük egy **const** függvénybe: **const setParent = (value) => setValue(value)** majd ezt adjuk át **<Children setParent={setParent} />**
- [x] A **Children** komponenst **React.memo** csomagoljuk és a **setParent** értékét **React.useCallback**-be csomagoljuk: **const setParent = React.useCallback((value) => setValue(value))** majd ezt adjuk át **<Children setParent={setParent} />**
- [x] A **Children** komponenst **React.memo** csomagoljuk és a **setValue**-t közvetlenük adjuk át **<Children setParent={setValue} />**

Magyarázat

A "A **Children** komponenst osztállyá alakítjuk" megoldás helytelen: hiába vizsgáljuk a **setParent** egyenenlőségét, mivel a **Parent** komponenstől mindig egy új függvény példányt kapunk.

A "A **Children** komponenst React.memo csomagoljuk és a **setParent** értékét kiemjük egy const függvénybe" megoldás helytelen: hiába emeljük ki a függvényt egy konstansba, az minden render során újra definiálódik, ezért a **React.memo** nem tud megfelelően működni.

A "A **Children** komponenst **React.memo** csomagoljuk és a **setParent** értékét **React.useCallback**-be csomagoljuk" megoldás helyes: a **useCallback** biztosítja, hogy mindig ugyanazt a függvénypéldányt adjuk át a **Children**-nek.

A " A **Children** komponenst **React.memo** csomagoljuk és a setValue közvetlenük adjuk át" megoldás helyes: a **React.useSate** által visszaadott értékadó függvény nem változik meg a render-ek során, ezért ezek átadhatóak a **Children**-nek.

4. feladat 0/1 pont

Az alábbi Portal komponens nem működik helyesen: a gyermek komponenseit "elveszíti" és nem jeleníti meg, ha a Portal-t tartalmazó komponens újra render-elődik. Hogyan lehetne ezt a hibát kijavítani?

```
function Portal({ children }) {
  const portalRoot = document.body;
  const el = React.useRef(document.createElement("div"));

  React.useEffect(() => {
    portalRoot.appendChild(el.current);

    return () => portalRoot.removeChild(el.current);
  }, []);

  return ReactDOM.createPortal(children, el.current);
}
```

Válasz

- [ ] a **createPortal** nem támogatott funkcionális komponensekben, mindenképpen át kell alakítani osztály komponenssé

- a **portalRoot** nem lehet a document.body, csak olyan DOM element lehet a **createPortal**-nak választani, ami a React alkamazás gyökér DOM eleme alatt található
- **React.useRef** használatával megoldható, hogy az el DOM element újra használjuk
- A **Portal** komponenst **React.Memo**-ba kell csavarni, hogy a DOM elmeket újra tudjuk használni

Magyarázat

A "**createPortal** nem támogatott funkcionális komponensekben" válasz helytelen, mert nicsen ilyen megkötés, a **createPortal** használható funkció komponensekben is.

A "a **portalRoot** nem lehet a document.body" válasz helytelen, mert nincsen ilyen megkötés, a **createPortal** lényege, hogy bárhová lehet vele a DOM-ba render-elni

A "**React.useRef** használatával" válasz helyes: a problémát az okozza, hogy minden render-nél egy új el DOM elem jön létre, ezt tudjuk kiköszöbölni az useRef használatával.

A "A **Portal** komponenst **React.Memo**-ba kell csavarni" válasz helytelen, mert a **React.Memo** nem akadályozza meg, hogy új el jöjjön létre, ha a komponens mégis újra render-elődik, így a Portal tartalma továbbra is elveszik.