Kódím.cz

/ Webové aplikace / Digitální akademie Web / React

11 **Async/await a ošetření chyb**

Ošetření chybových stavů API a použití async/await místo then()

Předchozí
Opakování: LeviExpress, druhá část

Nemá následující lekci

Použití try-catch pro ošetření fatálních chyb

Ošetření serverových chyb ve funkci používající `await` už máme vyřešené. Zbývá ošetřit chyby komunikace (např. nedostupný internet), který jsme v předchozí části lekce ošetřovali ve funkci `catch()`. Pro ošetření takovýchto chyb v případě použití `await` potřebujeme použít `try-catch` blok. To je další z konstrukcí JavaScriptu, se kterou jsme se ještě nesetkali. Tato konstrukce se používá pro ošetření chyb, které mohou nastat v kódu. Celá konstrukce vypadá tak, že kód, ve kterém může dojít k chybě, se uzavře do bloku `try { … }`. Za ním pak následuje blok `catch { … }`, který se provede v případě, kdy v bloku `try { … }` dojde k chybě. V takovém případě se provádění v bloku `try { … }` na řádku s chybou ukončí, zbytek bloku `try` se přeskočí a skočí se rovnou do bloku `catch`.

Následující kód z předchozí sekce:

```
const fetchData = () => {
  fetch('https://random.kodim.app/api/diceroll?act=shaky')
    .then((resp) => {
      if (resp.status === 200) {
        setErrorMessage(null);
        return resp.json();
      }

      if (resp.status === 500) {
        setErrorMessage('Server vrátil chybu.');
      } else if (resp.status === 503) {
        setErrorMessage('Server je přetížen.');
      }
```

```
    })
    .then((data) => {
      if (data !== undefined) {
        setRoll(data.result.number);
      }
    })
    .catch((error) => {
      console.error('Chyba komunikace se serverem:', error.message);
      setErrorMessage('Chyba komunikace se serverem');
    });
};
```

tedy můžete pomocí `async/await` a `try-catch` přepsat takto:

```
const fetchData = async () => {
  try {
    const resp = fetch('https://random.kodim.app/api/diceroll?act=shaky')
    if (resp.status === 200) {
      setErrorMessage(null);
    } else if (resp.status === 500) {
      setErrorMessage('Server vrátil chybu.');
      return;
    } else if (resp.status === 503) {
      setErrorMessage('Server je přetížen.');
      return;
    }

    const data = await resp.json();
    setRoll(data.result.number);
  } catch (error) {
    console.error('Chyba komunikace se serverem:', error.message);
    setErrorMessage('Chyba komunikace se serverem');
  }
};
```

Sekce finally

Kromě `try` a `catch` můžeme také použít sekci `finally`. Tento blok se vykonává vždy, ať už došlo k chybě nebo ne. Je to ideální místo, pokud potřebujeme něco *uklidit* poté, co je komunikace se serverem dokončena, ať už úspěšně nebo neúspěšně. Např. pokud máme v komponentě stav `loading`, který zobrazuje točící se kolečko při načítání dat, v sekci `finally` ho nastavíme na `false`, aby uživatel věděl, že komunikace se serverem už neprobíhá.

```
const fetchData = async () => {
  try {
    const resp = await fetch('https://random.kodim.app/api/diceroll?act=s
    if (resp.status === 200) {
      setErrorMessage(null);
    } else if (resp.status === 500) {
      setErrorMessage('Server vrátil chybu.');
      return;
    } else if (resp.status === 503) {
      setErrorMessage('Server je přetížen.');
      return;
    }

    const data = await resp.json();
    setRoll(data.result.number);
  } catch (error) {
    console.error('Chyba komunikace se serverem:', error.message);
    setErrorMessage('Chyba komunikace se serverem');
  } finally {
    setLoading(false);
  }
};
```

Předchozí
Asynchronní funkce

Následující
Bonus: Paralelní zpracování více požadavků

Kódím.cz

Verze 2.0.0-beta.7