REACT.JS

3. forduló

accenture

A kategória támogatója: Accenture

RENDELKEZÉSRE ÁLLÓ IDŐ:

15:00

Ismertető a feladathoz

Fontos információk

> Ezután a forduló után automatikusan jár a kitartóknak szóló garantált ajándékunk, érdemes kitöltened a feladatlapot! :)
>
> Ha kifutsz az adott feladatlap kitöltésére rendelkezésre álló időből, a felület **automatikusan megpróbálja beküldeni** az addig megadott válaszokat.
>
> A kérdésekre **mindig van helyes válasz,** olyan kérdés viszont nincs, amelyre az összes válasz helyes!

Egyéb információkat a versenyszabályzatban találsz!

Harmadik forduló

Tovább folytatjuk utunkat a React.js világában.

Felhasznált idő: 01:49/15:00 **Elért pontszám:** 0/5

1. feladat 0/2 pont

Melyik állítások igazak a hibahatárokra? (Error Bundaries)

Válaszok

- [x] Funkció komponensek is lehetnek hibahatárok a **useErrorBoundary** hook segítségével
- [ ] Csak egyetlen hibahatár komponens definiálható egy React alkalmazáson belül
- [x] A hibahatárok csak a render és az életciklus eseménykezelőkben keletkezett kivételeket kapják el, pl. eseménykezelőkben kiváltott kivételeket nem
- [ ] Kötelező hiba UI-t is definiálni, amikor egy hibahatárt definiálunk
- [x] Ha nincsen hibahatár komponens használva és kivétel történik egy komponens render-elése során, akkor a legújabb production React a teljes React alakalmazást eltünteti

Magyarázat

A "Csak egyetlen hibahatár komponens definiálható egy React alkalmazáson belül" válasz helytelen, mert több hibahatár komponens is definiálható, ezek egymásba is ágyazhatóak.

A "Kötelező hiba UI-t is definiálni, amikor egy hibahatárt definiálunk" válasz hibás, mert nem közelező (csak ajánlott) hiba UI készíteni. A render metódus bármivel visszatérhet egy hibahatár komponensben.

2. feladat 0/0 pont

Az alábbi Example komponensben a **setRef** metódus kétszer hívódik meg: először null-al, utána magával az **input** DOM node-al. Hogyan lehet megoldani, hogy csak egyszer hívódjon meg?

```
class Example extends React.Component {
  constructor() {
    super();
    this.inputElement = null;
    this.state = {
      input: "",
    };
  }

  setRef = (el) => {
    console.log("callback is called", el);
  };

  render() {
    return (
      <>
        <input
          value={this.state.input}
          onChange={(e) => this.setState({ input: e.target.value })}
          ref={this.setRef}
        />
      </>
    );
  }
}
```

Válasz

- Sehogy sem, mindenképpen kétszer hívódik meg a **setRef**
- **React.Strict** mód használatával
- bind-olni kell a **setRef** metódust az osztályhoz
- Az **onChange** eseménykezelő mindig újra difiniálódik, ezért ki kell emelni egy osztály szintű függvénybe

Magyarázat

Mivel a **setRef** függvény nincs bindolva, ezért kétszer hívódik meg a ref beállítása.

A "Sehogy sem, mindenképpen kétszer hívódik meg a **setRef**" megoldás helytelen, mivel létezik megoldás a problémára.

A "**React.Strict** mód használatával" megoldás helytelen, mert a Strict módnak nincsen hatása erre a problémára.

A "Az **onChange** eseménykezelő mindig újra difiniálódik, ezért ki kell emelni egy osztály szintű függvénybe" megoldás helytelen, mert igaz, hogy az **onChange** eseménykezelő újradifiniálódik, de ennek nincs köze a jelenséghez.

Frissítve (2021.11.08. 14:20): a feladatot 0 pontosra állítottuk, miután többen is jeleztétek, hogy hibás a példakód.

3. feladat 0/2 pont

Adott a következő komponens:

```
function MyInput(props) {
  return <input value={props.value} onChange={props.onChange}></input>;
}
```

Az alább felsoroltak közül melyikkel lehetséges egy szülő komponensben hivatkozást szerezzünk az input element-re?

Válaszok

- [ ] Mivel funkció komponensek nem kaphatnak **ref**-et, ezért class komponensé alakítjuk a **MyInput**-ot és a szülő komponensben a **ref** propertiben megadjuk a ref-et **<MyInput ref={this.refInParent} />** eztuán már használhatjuk **props.ref**-et a MyInput-on belül: **<input ref={props.ref} ... />**

- [x] **React.forwardRef-**be csavarjuk a komponenst és a kapott ref argumentumot használjuk a MyInput-on belül **React.ForwardRef((input, ref) => (<input ref={ref} ... />))** és a szülő komponenseben használhatjuk a ref-et **<MyInput ref={this.refInParent} />**

- [ ] **React.forwardRef**-be csavarjuk a komponenst és ezután már használhatjuk a **props.ref**-t a MyInput-on belül:

```
React.forwardRef(function MyInput(props) {
  return (
    <input
      ref={props.ref}
      value={props.value}
      onChange={props.onChange}
    ></input>
  );
});
```

  és a szülő komponenseben használhatjuk a ref-et **<MyInput ref={this.refInParent} />**

- [x] Egy saját tulajdonságon keresztül adjuk át a ref-et és azt használjuk a MyInput-ban:

```
function MyInput(props) {
  return (
    <input
      ref={props.inputRef}
      value={props.value}
      onChange={props.onChange}
    ></input>
  );
}
```

  és a szülő komponenseben használhatjuk a inputRef-et **<MyInput inputRef={this.refInParent} />**

Magyarázat

A "Funkció komponensek nem kaphatnak ref-et..." válasz helytelen: igaz ugyan, hogy funkció komponensek nem kaphatnak ref-et, de ilyenkor is szükség van a **React.ForwardRef**, mert a **props.ref** nem definiált.

A "**React.forwardRef**-be csavarjuk a komponenst és a kapott ref argumentumot használjuk" válasz helyes: a **React.ForwardRef** második argumentuma tartalmazza a ref értékét.

A "**React.forwardRef**-be csavarjuk a komponenst és ezután már használhatjuk a **props.ref**-t" válasz helytelen. mert a **React.forwardRef** egy függvényt vár 2 argumentummal, ahol a második argumentum a **ref**, a **props.ref** ilyenkor sem definiált.

A "Egy saját tulajdonságon keresztül adjuk át a ref-et" válasz szintén helyes, itt egy saját propery-n keresztük érjük el az input-ot, ezért nincs szükség a **React.ForwardRef**-re.

4. feladat 0/1 pont

Az alábbi komponens részletben egy nem kontrollált űrlapot látunk:

```
render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Name:
        <input type="text" ref={this.input} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

Hogyan lehet a beviteli mezőnek alapértelmezett értéket adni ("Enter your name") és megtartani az űrlap helyes működését?

Válasz

- Nem lehetséges az űrlap átírása nélkül, mindenképpen kontrollált mezőket kell használnunk, ha alapértelmezett értékeket szeretnénk.
- A **value="Enter your name"** tulajdonság használatával
- A **defaultValue="Enter your name**" tulajdonság használatával
- Az **initialValue="Enter your name"** tulajdonság használatával

Magyarázat

Az "Nem lehetséges az űrlap átírása nélkül" megoldás helytelen, mert a **defaultValue** tulajdonság pont erre az esetre lett kitalálva.

A **"value="Enter your name""** megoldás helytelen, mert kezdetben ugyan megjelenik a "Enter your Name", de a beviteli mezőt a felhasználó utána már tudja módosítani.

Az **initialValue="Enter your name**" megoldás helytelen, mert az **initialValue** nem egy beépített tulajdonság, semmilyen hatása nincsen.