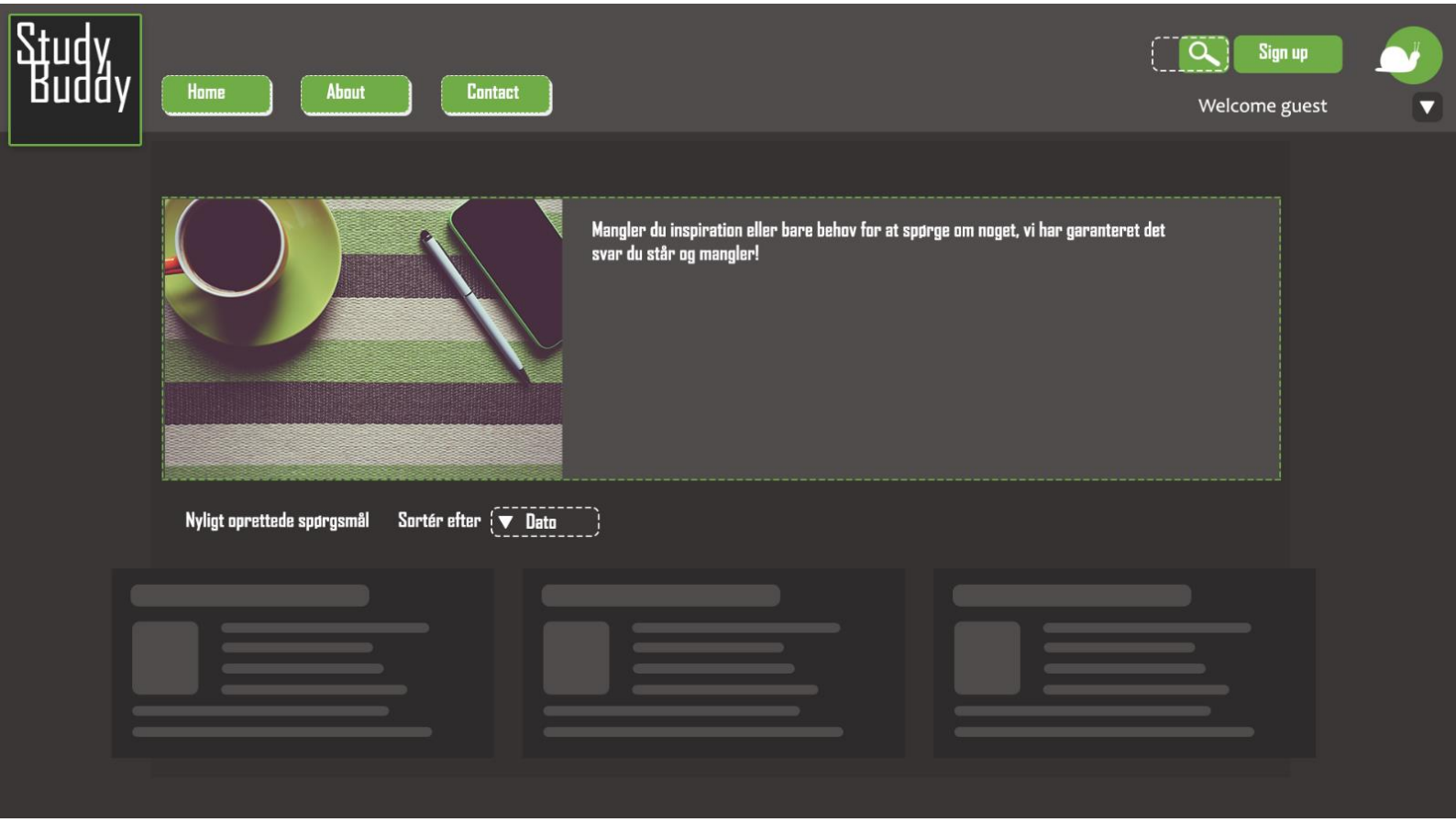
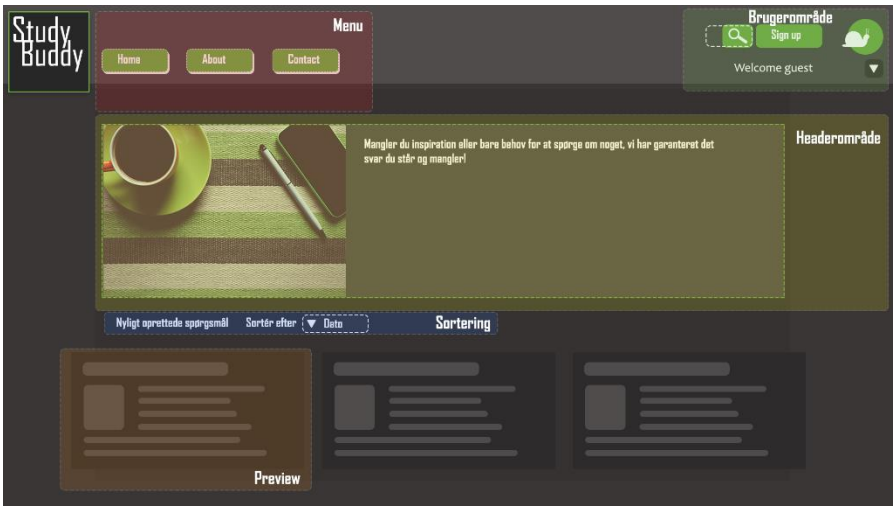
REACT HOOKS (m.m.)





Skrevet af: Kristian Hornbøll

Vejleder: Henrik Høltzer

Skole: Zealand Erhvervsakademi

Uddannelse: Datamatiker – 4. semester

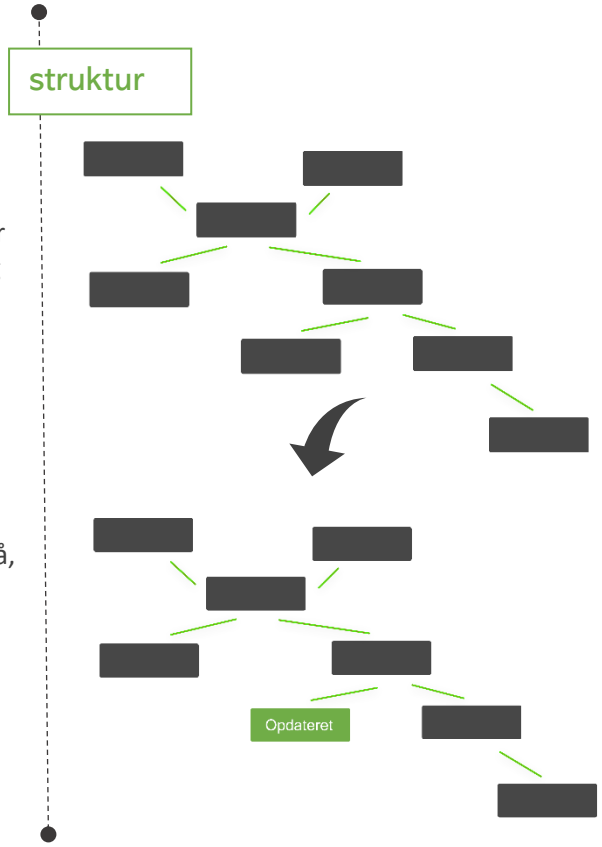
Afleveret: 29. maj, 2020

Antal tegn: 17259

Indhold

Introduktion

React er et frontend bibliotek som er udviklet af Facebook.
Det følger en retning indenfor webudvikling[1] som jeg finder enormt spændende.



React bruger intuitiv syntaks og en nem måde at opbygge struktur med komponenter der, hver især, kan indeholde sin egen logik og brugergrænseflade og ikke nødvendigvis afhænger af andre.
De forskellige komponenter kan afkobles fra hinanden, bruges og genanvendes til at sammensætte hele din app.

Efter at have haft undervisning i React vil jeg gerne gå mere i dybden og lære mere om hvordan React kan anvendes.

Der er selvfølgelig rigtig mange forskellige ting jeg kan fokusere på, bland andre:

- React Hooks
- State håndtering - Context eller Redux
- Higher Order Components
- I/O med Forms
- Autentificering & Autorisation
- Protected Routes & React Router Hooks
- MERN
- Test med Jest
- GraphQL API

Jeg vælger at fokusere på React Hooks som er relativt nyt indenfor React og herunder komme ind på Context, derefter vil kigge på Higher Order Components og til sidst, noget om brugerautentificering og adgangskontrol med React Router.

[1] SPA, PWA, Crossplatform, MERN

Problemstilling & formulering

Den Retning som webudvikling lige nu har og fremgangen af single page applikationer, progressive web apps og crossplatform, er frameworks, som er opsat til at udvikle dette koncept yderst populære for både arbejdsgiveren såvel som udvikleren.

Derfor vil jeg gerne undersøge mere om React og prøve at forklare de ting jeg finder interessante samt hvordan de kan anvendes i den proces, det er at udvikle en frontend app.

1. Hvad er React Hooks?
2. Hvad kan Higher Order Components bruges til?
3. State håndtering – Redux eller Context API?
4. Hvordan kan man autentificere en bruger i React?
5. Hvordan kan det hele så implementeres?

Metode

Min tilgang vil være at undersøge, afprøve og implementere.

Som udgangspunkt vil jeg undersøge de forskellige dele af React som jeg ikke er så bekendt med, læse dokumentation fra den officielle React hjemmeside[2] og gennemgå guides fra Udemy, Youtube og andre online tutorials.

Med den viden vil jeg afprøve de forskellige funktioner og finde ud hvordan de fungerer ved et implementere noget praktisk og prøve at sætte det hele sammen.

Jeg har en idé jeg har gået og tænkt over, hvor der er rig mulighed for at få afprøvet forskellige dele af de ting jeg finder ud af. Dette system vil danne basen for den praktiske del af mit arbejde.

[2] React

Plan

	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag
Uge 1	**Undersøgelse af forskellige emner**						
Uge 2							
Uge 3	**Afprøve og finde ud af hvordan de forskellige dele fungerer**						
Uge 4	**Her vil jeg samle alt det jeg har lært og prøve at oprette noget praktisk**						
Uge 5	**Synopsis skrivning – her vil jeg arbejde med selve synopsis og få styr på noter, hvad der skal med, samt få opstillet det hele.**				**Aflevering 29 / 5 11:00**		

Lad os komme i gang

Jeg har fundet et kursus på Udemy[3] som, sammen med den officielle dokumentation, vil være min grundlæggende "go to" for informationer. Derudover er der tit noget jeg bliver nødt til at slå op, lede på nettet efter forklaring på, her er Stackoverflow[4] brugt meget, min idé er at jeg vil prøve at forklare noget af de mere avanceret dele som jeg støder på.

React Hooks

Før React Hooks talte man om stateful (Class) som en opbevarende komponent og stateless (Functional) komponenten som den præsenterende.[5]

Med udgivelsen af React Hooks kan man håndtere state og andre React features i en funktionsbaseret komponent og samme effekt kan fås uden at opdele i funktioner og klasser. State beskriver den data der holdes i hukommelsen når din app kører, man kan sige det er en apps tilstand, om det er listen af venner som hentes ind når den starter op eller et simpelt vis eller ikke vis element. Der er derudover mulighed for at håndtere diverse sideeffekter på specificerede tidspunkter, om der er behov for genindlæsning af en komponent hver gang eller kun den ene gang.

Der er mange forskellige typer af hooks men fælles for dem alle er at de giver en mere lineær opbygning med en mere intuitiv metode til at håndtere alt hvad der foregår i en app.

[3] React for the Rest of Us
[4] Stackoverflow
[5] Presentational and Container Components

Det er vigtigt at huske på at state, medmindre det er opsat til at være det, ikke er persistent - det vil sige ting som ændres i løbet af en gennemgang af en app, vil være tilbage til udgangspunktet når appen lukkes og startes igen.

UseState

Til selve delen om state håndtering bruges useState, det er en indbygget funktion som returnerer 2 dele, selve værdien, som i princippet kan være hvad som helst: en boolean, en string, et objekt, et array osv. samt en funktion der kan opdatere denne værdi.

Når vi kalder useState kan vi trække de to ting ud ved hjælp af "destructuring", det vil sige vi trækker de to dele (værdi og funktion) ud og omstrukturere dem i hver deres variabel. Her er et eksempel med en Modal som initialiseres til false (den vises ikke).

```
const [modal, toggleModal] = useState(false)
```

Og toggleModal bruges til at opdatere denne værdi.

```
toggleModal (!modal)
```

Har du flere dele i en komponent kan der bruges flere useState funktioner, en til hver del.

UseEffect

UseEffect giver dig mulighed for at udføre forskellige ting i din app som kræver at alt andet er på plads, det er for eksempel svært for en app at loade data ind i noget brugergrænseflade som endnu ikke eksistere, det kan også være en simpel timer eller opretning af en logfil. UseEffect sørger for at din app er indlæst før den udføres.

Som argumenter forventes at modtage, først en callback funktion der fortæller hvad der skal udføres, derefter et array som specificere hvornår og hvor tit den skal udføres.

```
useEffect(() => {
  // Fetch API
}, [])
```

Et array, som kan være tomt, vil kun udføre funktionen første gang en app indlæses. Det kan også indeholde forskellige variabler som, hvis ændret, vil udføre funktionen.

Hvis du helt undlader at give useEffect det andet argument vil funktionen udføres hver eneste gang din komponent bliver genindlæst.

useRef

UseRef kan bruges til specifikt at referere til brugergrænsefladeelementer i en app, det kan være enormt brugbart hvis du for eksempel vil lave noget dynamisk med et inputfelt som en bruger kan skrive i, eventuelt fokusere det når siden er genindlæses, eller for eksempel ved et klik på en ”redigér” knap. Dette kan være med til at skabe en bedre brugeroplevelse.

```
const inputRef = useRef(null)
useEffect(() => {
  inputRef.current.focus()
}, [])
```

I html (jsx) skal vi så have en ref-property med referencen.

```
<input ref={inputRef} type="text" />
```

Dette kan lade sig gøre fordi useRef variablen ikke følger den normale React opdaterings cyklus, men opfører sig mere som instansvariablen som kendes fra en klasse.

I stedet for, på React manér at arbejde med JSX, så springer useRef det led over og går direkte til kilden, det aktuelle DOM element <input .. /> feltet. Derfor er det vigtigt at passe på hvordan man bruger useRef.

UseReducer

UseReducer er et alternativ til useState og anvendes typisk hvis du arbejder med lidt mere kompleks state og Context API'en. I modsætning til useState, som returnerer state + en funktion der kan opdatere denne state, så returnerer useReducer state og en dispatch funktion.

```
const [state, dispatch] = useReducer(reducer, initialState)
```

Når useReducer kaldes forventer den at modtage to argumenter, det første er en reducer-funktion og den anden er den indledende state.

Reducer funktionen er så der hvor du afgør hvilken action du skal udføre samt hvad der skal ske.

```
const reducer = (state, action) => {
  switch (action.type) {
    case 'login':
      return { ...state, isAuthed: true }
    case 'logout':
      return { ...state, isAuthed: false }
  }
}
```

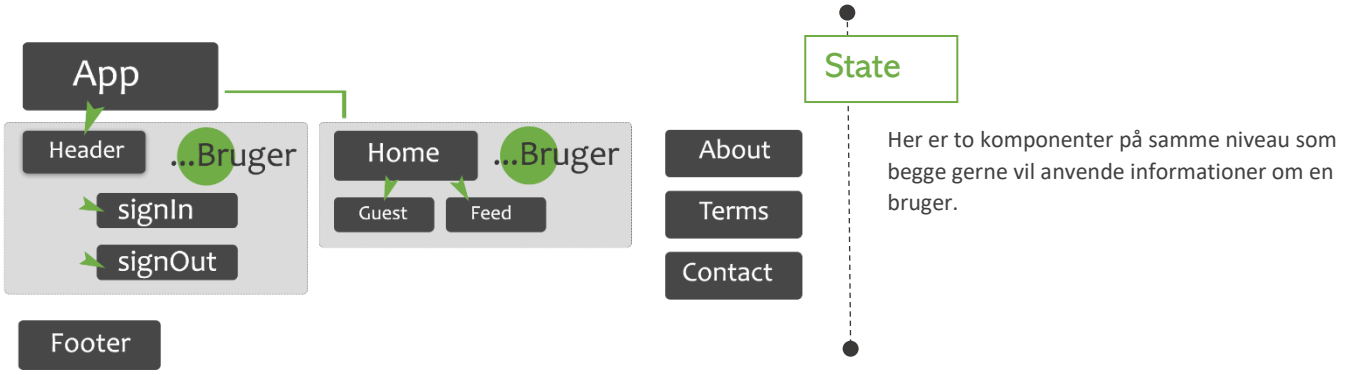
Den indledende state kan indeholde al den state din komponent har behov for.

Dispatch funktionen bruges til at udføre en action – dispatch er i sig selv bare en måde at fortælle hvad man gerne vil udføre men ikke hvordan. Det er så reducerens opgave at håndtere den logik der har med selve implementeringen at gøre.

Idéen er at du så, meget enkelt kan videregive dispatch funktionen til de komponenter hvor der er brug for at udføre en action. Det er så her Context kommer ind i billedet.

Redux eller Context

Når vi arbejder med state i en app, kan vi løbe ind i problemer hvor noget information er blevet oprettet dybt inde i en app, lad og se på eksemplet med en bruger.

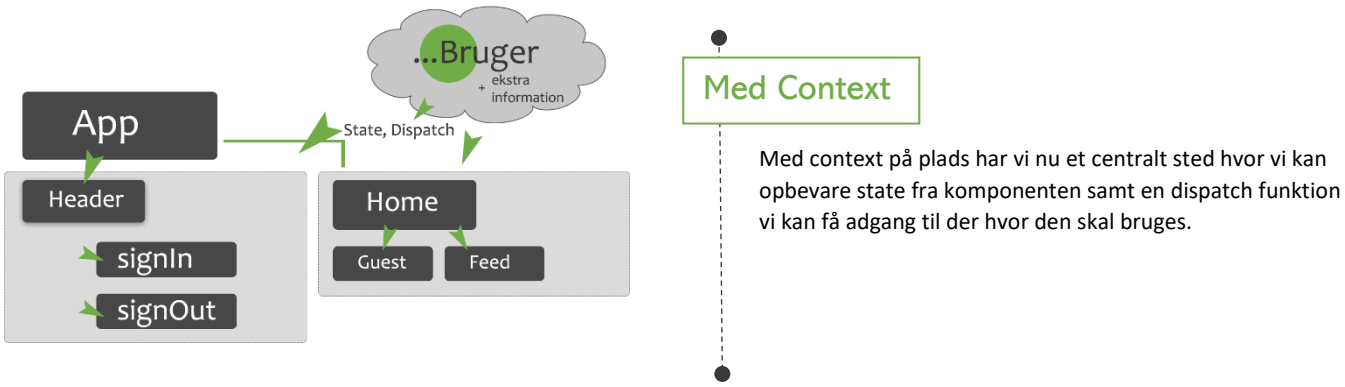


Der er adskillige måder at håndtere state i din app, den mest simple måde er det man kalder 'Lifting State Up'. Det vil sige at state initialiseres i en højere komponent som er fælles for de underliggende komponenter der har behov for at anvende den og på den måde kan state sendes nedad til de respektive komponenter (Også kaldet prop-drilling). Bliver det lidt mere omfangsrigt kan man anvende et bibliotek som for eksempel Redux[6], det er en slags boks som kan opbevare og levere state der hvor der er behov for det.

Med hensyn til state håndtering er Context blevet bedre med tiden, og siden React version 10.3.4 og Hooks blev introduceret er det blevet, ikke bare lettere at anvende, men også hurtigere at få op og køre til både små og større projekter. Det gør mange af de samme ting som Redux, bare på en lidt anden måde.

Context

Med useReducer Hook funktionen på plads er der nu mulighed for at anvende den state der hvor den skal bruges.



[6] Redux er et tredjeparts bibliotek med funktionalitet der hjælper med at håndtere state i din app

Context API'en fungerer ved at der oprettes en fælles Provider med en værdi (state, funktioner) som alle komponenter, med behov for det, har adgang til.

Det kan være isAuthed state, som initialiseres til false, i vores log-ind komponent kan vi nu hente dispatch context og udføre dispatch({type: 'login'}) ved et succesfuldt login.

```
dispatch({ type: 'login' })
```

Så ændres isAuthed til true, og de komponenter som arbejder med at vise noget på baggrund af denne del af state, vil så genindlæses og vise brugerens personlige sider.

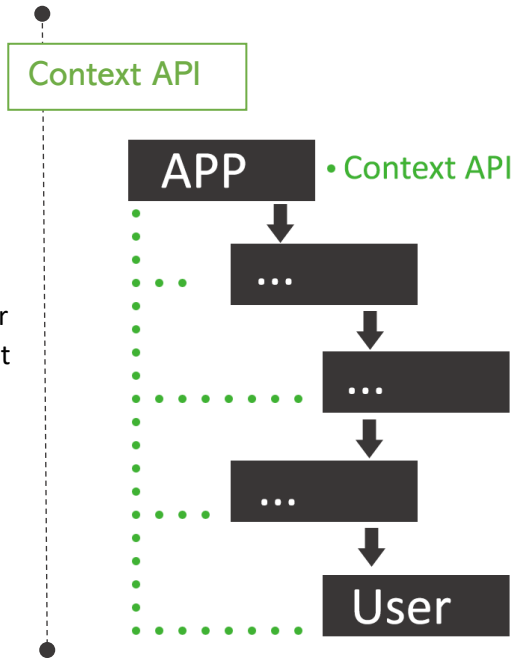
Idéen minder utroligt meget om Redux – det er der selvfølgelig en grund til, eftersom Redux anvender den bagvedliggende logik fra React Context til at opnå den 'magiske' effekt af kontakt mellem kontaktløse komponenter.

Hvorfor Context over Redux

Context API'en fungerer rigtig godt, opsætningen er ligetil og det er uden brug af tredjeparts tilføjelser. Sammen med useReducer og createContext er den i en sådan stand at det sagtens kan anvendes til langt de fleste ting.

Med hensyn til ydelse kan der være lidt problemer[7], men det er først når du kommer op i meget større applikationer hvor der arbejdes med noget langt mere kompliceret state, for eksempel caching af store mængder data, logning osv.

Alt fra bruger tilstande, temaer, UI-tilstande til farver eller lokalisering håndterer Context API rigtig godt.



[7] React Context API vs Redux

Det er vigtigt at huske på, at hvis du senere hen, selv om du ikke regnede med det, har en komponent som skal kunne håndtere noget state hvor du absolut har behov for mere performance, så kan man sagtens benytte Context API'en og Redux side om side.

Higher order Components

En Higher Order Component er i bund og grund bare en komponent der som argument tager en anden komponent som så returneres med ekstra funktionalitet.

```
const UpdatedComponent = OriginalComponent => {
  class NewComponent extends Component {
    // Ekstra funktionalitet
    render() {
      return <OriginalComponent {...funktion} />
    }
  }
  return NewComponent
}
export default UpdatedComponent
```

Det er en måde at abstrahere fra den bagvedliggende logik og genbruge funktionalitet i din app, uden behov for at duplikere noget kode.

Eksempler på en HOC er withRouter fra react-router, som importeres og vikles rundt om din export og det giver den adgang til router information som blandt andet history, der kan anvendes til at dirigere en bruger til en anden side.

Modal

Jeg fik den idé at jeg ville prøve at implementere min egen HOC med det traditionelle pattern af en HOC, det viste sig dog ikke at være så let.

Desuden jeg fandt så ud af senere hen at der selvfølgelig er et React bibliotek med selv samme funktion++.

Mit resultat, om end ikke en "rigtig" Hoc, så har den lidt af samme funktionalitet. Den kan importeres som enhver anden komponent i din app og vikles rundt om et element i din app, det kan være en tekst boks eller et billede for eksempel og give den mulighed for at blive vist som en modal i fokus i midten af din brugergrænseflade.

```
<Modal>
  <img className="image" src={image} alt="image"/>
</Modal>
```

Det er mulighed for videreudvikling af denne idé, med eventuel størrelse eller en titel osv., det kunne gives videre til komponenten via props.

Adgangsbegrænsning med React Router

I React bruges logiske og betinget udtryk rigtig meget, i det her tilfælde til at afgøre hvad der skal vises til brugeren. Her afhænger det af om en bruger er autentificeret eller ej.

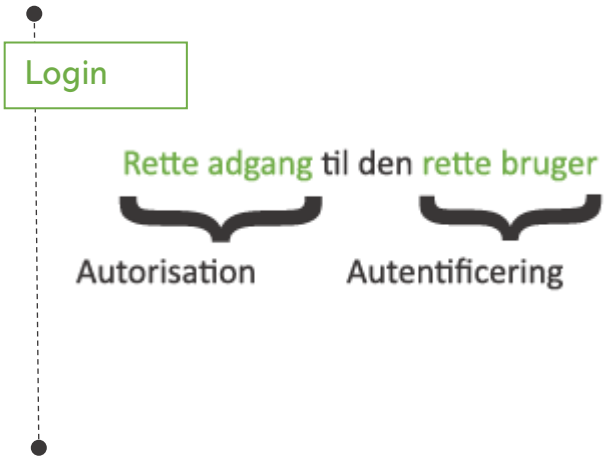
```
const userInfo = isAuthed ? <HeaderLoggedIn /> : <HeaderLoggedOut />
```

Samme fremgangsmåde kan bruges når det gælder visningen af en personlig side (hvis du er logget ind) eller en generel side (hvis du ikke er logget ind)

```
<Route path="/Home">
  {
    isAuthed ? <PersonligSide /> : <GenerelSide />
  }
</Route>
```

Autentificering & Autorisation

Ethvert it-system i dag, hvad enten vi taler om en web app på nettet eller en native app på en computer eller smartphone, vil formentligt have en eller anden form for login system der kan håndtere flere brugere som har hver deres eget personlige indhold.



Der er adskillige måder at håndtere flere brugere i et system, måden jeg vil bruge anvender et JWT[8] adgangs token til at afgøre hvem du er, det token vil blive oprettet og sendt over til dig når du logger ind, det gemmes lokalt på din enhed i localStorage og kan derefter sendes med til serveren når der anmodes om for eksempel at oprette en ny post.

I React kan vi oprette en isAuthed state, som vi manipulere når en bruger bliver godkendt (han logger på eller af), på baggrund af den state kan vi så, som vist tidligere, bestemme hvor han skal dirigeres hen og hvad der skal vises.

På samme måde kan vi sørge for at brugeren forbliver logget ind selvom der skiftes mellem forskellige sektioner ved at tjekke om token eksisterer.

Refleksion

Til at starte med havde jeg en vag idé om hvad jeg havde tænkt mig, når jeg nu ser tilbage på hele forløbet, så er der rigtig mange ting der kunne / burde have været anderledes.

Mit emne (delemne) nåede at ændre sig en del, selv i løbet af perioden. Der opstår rigtig mange spørgsmål og der kommer flere ting med som jeg kunne vælge at skrive om, det ender med at jeg bliver overvældet over hele situationen.

Det skal selvfølgelig også tages i mente hele den situation med pandemien og at vi alle sammen arbejder hjemmefra, meget af mit fokus lå i starten der.

Oven i det, kommer så presset om at finde praktikplads – så det ender med at jeg stort set ikke får lavet noget de første par uger.

Når jeg så småt kommer i gang igen, er der ikke så meget tid tilbage, og jeg har stadig ikke fuldstændig fået fastlagt et emne, Henrik har dog været meget behjælpelig med det hele og prøvede at styre mig ind på det rette spor.

Jeg burde have sørget for at mit emne var fastlagt fra starten af.

På baggrund af alt dette synes jeg dog at det egentlig lykkedes meget godt at få noget brugbart ud af perioden, jeg har i hvert fald fundet ud af at der er rigtig mange spændende ting at undersøge indenfor generel webudvikling og React.

Konklusion

1. Hvad er React Hooks?

   React Hooks er funktioner som udfører generelle React features og overgangen fra klassebaseret komponenter til funktioner er med til at streamline React biblioteket, man kan sige der bliver mere fokus på hvad i stedet for hvordan. Selvom det ikke er alle originale funktioner der er kommet med over endnu, er langt de fleste kommet med. Det er heller ikke alle jeg har været inde over i denne opgave.

[8] JSON Web Token

2. Hvad kan Higher Order Components bruges til?

Alt hvad du kan forestille dig – flere ting som fx temaer (dark, light fx), lokalisering, enkeltstående komponenter som har noget til fælles kan alle have glæde af at få sin funktionalitet fra et centralt sted.

3. State håndtering – Redux eller Context API?

Det kommer an på hvor stort et projekt du arbejder med og hvilke tilstande der arbejdes med, Context API er meget lettere at opsætte og komme i gang med, men kræver lidt ekstra arbejde i forhold til håndtering af kompleks state, hvor Redux er lidt mere besværlig med hensyn til opsætning og brug men har nogle indbyggede funktioner (bl.a. connect som er en HOC) som hjælper med håndtering af kompleks state.

4. Hvordan kan man autentificere en bruger i React?

Typisk vil man oprette noget adgangskontrol og afgøre om en bruger er autentificeret og på baggrund af dette hente data og vise den del der er personlig til denne.

5. Hvordan kan det hele så implementeres?

Planen er at vise hvordan de fleste ting fungere ved at udvikle en funktionel app hvor en bruger kan logge ind og oprette spørgsmål, som andre brugere kan svarer på.

Litteraturliste

React | Facebook, React - https://reactjs.org/ | Besøgt I hele perioden fra d. 27. april – 29. maj.

- Officiel hjemmeside for React

React for the rest of us | Brad Schiff - https://www.udemy.com/course/react-for-the-rest-of-us/ | besøgt i hele perioden fra d. 27. april – 29. maj.

- Udemy kursus om React

PWA vs. Native | Mike Harris - https://www.youtube.com/watch?v=vhg01Ml-8pI | Besøgt 16-05-2020 16:57

- En kort gennemgang af hvad PWA er og hvordan det fungerer.

Presentational and Container Components | Dan Abramov - https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0 | besøgt 24-05-2020

- En artikel der beskriver de forskellige måder at oprette en komponent

Stackoverflow | https://stackoverflow.com/questions/53314857/how-to-focus-something-on-next-render-with-react-hooks | besøgt 14-05-2020

- Stackoverflow-svar som viser hvordan useRef og useEffect kan bruges til at toggle et tekstfelt

Redux | https://redux.js.org/

- Officiel dokumentation for Redux

Refactoring Higher-Order-Components to React Hooks | Roy Derks - https://dev.to/gethackteam/from-higher-order-components-hoc-to-react-hooks-2bm9 | besøgt 19-05-2020

- En artikel der beskriver hvordan du kan oprette dine HOC som funktionelle komponenter

The Hooks of React Router | Agney Menon - https://css-tricks.com/the-hooks-of-react-router/ | besøgt 16-05-2020

- Dybdegående artikel omkring React Router Hooks

JWT | https://jwt.io/

- Officiel hjemmeside for JWT

React Context API vs. Redux | Dave Ceddia - https://www.youtube.com/watch?v=giSbnDTJ8Ao | besøgt 21-02-2020

- Forklaring af forskellene mellem Redux og Context API

Stackoverflow | https://stackoverflow.com/

- Spørgsmål og svar

Billede på forsiden af kaffekop og mobiltelefon, (i header på hjemmeside-mock) | fra https://www.pexels.com/da-dk/ | Besøgt 17-05-2020

Grafik og illustrationer er hjemmelavede