# REACT - HOOKS

Hooks nám umožňují použít state a další React funkce bez psaní class komponent. Jsou kompletně volitelné, 100% zpětně kompatibilní, a je jen na nás jestli je budeme používat.

## PRAVIDLA PRO POUŽÍVÁNÍ HOOKS

Hooks jsou JavaScript funkce, ale musíme dodržovat dvě pravidla:

- Voláme je jen v nejvyšší úrovni function komponenty. Nevoláme je v cyklech, podmínkách nebo vnořených funkcích.
- Hooks voláme jen z function komponenty. Nevoláme je z klasických JavaScript funkcí. Ještě je můžeme volat z vlastní custom hook.

## useState

Hook pro použití statu. Vrací state hodnotu a funkci, pomocí které tuto hodnotu můžeme měnit. Jako parametr můžeme předat defaultní hodnotu.

```
// hook useState vrací pole obsahující
// hodnotu a funkci k její změnění
// -můžeme použít destrukturovací syntaxi
const [value, setValue] = useState(0);
```

## useEffect

Hook useEffect slouží ke stejnému účelu jako metody componentDidMount, componentDidUpdate a componentWillUnmount v class komponentách. Hodí se pro fetchování dat, nastavování odběrů a tak podobně.

Jako parametr předáváme funkci, která se bude volat po každém renderování (i po prvním).

```
useEffect(() => {
  document.title = `Počet: ${count}x`;
});
```

Pokud předaná funkce vrátí funkci, tak se tato funkce zavolá před dalším volání předané funkce nebo při unmountnutí komponenty.

```
useEffect(() => {
  const subscription = props.subscribe();
  return () => {
    // zde můžeme provést nějaký úklid
    subscription.unsubscribe();
  };
});
```

Pokud nechceme aby se předaná funkce volala po každém renderování, tak můžeme do useEffect předat také pole hodnot. Funkce se potom bude volat, když se v tomto poli změní hodnoty.

```
// tato funkce se spustí, jen když se
// změní hodnota props.source
useEffect(() => {
  const subscription = (
    props.source.subscribe()
  );
  return () => {
    subscription.unsubscribe();
  };
}, [props.source]);

// tato funkce se spustí jen jednou
useEffect(() => {console.log("1")}, []);
```

## useContext

Tato hook bere context objekt a vrací jeho aktuální hodnotu.

```
const value = useContext(MyContext);
```

## useRef

Pomocí hook useRef můžeme uložit hodnotu, kterou když změníme, tak se komponenta nebude rerenderovat.

```
const myRef = useRef(0);

// myRef jde používat jako normální ref
myRef.current = 1;
```

Nejvíce se refs používají k referencování elementů v HTML.

```
const inputEl = useRef(null);

return (
  <input ref={inputEl} type="text" />
);
```

## useMemo

Hook useMemo můžeme použít, když máme nějakou drahou operaci, kterou nechceme provádět při každém renderování.

Jako parametr předáváme funkci a pole hodnot. Předaná funkce se spouští jen v případě, že se změní hodnoty v poli. Hodnotu kterou předaná funkce vrátí se pomocí hook useMemo uloží a vrací se při každém renderování.

```
const value = useMemo(() => {
  return vypocitejHodnotu(a, b);
}, [a, b]);
```

## useCallback

Hook useCallback bere jako parametr funkci a pole hodnot. Funkce se pomocí useCallback uloží a bude se vracet při každém renderování namísto vytváření stále nové funkce. Funkce se vytvoří znovu jen v případě, že se změní některé z hodnot v předaném poli.

```
const funkce = useCallback(() => {
    doSomething(a, b);
}, [a, b]);
```

## CUSTOM HOOKS

Někdy můžeme chtít znovu použít nějakou stateful logiku mezi více komponentami. K tomu si můžeme vytvořit vlastní hook.

Custom Hook můžeme vytvořit jako JavaScript funkci, kterou na začátku pojmenuje s textem "use". Uvnitř této funkce můžeme používat jiné hooks. Custom Hook nemusí mít žádnou specifickou strukturu, je na nás co bude přijímat za argumenty a co bude vracet (pokud bude něco vracet).

```
const useFetch = (url) => {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch(url)
      .then((res) => res.json())
      .then((data) => setData(data));
  }, [url]);

  return [data];
};

export default useFetch;
```