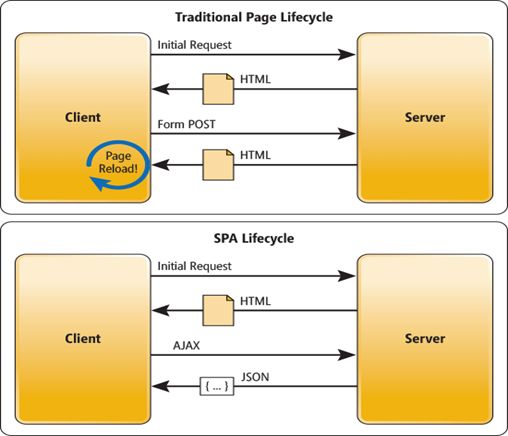
**React 1:**

General part

* Describe the term Single Page Application and why it has become so popular for modern web-applications

SPA skifter indholdet ud i dele af websitet. Det gør at, man ikke skal lave for mange page refresh fordi den bruger AJAX kald til at udskifte det med og rendere for det meste på klientsiden.

Alle browsere supportere Javascript (SPA). Hvis du vil lave et website og give brugeren en god brugeroplevelse så er Javascript et rigtigt godt bud på det. Der er mange forskellige typer af frameworks til Javascript som fx. React og Angular. Det gør at man kan flytte det mere over på klienten end på serveren.



* Explain the “recommended” React way of passing data into Components (at the top, or at the bottom or..)

import top. ??

* Explain how JavaScript array methods, like filter, map and (reduce) can be used to generate dynamic HTML structures (tables, ul's etc.)

Det er en bedre måde at skrive et Forloop på, kan man sige. fx. map funktionen gør at man kan løbe et array igennem og skrive JSX tags med outputs imellem hvert tag. Ofte har vi brug for at modificere hvert element på den samme måde. En map funktion acceptere et callback som det første argument. Når du kalder en map funktion på et array, den eksekvere den callback på hvert element inden i det. Det returnere et nyt array med all de værdier som callbacken returnere.

En map funktion tager 3 argumenter til callbacken:

1. Det nuværende item i array’et
2. Array’ets index på det nuværende item
3. Hele callbacken på det array

**const numbers = [1, 2, 3, 4, 5];  
const listItems = numbers.map((number) =>  
 <li>{number}</li>  
);**

filter() metoden laver et nyt array med alle elementer som “består” den sammenligning som er blevet implementeret i den givet funktion.

**family.filter(person => person.age > 18);**

* Explain about the Observer pattern, and where you have used it, both with Java and JavaScript.

HVAD??? HAR VI BRUGT DET??? HVOR??? THREADS?? MEN HVOR I JS??

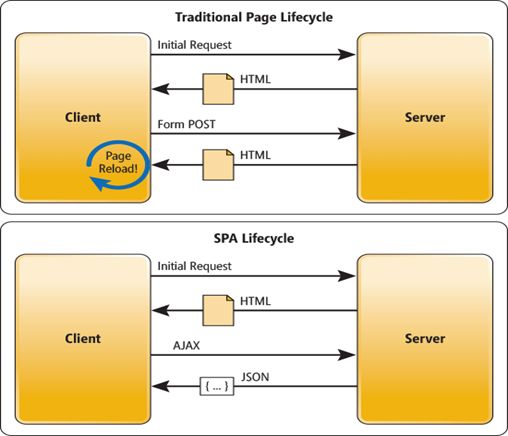
**React 2:**

General part

* Describe the term Single Page Application and why it relevant for modern web-applications

SPA skifter indholdet ud i dele af websitet. Det gør at, man ikke skal lave for mange page refresh fordi den bruger AJAX kald til at udskifte det med og rendere for det meste på klientsiden.

Alle browsere supportere Javascript (SPA). Hvis du vil lave et website og give brugeren en god brugeroplevelse så er Javascript et rigtigt godt bud på det. Der er mange forskellige typer af frameworks til Javascript som fx. React og Angular. Det gør at man kan flytte det mere over på klienten end på serveren.



* Describe how SPA's are implemented with React

Det virker på den måde at <App /> komponenten som standard, som indeholder hele SPA’en, bliver renderet i <div id=”root”> og outputter hele websitet. Det er overordnet set en masse små komponenter som bliver sat sammen til en stor komponent til sidst.

* Explain about Routing in React

Routing i React er den måde man kan navigere rundt på forskellige komponenter, så at sige forskellige sider.

<Switch>

<Route exact path='/' component={CarStore} />

<Route exact path='/about' component={About} />

<Route exact path='/cars/add' component={AddCars} />

<Route exact path='/cars/edit/:id' component={EditCars} />

<Route exact path='/cars/:id' component={CarStoreDetails} />

</Switch>

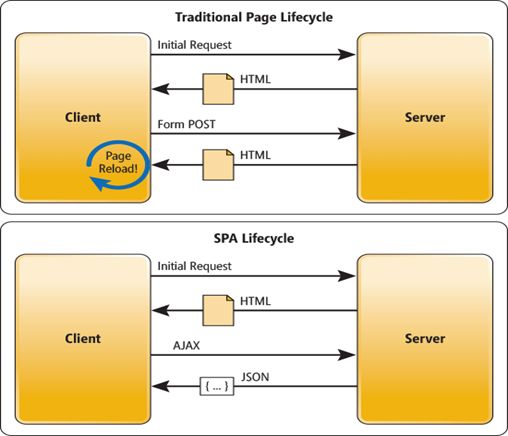
**React 3:**

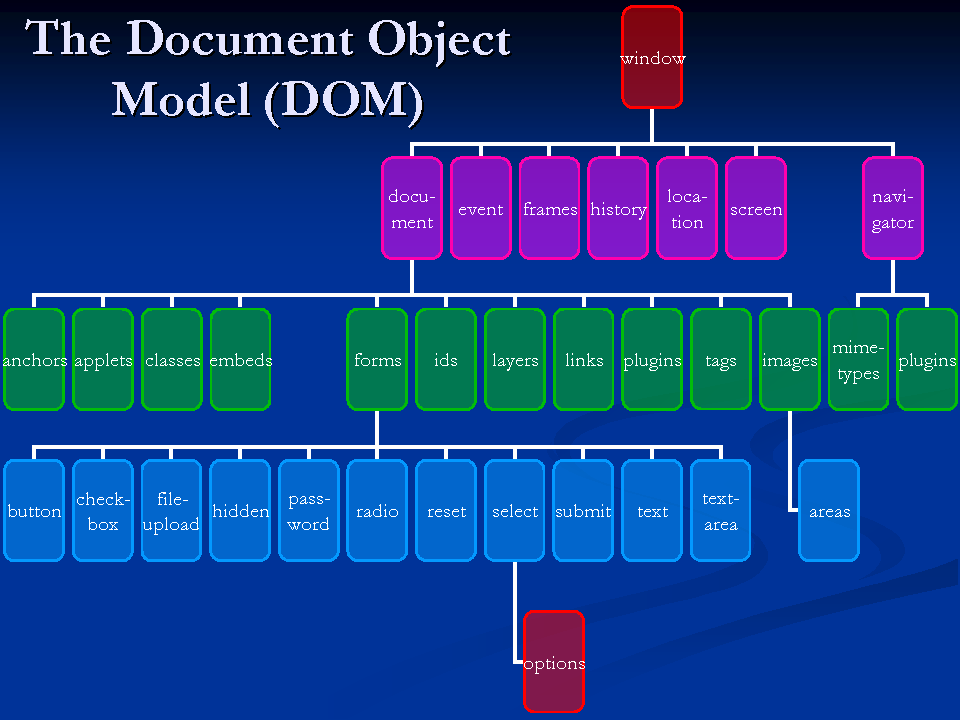
General part

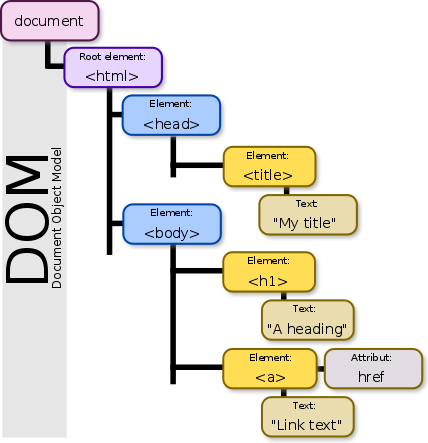
* Describe the term Single Page Application and why it relevant for modern web-applications

SPA skifter indholdet ud i dele af websitet. Det gør at, man ikke skal lave for mange page refresh fordi den bruger AJAX kald til at udskifte det med og rendere for det meste på klientsiden.

Alle browsere supportere Javascript (SPA). Hvis du vil lave et website og give brugeren en god brugeroplevelse så er Javascript et rigtigt godt bud på det. Der er mange forskellige typer af frameworks til Javascript som fx. React og Angular. Det gør at man kan flytte det mere over på klienten end på serveren.







* Describe fundamental topics in React, like the Virtual Dom, JSX, Rendering, Components, Props, State, Life Cycle etc.

I React for hver DOM objekt, er der et tilsvarende virtuelt DOM objekt. Et virtuelt DOM objekt repræsentere en form for en letvægts kopi. At manipulere DOM’en er langsomt. Manipulere den virtuelle DOM er meget hurtigere fordi det ikke sker direkte på skærmen. Tænk det som at manipulere den virtuelle DOM som at, redigere i en tegning, i stedet for at flytte rundt på ting i et hus.

JSX er en form af det ligner XML/HTML syntax som bliver brugt i React der udvider ECMAScript, det samarbejder sammen med Javascript/React kode. Syntaxen er beregnet til at blive brugt af præprocessorer såsom Babel som transformere HTML-lignende tekst findes i JavaScript-filer i standard JavaScript objekter, som en JavaScript-motor vil fortolke.

**Dybest set, ved at bruge JSX kan du skrive kortfattet HTML / XML-lignende strukturer (fx DOM ligesom træstrukturer) i den samme fil som du skriver JavaScript-kode, så Babel vil omdanne disse udtryk i selve JavaScript kode. I modsætning til fortiden, i stedet for at sætte JavaScript i HTML, tillader JSX os at sætte HTML ind i JavaScript.**

rendering

Components giver en mulighed for at splitte User interfacet op i uafhængige dele. Genbruge dele og at kunne tænke at hver lille del er et stykke isolerede stykke kode. Overordnet set er componenter ligesom Javascript funktioner. Componenter accepterer vilkårlig input (kaldet props) og returnerer React-elementer, der beskriver hvad der skal vises på skærmen.

Props er uforanderlige. Props og state er relateret til hinanden. Tilstanden af ​​en component bliver ofte props til et child component. Props sendes til child’en inden for hoved renderingsmetoden som det andet argument til React.createElement () eller, hvis du bruger JSX, de mere kendte tagattributter.

**<MyChild name = {this.state.childsName} />**

Use **state** to store the data your current page needs in your controller-view.

Use **props** to pass data & event handlers down to your child components.

These lists should help guide you when working with data in your components.

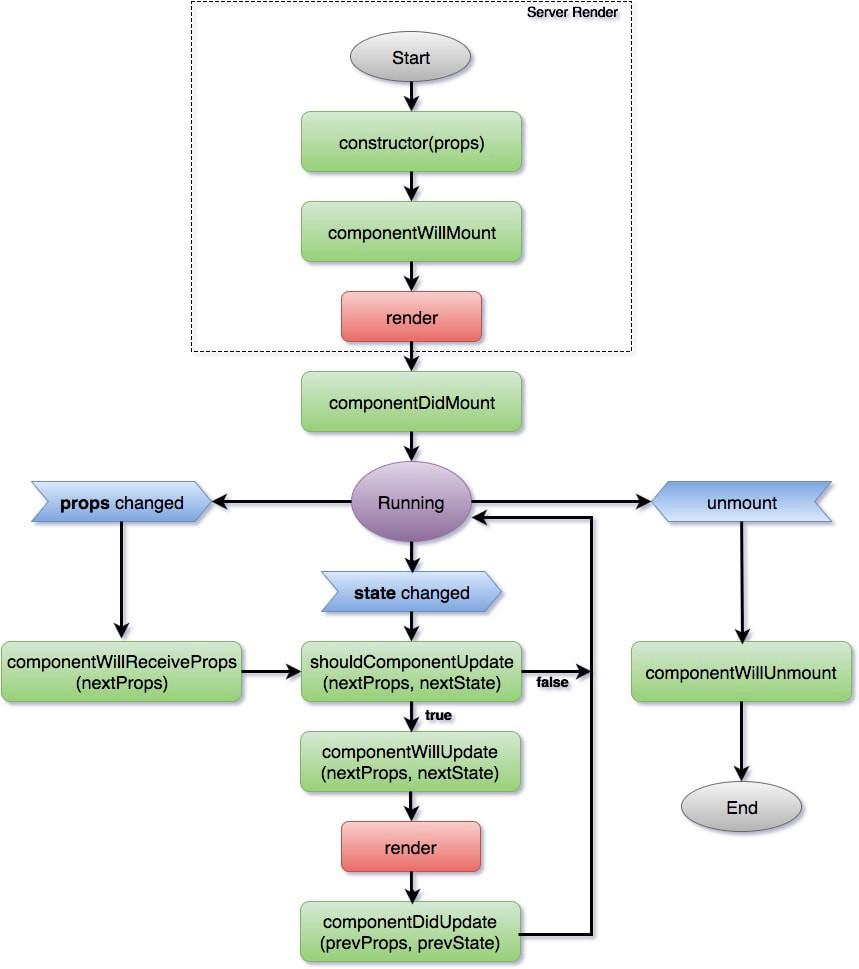
**Props**

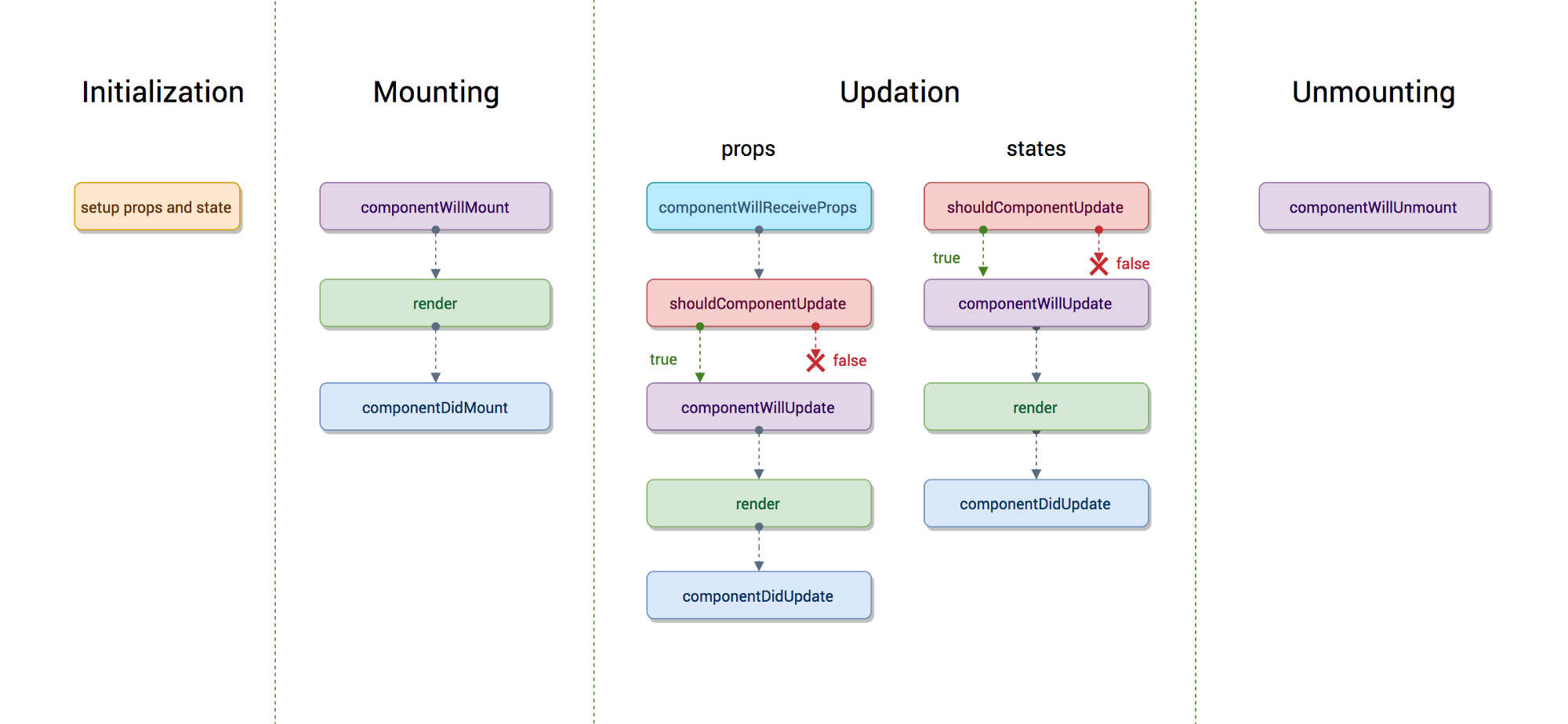
* are immutable
  + which lets React do fast reference checks
* are used to pass data down from your view-controller
  + your top level component
* have better performance
  + use this to pass data to child components

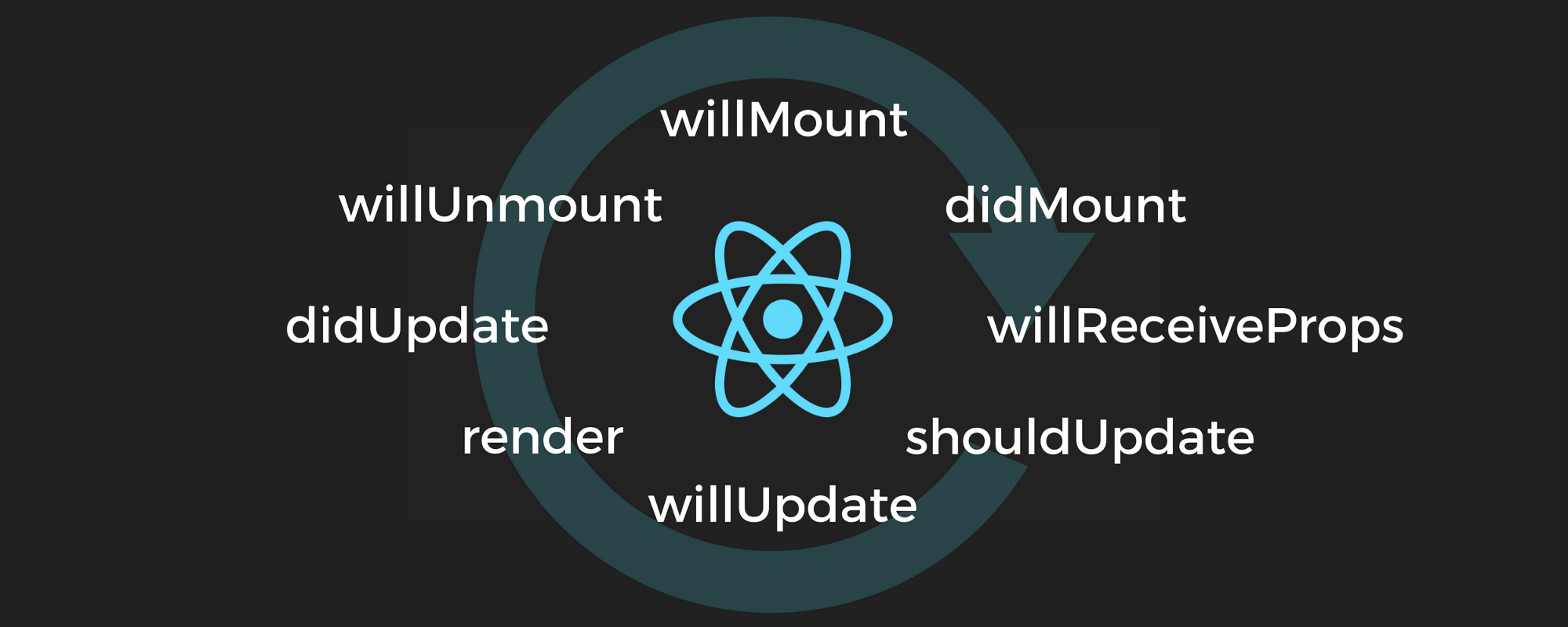
**State**

* should be managed in your view-controller
  + your top level component
* is mutable
* has worse performance
* should not be accessed from child components
  + pass it down with props instead

Der er flere typer af life cycles / stadier men dem vi er blevet præsenteret for og har arbejdet med er componentWillMount og componentDidMount. ComponentWillMount afviger ikke meget fra konstruktøren - det kaldes også en gang kun i den indledende mouting lifecycle. Vi har brugt det til at fetche fra et API og da det kører asynkront så har vi brug for at bruge componentWillMount og sætte state når vi har data klar fra vores fetch.







* Explain about Components and Routing in React

Components giver en mulighed for at splitte User interfacet op i uafhængige dele. Genbruge dele og at kunne tænke at hver lille del er et stykke isolerede stykke kode. Overordnet set er componenter ligesom Javascript funktioner. Componenter accepterer vilkårlig input (kaldet props) og returnerer React-elementer, der beskriver hvad der skal vises på skærmen. Componenter har altid en render metode.

Routing i React er den måde man kan navigere rundt på forskellige komponenter, så at sige forskellige sider.

<Switch>

<Route exact path='/' component={CarStore} />

<Route exact path='/about' component={About} />

<Route exact path='/cars/add' component={AddCars} />

<Route exact path='/cars/edit/:id' component={EditCars} />

<Route exact path='/cars/:id' component={CarStoreDetails} />

</Switch>

* Explain the concept CORS, and why it is (often) relevant for Single Page Applications

Cross-Origin Resource Sharing (CORS) er en mekanisme, der bruger yderligere HTTP-overskrifter for at lade en brugeragent få tilladelse til at få adgang til udvalgte ressourcer fra en server i en anden oprindelse (domæne) end det websted, der aktuelt er i brug. En brugeragent laver en HTTP-anmodning med tværgående oprindelse, når den anmoder om en ressource fra et andet domæne, en protokol eller en port end den, hvorfra det nuværende dokument stammer.

Et eksempel på en anmodning om krydsoprindelse: En HTML-side, der serveres fra http://domain-a.com, gør en <img> src-anmodning til http://domæne-b.com/image.jpg. Mange sider på internettet indlæser i dag ressourcer som CSS stylesheets, billeder og scripts fra separate domæner, såsom content delivery networks (CDNs).

Af sikkerhedsmæssige grunde begrænser browsere HTTP-anmodninger fra HTTP, der er initieret fra scripts. For eksempel følger XMLHttpRequest og Hent API'en den samme oprindelse politik. Dette betyder, at en webapplikation, der bruger disse API'er, kun kan anmode om HTTP-ressourcer fra det samme domæne, hvor applikationen blev indlæst, medmindre CORS-overskrifter anvendes.

Hvis du prøver at bruge AJAX uden dit domæne, skal du sandsynligvis få denne fejlmeddelelse:

XMLHttpRequest kan ikke indlæse http://www.domain.com/path/filename. Oprindelse null er ikke tilladt af Access-Control-Allow-Origin.

Grunden til, at du får denne fejlmeddelelse, er på grund af politikken med samme oprindelse. Politikken tillader scripts, der kører på sider, der stammer fra det samme websted, for at få adgang til hinandens data uden specifikke begrænsninger, men forhindrer script adgang til data, der er gemt på et andet domæne.

Dette kan være et problem, hvis du forsøger at få adgang til offentligt hostede data, men der er måder rundt om det.

Her er listen over metoder:

***Implementere CORS (Cross-Origin Resource Sharing) <--- Som vi har brugt***

Brug JSONP (JSON Padding)

Brug postMessage metode

Konfiguration af en lokal proxy

CORS er en mekanisme, der gør det muligt at anmode om ressourcer på en webside fra et andet domæne uden for det domæne, hvorfra ressourcen stammer fra. Især kan JavaScript AJAX-opkald bruge XMLHttpRequest-mekanismen. Sådanne "cross-domain" -forespørgsler ville ellers forbydes af webbrowsere, i henhold til samme oprindelses sikkerhedspolitik. CORS definerer en måde, hvorpå browseren og serveren kan interagere for at afgøre, om anmodningen om krydsoprindelse skal tillades eller ikke. Det er mere nyttigt end kun at tillade anmodninger om samme oprindelse, men det er mere sikkert end blot at tillade alle sådanne anmodninger om krydsoprindelse.

CORS er en W3C-anbefaling og understøttes af alle større browsere. Den bruger HTTP-overskrifter til at hjælpe browseren med at afgøre, om en AJAX-anmodning på tværs af domæner er sikker. Når du foretager en CORS-forespørgsel, tilføjer browser grundlæggende Origin-overskrift med den aktuelle domæneværdi.

