Arbetsprocess i frontend

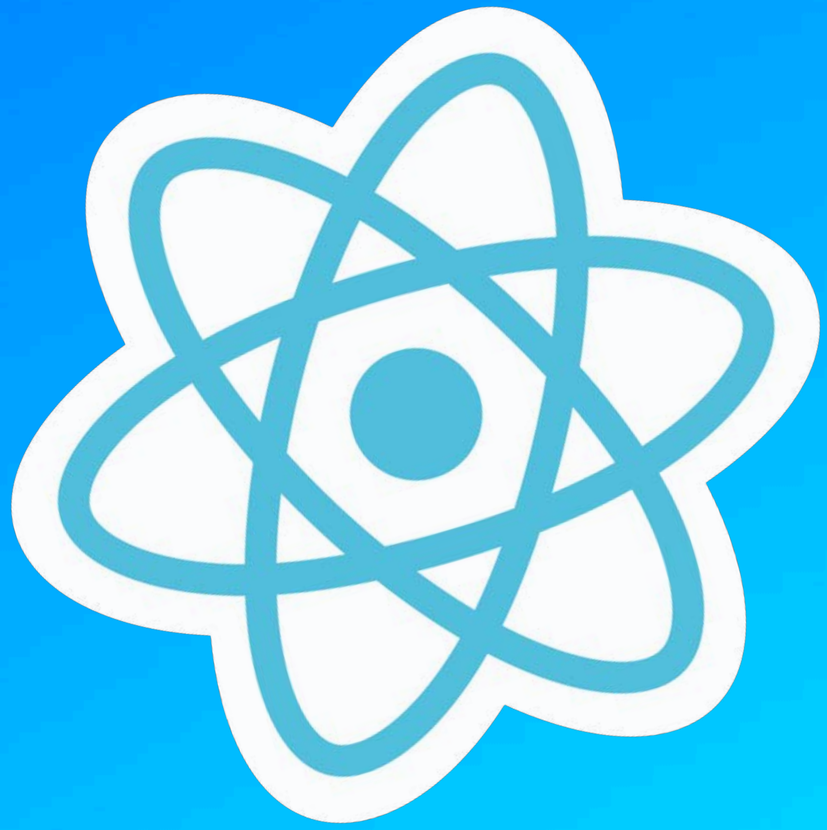
Processen

- Följer samma mall som att bidra med projektet i helhet
- Godkänd design innan implementation
- Demo i Figma

- Tillvägagångssätt beskrivet i README.md och Figma

React

Grunderna



Christoffer Billman 2023-04-14

Outline

01 Introduktion

02 Komponenter

03 Hooks

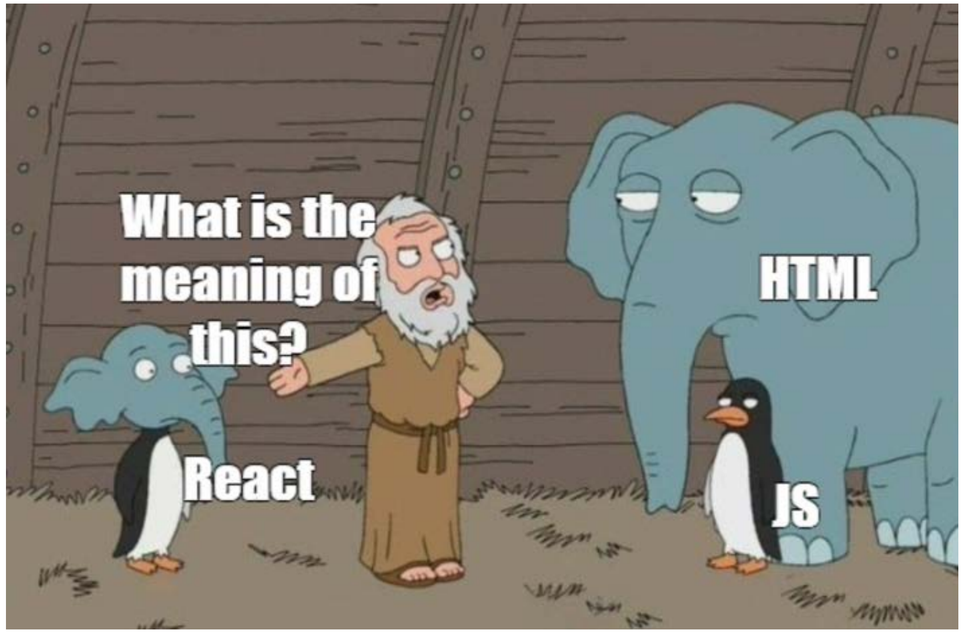
04 Rendering

01 Introduktion

React

Ett ramverk

- Javascript-ramverk för att snabbt skapa interaktiva webbapplikationer
- Kör i webbläsaren
- Istället för att separera HTML och JS så gör man allt i JS.
- Utökad version av Javascript (JSX)
- React-projekt kompileras till statisk HTML, CSS och JS.



NodeJS

Ett ramverk

- Javascript-ramverk för att köra JS utan en webbläsare
- Kompilering
- npm

Filstruktur

Demo

U1 Skapa ett React-projekt

https://nodejs.org

```
npx create-react-app workshop

cd workshop

npm start
```

02 Komponenter

Komponenter

Återanvändbara delar av gränssnittskod

- Delar av gränssnittet som går att återanvända
- Kan liknas funktioner i vanlig programmering
- En funktion som returnerar JSX

Ex. Komponenter

Skapa en komponent

```
function Button() {
    return (
        <div>
            <p>Jag är texten i en knapp!</p>
        </div>
    )
}
```

Använda en komponent

```
<Button/>
```

Props

Kontrollerar utseende och beteende

- Kan liknas parametrar/argument i funktioner

Ex. Props

Skapa en komponent

```
function Button(props) {
    return (
        <div>
            <p>{props.text}</p>
        </div>
    )
}
```

Använda en komponent

```
<Button text='Hej världen!'/>
```

Children

Komponenter kan ges andra komponenter

- Det går att skicka med andra komponenter till komponenter
- Görs via children
- Samma syntax som i HTML
- Komponenten väljer var children ska visas

Ex. Children

Skapa en komponent

```
function Button(props) {
    return (
        <div>
            {props.children}
        </div>
    )
}
```

Använda en komponent

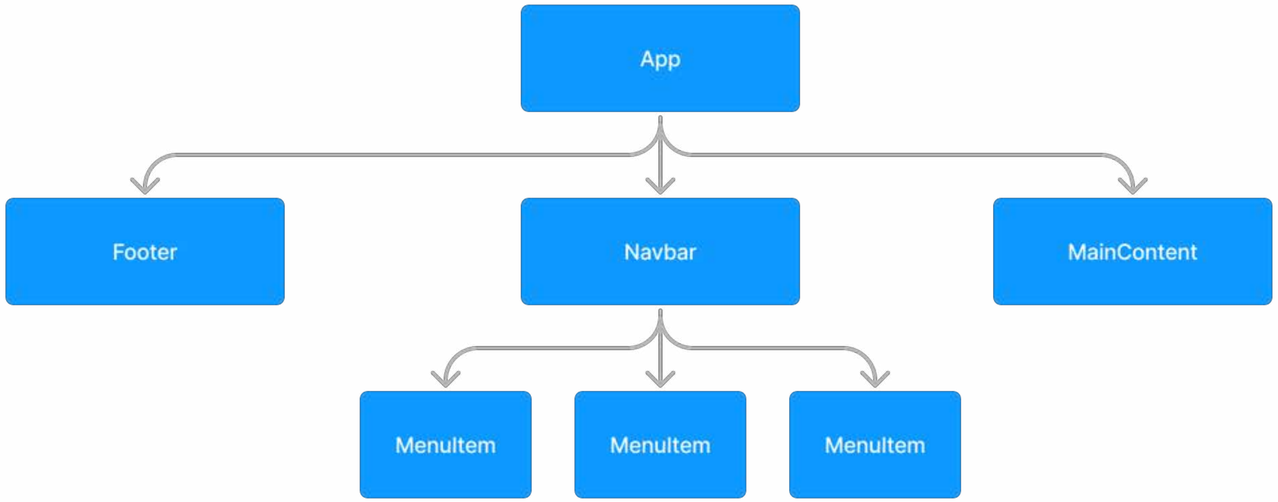
```
<Button>
    <p>Jag kommer visas i knappen!</p>
</Button>
```

U2 Skapa en egen komponent

Öppna mappen src, skapa en ny fil med *filnamn*.jsx

Komponentträd

- Komponenter nästlas i varandra
- Bildar ett komponentträd
- Hela React-appen består av detta komponentträd



03 Hooks

Hooks

Funktioner som låter oss ”haka in” i Reacts finesser

- Börjar alltid med ”use” i sitt namn.
- OBS: Får ej användas villkorligt eller i loopar.
- Ex. if(villkor) {useSomeHook()}

Tillståndsvariabler

Variabler som är knutna till gränssnittet

- Tillståndsvariabler är speciella variabler som automatiskt uppdaterar gränssnittet med det nya värdet.
- Varje gång en tillståndsvariabel ändras, så uppdateras gränssnittet.
- Hook: useState

useState

- Tar en parameter, initiala värdet av variabeln
- Returnerar en tupel
- Första är variabelns värde
- Andra är en funktion som man ändrar variablens värde med.
- OBS: En tillståndsvariabels värde får endast ändras med den givna funktionen.

Ex. useState

```
function Increment() {
    const [count, setCount] = useState(0)

    const incrementCount = () => {
        setCount(count + 1)
    }

    return (
        <div>
            <p>The value of count is: {count}</p>
            <button onClick={incrementCount}> Increment </button>
        </div>
    )
}
```

U3 Tillståndsvariabler

Återskapa exemplet, och utöka det om du vill!

Sidoeffekter

Att köra kod när en tillståndsvariabel ändras.

- Används främst för att synkronisera React med andra yttre system.
- Ex. Hämta från API, spara till localStorage etc.
- Hook: useEffect

useEffect

- Tar två parametrar
- En funktion
- En array med tillståndsvariabler. ”Dependency array”
- Körs när någon av tillståndsvariablerna ändras.

useEffect (2)

- Går oftast att göra på andra sätt.
- Leder ofta till oändliga loopar och kod som är svår att debugga.
- OBS: I utvecklingsläge kör den två gånger.

Ex. useEffect

```
function Increment() {
    const [count, setCount] = useState(0)

    useEffect(() => {
        console.log("This log was a side effect!")
    },[count])

    const incrementCount = () => {
        setCount(count + 1)
    }

    return (
        <div>
            <p>The value of count is: {count}</p>
            <button onClick={incrementCount}> +1 </button>
        </div>
    )
}
```

Ex. useEffect

```
1  useEffect(() => {
2          console.log("This effect runs only on first render!")
3      },[])
```

Ex. useEffect

```
1  useEffect(() => {
2          console.log("This effect runs on every re-render!")
3      })
```

Ex. useEffect

```
function Increment() {
    const [count, setCount] = useState(0)

    const incrementCount = () => {
        setCount(count + 1)
    }

    useEffect(incrementCount, [count])

    return (
        <div>
            <p>The value of count is: {count}</p>
            <button onClick={incrementCount}> +1 </button>
        </div>
    )
}
```

Contexts

Dela data mellan komponenter

- Jobbigt att explicit skicka data djupt ned i komponentträdet.
- ”Prop drilling”
- Ett alternativ till att skicka props
- Kan liknas med scopes I vanlig programmering.
- En funktion kan använda en variabel även om den inte var ett argument.
- Hook: useContext

Ex. Contexts

```
1  export const ThemeContext = createContext(true)
```

Skapa context i separat fil.

Ex. Contexts

```
1  import ThemeContext from './sökväg/till/themecontext'
2  function App(){
3      const [isDark, setIsDark] = useState(true)
4
5      return (
6      <ThemeContext.Provider value={isDark}>
7          <Button/>
8      </ThemeContext.Provider>
9      )
10 }
```

Ange vilka komponenter som kan använda contexten, sätt värde

Ex. Contexts

```
1  import ThemeContext from './sökväg/till/themecontext'
2  function Button() {
3      const isDark = useContext(ThemeContext)
4
5      return (
6          <div style={isDark ? {color: 'black'} : {color: 'white'}}>
7              <p>Klicka mig!</p>
8          </div>
9      )
10 }
```

Använd den

Observera att Button får värdet av isDark utan att ta props

04 Rendering

Rendering

Att visa gränssnittet på skärmen

- React representerar komponentträdet som ett objekt i JS (Virtual DOM)
- När en tillståndsvariabel ändras, måste detta komponentträdet översättas till HTML, och skrivas in i DOM-trädet för att ändringen skall visas.
- Kallas rendering

Villkorlig rendering

- Ibland så vill man visa olika saker beroende på något villkor
- Man använder sig av ternary-operatorn (?) i Javascript
- Ex.

```
1  condition ? console.log('was true!') : console.log('was false!')
```

Villkorlig rendering (2)

- Vill man rendera endast om något är sant, använd &&-operatorn.
- Kallas ”short circuiting”
- Ex.

```
1  condition && console.log('körs endast om conditon är sant!')
```

Ex. Villkorlig rendering

```
function Button(props) {
    return (
        <div>
            {props.shouldShowIcon ?
                <Icon/>
                :
                <SomethingElse/>
            }
        </div>
    )
}
```

```
function Button(props) {
    return (
        <div>
            {props.shouldShowIcon &&
                <Icon/>
            }
        </div>
    )
}
```

U4 Villkorlig rendering

Visa olika saker på skärmen, baserat på vad värdet av er tillståndsvariabel är

Det var allt!
Tack för mig