React JS Hooks

1-useState: useState() är en inbyggd hook som låta oss använda state i våra komponenter .

Syntax:

```
Const [ value, setValue ] = useState(Initial value);
```

Note: useState() gör att komponenten rendera när state-value ändras.

Note2: Man ska ändra state värde genom att använda useState(nya värden).

Note3: Vi måste importera useState() för att kunna använda i komponenten så här:

```
import { useState } from "react";
```

Note4: State värde kan våra nummer, text, Array, object osv.

Exempel:

```
Const [ counter, setCounter ] = useState(0);
Const handleIncrease = () =>{
        setCounter(counter + 1);
}
Return (
<button onClick = {handleIncrease}>+</button>
<lable> Counter is {counter} <lable>
);
```

Note5: Vi kan använda hur många som helst vi behöver av useState() i en komponent.

Exampel2:

```
Const [ userName, setUserName ] = useState(“”);
Return(
<input name=’ userName’ onChange={(e)=> setUserName(
e.target.value)}/>
<lable> Välkommen {userName} </lable>
);
```

OR

```
Const handleSetUserName = (e) =>{
    setUserName(e.target.value);
}
Return(
    <input name=’ userName’
    onChange={handleSetUserName} />
    <lable> Välkommen {userName} </lable>
    );
```

Note6: Vi definierar useState() – och alla andra hooks- på Top-Level-Definition dvs direkt i vår funktion komponent och inte i någon loop eller sådant.

2- useEffect: useEffect() är en hook som låta oss köra kod eller funktion beroende på ändring i dennes dependencies, den tar två argument, den första är funktionen – och det kan våra en pilfunktion(Arrow FN), och den andra arg är en array som behåller dem variabler som vi vill vår funktion körs vid deras ändringar.

Syntax:

```
useEffect( function, [dependencies]);
```

Det kan vara Arrow :

```
useEffect( ()=>{
        vår kod
        }
, [dependencies]);
```

Note1: useEffect() har tre olika fall:

1- Den andra argument har dependency dvs en eller fler element i arrayn då körs koden vid första gången komponenten körs dvs när sidan laddas (inte vid rerendering) och vid ändring på en av dem dependencies.
2- Den andra argument är en tom array och då körs koden bara när sidan laddas eller laddas om.
3- Den andra argument inte finns och då körs koden vid rerendering.

3-useRef(): En hook som kan hålla referens till ett element i sidan då blir det lättare att komma åt den.

Syntax:

```
Const firstName = useRef();
Return(
<input ref={firstName}/>
<lable> Välkommen {firstName.current.value}
        )
```

Current har olika alternativ t ex focus osv.

4-useMemo(): Liknar useEffect(), men useMemo() returnerar ett värde.

Syntax:

```
Const value = useMemo ( function, [dependencies]);
```

Det kan vara Arrow :

```
Const value = useMemo( ()=>{
Return(
        <JSX>
        )                    }
, [dependencies]);
```

5-useContext() (ContextAPI):

Används ofta för att hantera globalt tillstånd i applikationer. Där lagras våra state och funktioner som vi kan nå i alla

komponenter och så kan man undvika att passera props från en komponent till en annan, vilket kan vara besvärligt om du har flera lager av komponenter. I stället kan man lägga till data i en "context".

Syntaxen :

Vi behöver först skapa context genom att använda createContext() metod som vi importerar från react:

```
import { createContext } from "react";
```

Sen skapar vi context:

```
const AppInfoContext = createContext();
```

Sen behöver vi skapa context Provider komponent:

```
const AppInfoProvider = ({ children }) => {
```

Children props här används för att kunna sen skicka App komponent som barn till context Provider Här definierar våra state och setState som behövs:

1- Ett värde:

```
const [showSideText, setShowSideText] = useState(false);
```

2- En metod:

```
  const handleShowSideText = () =>{
            setShowSideText (true);
  }
```

Efter det returnerar context Provider med alla state (värden) och metoder:

```
  return (
    <AppInfoContext.Provider
       value={{showSideText,
handleShowSideText }} >
```

```
      {children}//<App/>
    </AppInfoContext.Provider>

      export { AppInfoContext };
      export default AppInfoProvider;
```

Efter det importerar vi Provider i index.js komponent:

```
import AppInfoProvider from
'./components/context/AppInfoContext';
```

sen skickar vi App som barn:

```
root.render(
  <AppInfoProvider>
    <App />
  </AppInfoProvider>
```

För att använda dem i en komponent behöver vi importera contexten (AppInfoContext) från context-fil och useContext() hook från react:

```
import { useContext } from "react";
import { AppInfoContext } from
”../../components/context/AppInfoContext";
```

Sen hämtar vi värden:

```
const SidBar = () => {
  const {
    showSideText,
  handleShowSideText
  } = useContext(AppInfoContext);
```