REACT

3. forduló

A kategória támogatója: TCS - Tata Consultancy Services

Ismertető a feladathoz

A 3.forduló feladatait a hosszú hétvége miatt kivételesen szerda (11.02.) éjfélig tudod megoldani!

Érdemes ebben a fordulóban is játszanod, mert a következő forduló kezdetekor, 11.03-án 18 órától kiosztjuk az 1.-2.-3. fordulóban megszerzett badgeket!

A verseny közben az alábbi teljesítményeket díjazzuk:

- fordulógyőztes
- átlagnál jobb időeredmény
- átlag feletti pontszám
- hibátlan forduló

Szeretnénk rá felhívni figyelmedet, hogy az egyszer megkapott badge-eket nem vonjuk vissza, akkor sem, ha esetleg az adott fordulóban a visszajelzések alapján változások vannak.

Jó játékot!

Felhasznált idő: 00:00/20:00 **Elért pontszám:** 0/8

1. feladat 0/2 pont

Az alábbi "**legacy**" Class-Based komponenst átírtuk "**modern**" Hook-osra:

```
export default class Comp extends React.Component {
    constructor(props) {
        super(props);
        this.state = { counter: 0, input: 'Test' };
    }
    render() {
        return (
            <div>
                <div>{this.state.counter}</div>
                <input
                    value={this.state.input}
                    onChange={(e) => this.setState({ input: e.target.value })}
                />
                <button onClick={() => this.setState({ counter: this.   state.counter + 1 })}>
```

```
                    Inc
                </button>
            </div>
        );
    }
}
```

átírás után (**Hook**)

```
export default function Comp() {
    let [state, setState] = useState({ counter: 0, input: 'Test' });
    return (
        <>
        <div>{state.counter}</div>
        <input
            value={state.input}
            onChange={(e) => setState({ input: e.target.value })}
        />
        <button onClick={() => setState({ counter: state.counter + 1 })}>
            Inc
        </button>
        </>
    );
}
```

Működik!? Ha nem, miért nem?

Válaszok

- [ ] Igen, működik
- [x] Igen, működik, egy darabig...
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Egyáltalán nem működik, mert syntax error-t ad
- [x] Nem működik jól, mert A button és az input eseménykezelők felülírják a ***state***-t
  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

A React.Component "**this.setState**" metódusa és az **useState** hook által visszaadott setter függvény különbözően működik. A class komponenseknél a **setState** autómatikusan összefésüli (merge) az előző állapotot az új állopottal, míg az useState setter függvénye teljesen lecseréli a régi állapotot az újra.

Ezért a függvény komponensben a **setState** hívások kitörlik az előzőleg beállított input és counter értékeket.

Ezek alapján:

Az "Igen, működik" válasz **helytelen,** mivel az Inc gomb megnyomása után, ha változik a beviteli mező a számláló értéke elveszik.

Az "Igen, működik, egy darabig..." válasz **helyes**, mivel amennyiben csak az Inc gomb-ot nyomjuk meg és a bevitelő mezőt nem változtatjuk a számláló értéke helyesen nő.

A "Egyáltalán nem működik, mert syntax error-t ad" válasz **helytelen**, a kód szintaktikailag helyes.

A "Nem működik jól, mert A button és az input eseménykezelők felülírják a ***state***-t" válasz **helyes**.

2. feladat 0/3 pont

Az alábbi output-ot kapjuk a console-ban, amikor az "email"-fieldbe gépeljük (gyorsan) azt, hogy "apple":

```
| Output   |
| -------- /
| cleanup  |
| cleanup  |
| cleanup  |
| cleanup  |
| cleanup  |
| checking... |
```

Melyik alábbi effect kódja eredményezi ezt és megy át egy "code review-n"?

Válasz

- [x]
```
useEffect(
        () => {
            const timeoutHandler = setTimeout(() => {
                console.log('checking...');
            }, 500);

            return () => {
                console.log('cleanup');
                clearTimeout(timeoutHandler);
            }
        },
        [emailField]
    );
```
Ez a válasz helyes, de nem jelölted meg.

- [ ]
```
useEffect(
        () => {
            const timeoutHandler = setTimeout(() => {
                console.log('checking...');
                clearTimeout(timeoutHandler);
            }, 500);

            return () => {
                console.log('cleanup');
                clearTimeout(timeoutHandler);
            }
        },
        [emailField]
    );
```

- [ ]
```
useEffect(
        () => {
            const timeoutHandler = setTimeout(() => {
                console.log('checking...');
            }, 500);
        },
        [emailField]
    );
```

Magyarázat

A "**useEffect**"-nél megadott függvény visszatérési értéke egy másik függvény ami a "**cleanup**"-ot valósítja meg, ebben az esetben a "**setTimeout**" által visszaadott "**handler**"-t törli, ez szükséges ahhoz hogy a helyes output-ot kapjuk.

Ezért a,

```
useEffect(
        () => {
            const timeoutHandler = setTimeout(() => {
                console.log('checking...');
            }, 500);

            return () => {
                console.log('cleanup');
                clearTimeout(timeoutHandler);
            }
        },
        [emailField]
    );
```

válasz **helyes**.

A

```
useEffect(
        () => {
            const timeoutHandler = setTimeout(() => {
                console.log('checking...');
                clearTimeout(timeoutHandler);
            }, 500);

            return () => {
                console.log('cleanup');
                clearTimeout(timeoutHandler);
            }
        },
        [emailField]
    );
```

válasz **helytelen**, mivel a **clearTimeout(timeoutHandler);** feleslegesen szerepel a **timeoutHandler**-ben, ezért nem menne át egy **"code review-n".**

A

```
useEffect(
        () => {
            const timeoutHandler = setTimeout(() => {
                console.log('checking...');
            }, 500);
        },
        [emailField]
    );
```

válasz **helytelen**, mivel nem tér vissza a **cleanup** függvénnyel, így nem kapjuk meg az elvárt output-ot.

3. feladat 0/3 pont

Az alábbi egyszerű app nem működik helyesen:

```
export default function App() {
  const [numbers, setNumbers] = useState([]);

  useEffect(() => {
    console.log('effect...');
    setTimeout(() => {
      setNumbers([1,2,3])
    }, 1000)

  }, []);

  const addOne = useCallback(() => {
    setNumbers([...numbers, numbers.length + 1]);
  }, []);

  return (
    <div>
      <div>Numbers: {JSON.stringify(numbers)}</div>
      <button onClick={addOne}>Add</button>
    </div>
  );
}
```

Melyik alábbi módosítás teszi lehetővé a helyes működést, és azon túl a leghatékonyabb?

Válasz

- ```
  const addOne = useCallback(() => {
      setNumbers([...numbers, numbers.length + 1]);
    }, [numbers]);
  ```

- ```
  const addOne = useCallback(() => {
      setNumbers(prev => [...prev, prev.length + 1]);
    }, []);
  ```

  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

A

```
const addOne = useCallback(() => {
    setNumbers([...numbers, numbers.length + 1]);
  }, [numbers]);
```

válasz **helytelen**

Mivel "**spread**"-elni a "**numbers**" nevű tömböt nem hatékony, mert minden új **numbers** értékre lefut az **useCallback** és egy új függvényt készítünk.

Ezzel szembe állítva lehet használni a react által adott latest snapshot verziót a "**prev**"-el (függvény formában), mely garantálja a leghatékonyabb működést (így nem készülnek új függvénnyek mert az useCallback belseje csak egyszer fut le)

Ezért a

```
const addOne = useCallback(() => {
    setNumbers(prev => [...prev, prev.length + 1]);
  }, []);
```

válasz **helyes**.