REACT.JS

2. forduló

accenture

A kategória támogatója: Accenture

RENDELKEZÉSRE ÁLLÓ IDŐ:

12:00

Ismertető a feladathoz

Fontos információk

Ha kifutsz az adott feladatlap kitöltésére rendelkezésre álló időből, a felület **automatikusan megpróbálja beküldeni** az addig megadott válaszokat

A kérdésekre **mindig van helyes válasz**! Ha csak egy helyes válasz van az adott kérdésre, radio button-os választási lehetőségeket fogsz látni.

Olyan kérdés viszont nincs, amelyre az összes válasz helyes!

Egyéb információkat a versenyszabályzatban találsz!

Második forduló

Tovább folytatjuk utunkat a React.js világában.

Felhasznált idő: 01:48/12:00 **Elért pontszám:** 0/5

1. feladat 0/1 pont

Az alábbi komponens kezdetben helyesen növeli a számláló címsor értékét. Mi történik a számláló értékével a **h1**-ben, ha a felhasználó megváltoztatja a **Test** stringet a beviteli mezőben?

```jsx
function Counter() {
  let [state, setState] = React.useState({ counter: 0, input: "Test" });
  return (
    <>
      <div>{state.counter}</div>
      <input
        value={state.input}
        onChange={(e) => setState({ input: e.target.value })}
      ></input>
      <button onClick={() => setState({ counter: state.counter + 1 })}>
        Increment
      </button>
    </>
  );
}
```

Válasz

- A két eseménykezelő független egymástól, ezért a számláló tovább működik
- Futás idejű hibát kapunk, amiért két függvény is módosítja ugyanazt az állapot változót
- A gomb és a beviteli mező felülírja egymás értékeit, ezért a számláló értéke eltűnik a címsorból
- A számláló értéke megmarad, de már nem növekszik tovább

Magyarázat

A **useState** nem olvasztja össze a kapott értékeket, ezért amikor a beviteli mező megváltozik, a counter értéke undefined lesz, ezért eltűnik a címsorból.

A "A két eseménykezelő független egymástól, ezért a számláló tovább működik" válasz helyteten, mert az eseménykezelők függetlenek, de a számláló nem fog tovább működni.

A "Futás idejű hibát kapunk ..." válasz helytelen, nem kapunk hibát, tetszőleges eseménykezelő módosíthatja ugyanazt az állapot változót.

A "A számláló értéke megmarad, de már nem növekszik tovább" válasz helytelen mert a számláló értéke nem marad meg.

2. feladat 0/1 pont

Mi a probléma az alábbi DataLoader komponenssel?

```
function DataLoader() {
  let [data, setData] = React.useState({});
  React.useEffect(() => {
    let fetchData = async () => {
      let response = await fetch(
        "https://jsonplaceholder.typicode.com/albums/1"
      );
      let album = await response.json();
      setData({ title: album.title });
    };
    fetchData();
  });

  return <div>{data.title}</div>;
}
```

Válasz

- ○ Futás idejű hibát kapunk, mert az **useEffect**-ben nem használhatunk async await-et
- ○ Használhatunk async await-et, de csak akkor, ha egész függvény async az **useEffect**-ben
- ◉ Az adat betöltés minden alkalommal lefut, amikor a komponens újra render-elődik
- ○ A kód helyesen működik, semmi probléma nincsen vele

Magyarázat

"Az adat betöltés minden alkalommal lefut, amikor a komponens újra render-elődik" válasz helyes, mert hiányzik a függőségi tömb (az **useEffect** második paramétere), ezért az adatbetöltés nem csak egyszer fog lefutni.

A "Futás idejű hibát kapunk..." válasz helytelen, nem kapunk semmilyen futás idejű hibát az async használata miatt.

A "Használhatunk async await-et, de csak akkor, ha egész függvény async az **useEffect**-ben" válasz helytelen, ha az **useEffect** függvény async, akkor a React figyelmeztetést ír ki, ezért nem javasolt a használata.

A "A kód helyesen működik, semmi probléma nincsen vele" válasz hibás, mert ugyan a kód kiírja a betöltött album címét, de feleslegesen újra meg újra beölti azt amikor a komponens újra render-elődik.

3. feladat 0/1 pont

Melyik állítások igazak a saját hook-okra?

Válasz

- ○ Tartalmazhatnak egyéb hook hívást elágazásban és ciklusban is
- ◉ A nevüknek mindenképpen a **use** szóval kell kezdődniük
- ○ Csak a legfelső root komponensben használhatók saját hook-ok, belső komponensekben már nem
- ○ Saját hook-ok csak beépített hook-okat hívatnak, másik saját hook-ot nem

Magyarázat

A "Tartalmazhatnak egyéb hook hívást elágazásban és ciklusban is" válasz helytelen, mert a saját hook-ra is ugyanazok a szabályok vonatkoznak mint a komponensekre, nem tartalmazhatnak hook hívást elágazásban vagy ciklusban.

A "Csak a legfelső root komponensben használhatók saját hook-ok, belső komponensekben már nem" válasz helytelen, mert bármilyen szinten lehet saját hook-okat használni

A "Saját hook-ok csak beépített hook-okat hívatnak, másik saját hook-ot nem" válasz helytelen, mert saját hook-ok hívhatnak egyéb saját hook-okat is.

4. feladat 0/2 pont

Melyik állítások igazak az alábbi Stopper komponensre, miután a felhasználó megnyomta a Start gombot?

```
function Stopper() {
  let [value, setValue] = useState(0);
  let [isRunning, setIsRunning] = useState(false);
  let [intervalId, setIntervalId] = useState();

  React.useEffect(() => {
    if (isRunning) {
      if (!intervalId) {
        let timerId = setInterval(() => {
          setValue(value + 1);
        }, 1000);
        setIntervalId(timerId);
      }
    } else {
      if (intervalId) {
        clearInterval(intervalId);
        setValue(0);
      }
    }
  }, [isRunning, intervalId]);

  return (
    <div>
      <div>{value}</div>
      <button onClick={() => setIsRunning(true)}>Start</button>
      <button onClick={() => setIsRunning(false)}>Stop</button>
    </div>
  );
}
```

Válaszok

- [ ] A komponens végtelen ciklusba kerül és a komponens végig 0-at mutat
- [ ] A komponens által mutatott érték növekszik, de a Stop gombbal nem lehet leállítani
- [x] A komponens által mutatott érték nem növekszik tovább 1-nél

- [ ] A Stop gomb visszaállítja a komponens értékét 0-ra és Start után már az értéke folyamatosan növekszik
- [x] A Stop gomb megnyomása után a Start gomb megnyomására már nem változik a komponens értéke, az végig 0 marad

Magyarázat

A "A komponens végtelen ciklusba kerül és a komponens végig 0-at mutat" válasz helytelen, mert a komponens nem kerül végtelen ciklusba és a mutatott érték is változik.

A "A komponens által mutatott érték növekszik, de a Stop gombbal nem lehet leállítani" válasz helytelen, mert a **setValue**-t helytelenül használjuk, ezért a setInterval mindig az utolsó render-beli value értéket tartja meg, ami nem lesz nagyobb mint 1.

A "A Stop gomb visszaállítja a komponens értékét 0-ra és Start után már az értéke folyamatosan növekszik" válasz helytelen, mert az implementáció hibás és nem törli ki az **intervalId** értékét. Ezért a következő Start úgy érzékeli, hogy még fut az előző, ezért nem kezdi újra a számlálást.