

Sovelto

React

Agenda

- Intro, eli Hello React
- React komponentit
- Create React App
- Koosteiset komponentit
- Tilan hallinta ja lomakkeet
- REST palvelun hyödyntäminen
- Reititys - react-router

- Ja: harjoituksia

Intro

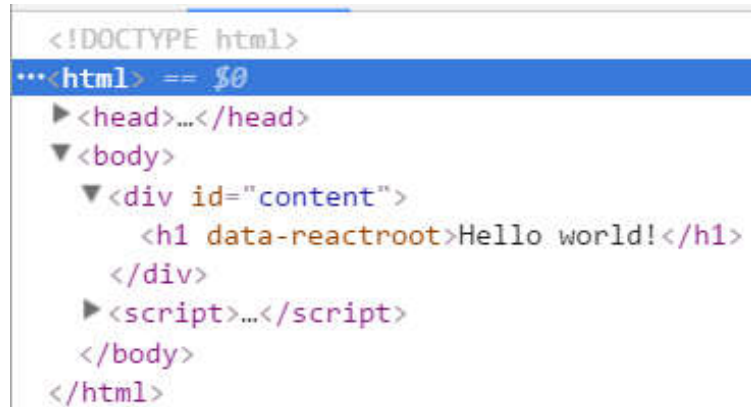
React

- Avoimen lähdekoodin kirjasto ja viitekehys, alunperin kehittäjinä Facebook ja Instagram
- Selitetään usein että toteuttaa V:n MVC mallissa
- Vain esityskerros toteutettu, muut teknologiakerrokset ovat Reactin ulkopuolella
- Komponenttiperusteinen, luodaan uudelleenkäytettäviä komponentteja näyttämään tietoa
- Yksisuuntainen datan sidonta, React näyttää (renderöi) muuttuneen tiedon automaagisesti
- Pyrkii olemaan tehokas ratkaisu massiivisiin SPA (Single Page Application) ratkaisuihin, sekä toteutuksen että ylläpidon kannalta

Hello React

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello React</title>
<meta charset="utf-8" />
  <script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>
  <script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>
</head>
<body>
  <div id="content"></div>
  <script>
    ReactDOM.render(
      React.DOM.h1(null, "Hello world!"),
      document.getElementById("content")
    );
  </script>
</body>
</html>
```

Hello world!

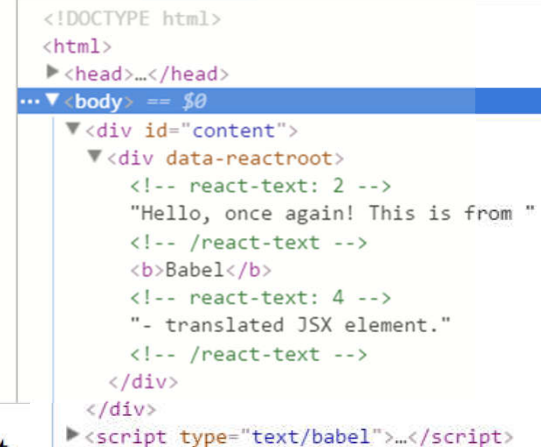


```
<!DOCTYPE html>
...<html> == $0
  <head>...</head>
  <body>
    <div id="content">
      <h1 data-reactroot>Hello world!</h1>
    </div>
    <script>...</script>
  </body>
</html>
```

Hello World - JSX käyttäen

```
<!DOCTYPE html>
<html>
<head><title>Hello React</title><meta charset="utf-8" />
  <script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>
  <script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>
<body>
  <div id="content"></div>
  <script type="text/babel">
    class MyComponent extends React.Component {
      render () {
        return (
          <div>
            Hello, once again! This is from <b>Babel</b>-
            translated JSX element.
          </div>);
        }
    };
    ReactDOM.render(<MyComponent/>,
      document.getElementById("content"))
  </script>
</body>
</html>
```

Hello, once again! This is from **Babel**- translated JSX element.



```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>...
    <div id="content">
      <div data-reactroot="
        <!-- react-text: 2 -->
        "Hello, once again! This is from "
        <!-- /react-text -->
        <b>Babel</b>
        <!-- react-text: 4 -->
        "- translated JSX element."
        <!-- /react-text -->
      </div>
    </div>
    <script type="text/babel">...</script>
```

JSX

- JavaScript-extensions, ottaa XML-piirteitä mukaan
- Käytä text/babel (ennen text/jsx) skriptin tyyppinä
 - `<script type="text/babel"> ...</script>`
- Selain ei ymmärrä JSX-koodia, täytyy muuntaa JavaScriptiksi
- Pienissä kokeiluissa voi tehdä muunnoksen selaimessa, tuotantokoodissa - ja nykyisin usein myös kehtiysvaiheessa - muunnos tehdään jo palvelimella
 - Esimerkiksi Babel kertoo tämän JavaScript konsolissa
- JSX ei ole pakollinen React-kehityksessä, mutta on käytössä oikeastaan aina

Create React App

Create React App

- Facebookin tapa alustaa React-projekti
 - <https://github.com/facebook/create-react-app>
- Käytössä npm (tai yarn) ja package.json projektin määrittelemiseen
 - npm start
 - npm run build
 - npm test
 - npm run eject
- Mukana oleva react-scripts käyttää mm. Babel ja Webpack kirjastoja. Webpackin builder mahdollistaa projektin paketoinnin staattisiin ja minifioituihin JavaScript tiedostoihin
- Webpackin dev server ja hotdeploy helpottavat nekin kehitystyötä
 - Ei tarvitse kuin tallettaa tiedosto, ja muutokset heti näkyvissä selaimessa
- ESLint ja Jest kuuluvat myös oletustyökaluihin
- VS Code: Reactjs code snippets
<https://marketplace.visualstudio.com/items?itemName=xabikos.ReactSnippets> helpottavat kehitystä kummasti

Harjoitus 2 - ensimmäinen projekti

- Avaa komentotulkki (command prompt), cd \kurssi tms.
- Asenna create-react-app globaaliksi
 - npm i create-react-app -g
- Luo nyt projekti komentoriviltä
 - create-react-app my-app
 - cd my-app
 - Voit avata projektin VS Codessa: File | Open - ja etsi projektin hakemisto
- Projekti on nyt valmis
 - Tutki package.json tiedostoa
 - Tutki mitä kaikkea muuta löydät projektista
 - Käynnistä projekti: npm start
 - Palvelimen sammutus: Ctrl-C
- Kuten FB suosittelee: käynnistä projekti, käynnistyy osoitteeseen <http://localhost:3000/>
- Katso koodia src-hakemistossa: index.js ja App.js - muokkaaminen lisätehtävä

```
my-app/  
  README.md  
  node_modules/  
  package.json  
  .gitignore  
  public/  
    favicon.ico  
    index.html  
  src/  
    App.css  
    App.js  
    App.test.js  
    index.css  
    index.js  
    logo.svg
```

Komponentit

- Komponentit ovat tyypillisesti omissa .js tiedostoissaan
- Muut komponentit importoidaan haluttuun paikkaan
- Myös CSS tiedostot voi muokata komponenttikohtaisesti
- Erityisesti jos projekti kasvaa, kannattaa usein lisätä uusi Components hakemisto, src hakemiston alle, ja sijoittaa komponentit sinne
- Huomaa, että käytämme uutta import lausetta emmekä require'ia joten moduulit käyttävät **export default** tiedoston lopussa - eivätkä **modules.export**
 - export default class, tai export default function
 - modules.export = {fun1, fun2} // vanha tyyli

Testaus

- package.json käynnistää *Jest test watcherin* kun ajaa komennon `npm test`
 - Jest lähellekään ainoa mahdollinen yksikkötestauskehys, mutta oletuksena käytetty
 - Mocha, Karma, .. ovat myös käytettyjä
 - Test watcherin saa pois päältä kun väittää Jestille, että CI on käytössä, eli muokkaa package.json tiedoston test-komentoa: `"set CI=true&&react-scripts test --env=jsdom"`
- Jest olettaa että kaikki tiedostot `__test__` hakemistossa, tai tiedostot joiden nimi päättyy `test.js`, ovat testitiedostoja
 - Generoidussa projektimallissa on `App.test.js` tiedosto, joka vain tarkistaa että `App` komponentti renderöityy normaalisti
- Kuten normaalia, kannattaa käyttää jotain staattista koodintarkistustyökalua, esimerkiksi ESLint

React komponentit

JSX
props

JSX

- JSX tulee sanoista JavaScript XML, ja se mahdollistaa JavaScript olioiden luonnin HTML:n kaltaisella syntaksilla
 - JSX speksit: <https://facebook.github.io/jsx/>
- JSX koodi täytyy jossain vaiheessa kääntää (transpile) JavaScriptiksi. Meillä on käytössä Babel, ja aluksi käännös tehtiin selaimessa. Nyt create-react-app –projektissa käännös tapahtuu jo kehitysympäristössä
- Reactia **voi** kirjoittaa myös ilman JSX:n käyttöä, mutta sitä pidetään yleensä liian työläänä vaihtoehtona
- Tämä osuus käy läpi JSX:n perusteita, myöhemmin tulee lisää detaljeja

React komponentti

- Creat React App:ssä komponentin perusrakenteen saa helposti tehtyä luomalla komponentin nimisen tyhjä js-tiedoston ja käyttämällä Reactjs code snippettiä rcc (eli kirjoittamalla rcc ja klikkaamalla tabulaattoria)

```
import React, { Component } from 'react';

class HelloWorld extends Component {
  render() {
    return (
      <div>
        Hello World
      </div>
    );
  }
}

export default HelloWorld;
```

Komponenttien hyödyntäminen

- Komponentin hyödyntäminen vaatii importin ja sen jälkeen sitä käytetään elementtinä hyödyksi. Esim. CRA:n (create-react-app) App.js:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import HelloWorld from './components/HelloWorld';
```

```
class App extends Component {
  render() {
    return (
      <div className="App">
        <HelloWorld />
      </div>
    );
  }
}
```

```
export default App;
```


Komponentit vs. elementit

- JSX voi renderöidä sekä HTML elementtejä, että React komponentteja

```
var componentClass = 'component';  
class MyComponent extends React.Component{/*...*/}  
var myElement = <MyComponent className={componentClass}/>;  
ReactDOM.render(myElement, document.getElementById('sample'));
```

- Jos käytetään HTML elementtejä, niin käytä niiden nimissä pieniä kirjaimia

```
var myElement = <div className={componentClass}/>;  
ReactDOM.render(myElement, document.getElementById('sample'));
```

- Omien komponenttien kohdalla aloita AINA luokan nimi Isolla alkukirjaimella
 - JSX on perinteisesti käsittänyt pienellä kirjaimella alkavat komponentit HTML elementeiksi

JSX vs. HTML

- Kertauksena **html** and **OmaKomponentti** (lisää myöhemmin)
- Nimeämiskäytäntö koskee myös muuttujia, eli jos renderöit muuttujan joka osoittaa komponenttiin, niin muuttujan nimen pitää myös alkaa isolla kirjaimella `OmalteToteutettuKomponentinNimi`
- Ehdollinen renderöinti on helppoa, sillä JSX ei renderöi arvoja `null`, `undefined`, tai `false`

```
<div>
  {isLoggedIn && <UserName />}
</div>
```

- JSX mahdollistaa JavaScript koodin kirjoittamisen komponentteihin, **mutta** ei renderöintiosaan - esim. `if`, `for` jne.
 - Kontrollirakenteet eivät ole *lausekkeita* siksi niitä ei voi renderöidä
 - Eli kirjoita kaikki JavaScript ennen `return` -osuutta `render()` funktiossa, ja käytä vain tuloksia renderöintiosuudessa

```
if (value < 0 ) value = -value;
return (<div>Rendering value: {value}</div>);
```

Komponentit

- Jokainen komponentti on React-luokka
- Luokan render() metodia kutsutaan kun komponentin pitää piirtä itsensä
- Luokkaan lisätään tilanhallintaa, tapahtumankäsittelyitä ja muuta koodia myöhemmin
- Komponenttia käytetään sen nimellä joko ReactDOM.render-kutsussa tai muista React komponenteista kutsuttuna
- Render-funktio palauttaa HTML palan, jonka haluamme dokumenttiin komponentin kohdalle
- Pari vinkkiä:
 - Render voi palauttaa vain yhden React(/html) elementin, sen sisällä voi olla lapsia niin paljon kuin on tarvetta
 - Sulje palautettava JSX osuus sulkeilla

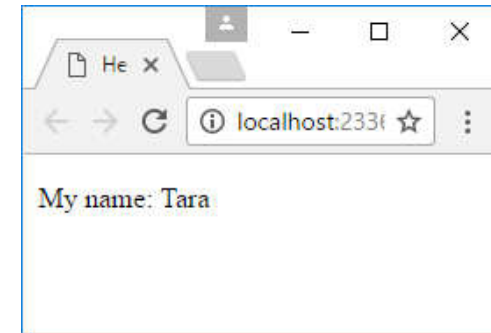
```
class MyComponent extends Component {  
  render () {  
    return(  
      <h1>My component.</h1>  
    );  
  }  
}
```

```
class MyComponent extends Component {  
  render () {  
    return(  
      <p>  
        Text <b>element</b> text  
      </p>  
    );  
  }  
}
```

Ominaisuudet/muuttujat/komponenttiparametrit: props

- **props** toimii datan välittäjänä komponenteille
- Niitä voi käyttää kuten parametreja

```
class MyComponent extends Component {  
  render () {  
    return(  
      <p>My name: {this.props.name}</p>  
    );  
  }  
}  
ReactDOM.render(  
  <MyComponent name="Tara"/>,  
  document.getElementById('content'));
```



- Kuten normaalisti this-sanan käyttö on kontekstista riippuvaista
- Koodi on JSX, ei puhdasta JavaScriptiä, joten JavaScript lausekkeet täytyy kirjoittaa aaltosulkeiden {} sisälle

ES6+ komponentit ja props

- ES6+ syntaksin mukaiset komponentit, jotka eivät tarvitse tilanhallintaa, tai käytä elinkaarimetodeja, voi kirjoittaa yksinkertaisesti funktioina
- Kaikki esimerkit alla määrittelevät saman komponentin, voit valita itsellesi parhaiten sopivan

```
class MyHello extends Component {  
  constructor(props) {  
    super(props);  
  }  
  render() {  
    return (<p>Hello {this.props.name}!</p>);  
  }  
}
```

```
class MyHello extends Component {  
  render() {  
    return (<p>Hello {this.props.name}!</p>);  
  }  
}
```

```
function MyHello(props) {  
  return (<p>Hello {props.name}!</p>);  
}
```

```
const MyHello = (props) => (<p>Hello {props.name}!</p>);
```

Harjoitus 3

- Toteuta tämä Harjoituksessa 2 luomaasi CRA-projektiin.
- Tee oma Kayttaja-komponentti Reactilla, ja näytä se sivulla
- Koita ensin tekemällä komponentti, joka näyttää staattista tekstiä
- Muokkaa sitten komponentin render() funktiota siten, että se näyttää Table-elementin, jossa kolme saraketta: nimi, sähköposti, ikä - näille otsikot (<th>, ja yksi rivi jossa valmiiksi kirjoitetut arvot, esimerkiksi:

Nimi	Sähköposti	Ikä
Seppo Hovi	sepe@yle.fi	71

- Seuraavaksi anna näytettävän henkilön tiedot parametrina (props) komponentillesi
- Lisätehtävä: käytä muitakin läpikäytyjä Reactin piirteitä

Lisää props ja JSX -asiaa

- HTML attribuutit vaativat usein hieman uuden variantin JSX koodissa

- Esimerkiksi className normaalin class sijaan

Huom: välttä käyttämästä JavaScript avainsanoja muuttujina/attribuutteina

```
<p className="sample">My name: {this.props.name}</p>
```

```
<div id="content">
  <p data-reactroot="" class="sample">...</p>
</div>
```

- Muunnos muuntaa JSX koodin JavaScriptiksi, joten `<input name='ip' defaultValue='A' />` muuntuisi muotoon: `React.createElement('input', {name: 'ip', defaultValue: 'A'})`;
- Babel REPL näyttää webissä muunnoksen: <https://babeljs.io/repl/>

```
var MyComponent;
var myComp = <MyComponent attr='value' />;
/* Converts to */
var MyComponent;
var myComp = React.createElement(MyComponent, { attr: 'value' });
```

- HTML to JSX Compiler näyttää kuinka HTML pitäisi kääntää JSX koodiksi <http://magic.reactjs.net/htmltojsx.htm>

Lisää propseista

- Muuttuja props on siis JavaScript muuttuja, joten sen arvoina voi käyttää mitä tahansa validia JavaScriptia

```
class MyComponent extends React.Component {  
  render () {  
    return(<div>Name: {this.props.person.name}</div>);  
  }  
}  
var person = {name: 'John Doe', email: 'john@acme.org'};  
ReactDOM.render(<MyComponent person={person}/>,  
  document.getElementById('content'));
```

Name: John Doe

- Luonnollisesti voidaan käyttää myös useita muuttujia

```
class MyBook extends React.Component {  
  render () {  
    return(<div>A book &quot;{this.props.title}&quot; by: {this.props.author}</div>);  
  }  
}  
var book = {author: 'Jane Doe', title: 'One, two, React!'};  
ReactDOM.render(<MyBook author={book.author} title={book.title} />,  
  document.getElementById('sample'));
```

A book "One, two, React!" by: Jane Doe

Harjoitus 4 - aforismikomponentti

- Toteuta komponentti, joka näyttää sanonnan (aforismi) ja sen kaksi jäsentä: teksti ja sanoja
- Näytä sanonnan teksti, ja keneltä sanonta on peräisin, mutta eri riveillä
- Lisätehtävä:
 - Näytä toinenkin komponentti samalla sivulla, esim. Käyttäjä / Henkilö
 - Lisää hieman tyyliä, eli CSS määrittystä

Koosteiset komponentit

Composite components

Usean komponentin palauttaminen

- "React komponentti voi palauttaa vain yhden juurielementin", mutta elementtejä/komponentteja voi palauttaa kuinka paljon vain, *kunhan ne on yhdistetty yhden komponentin alle*
- Tämä on oikeastaan jo tehty, sillä HTML elementithän (ja tekstisolmut) voi ajatella React elementeiksi
- Erityistä `children` propsia käyttäen voi käsitellä lapsikomponentteja

```
<ChildrenProps>One <b>two</b> three</ChildrenProps>
```

```
return(<div>Got {this.props.children.length} children,  
      which are: {this.props.children}</div>);
```

```
Got 3 children, which are: One two three
```

- Kuten niin usein muutenkin, selaimen kehittäjätyökalut toimivat mainiosti tarkistuksessa

```
▼ <div id="content">  
  ▼ <div data-reactroot>  
    <!-- react-text: 2 -->  
    "Got "  
    <!-- /react-text -->  
    <!-- react-text: 3 -->  
    "3"  
    <!-- /react-text -->  
    <!-- react-text: 4 -->  
    " children, which are: "  
    <!-- /react-text -->  
    <!-- react-text: 5 -->  
    "One "  
    <!-- /react-text -->  
    <b>two</b>  
    <!-- react-text: 7 -->  
    " three"  
    <!-- /react-text -->  
  </div>  
</div>
```

Omat koosteiset komponentit

- Aivan kuten usean HTML elementin kanssa, voimme renderöidä myös omia React komponentteja

```
function MyComponent(){  
  return (<p>Hic sunt dragones &#9967;</p>);  
}
```

```
class MyComposite extends Component{  
  render(){  
    return(  
      <div>  
        <h1>Composite</h1>  
        <MyComponent/>  
        <hr/>  
      </div>  
    );  
  }  
}  
ReactDOM.render(<MyComposite/>, document.getElementById('content'));
```

Composite

Hic sunt dragones ☼

- Vaikka esimerkissä yllä renderöidään vain yksi oma komponentti, niin vastaavalla tavalla niitä voisi renderöidä useitakin

Omistus ja props

- React-koodissa vanhemmat välittävät propsit lapsilleen
- Esimerkki: näytä kaksi henkilöä taulussa

```
import React, { Component } from 'react';
import PeopleTable from './PeopleTable';
class Page extends Component {
  render() {
    return (
      <div>
        <h1>People</h1>
        <PeopleTable people={data}/>
      </div>
    );
  }
}
var data = [{ id: 4, name: 'John Doe', email: 'john@acme.org' },
  { id: 2, name: 'Jane Doe', email: 'jane@acme.org' },];
export default Page;
```

People

John Doe john@acme.org

Jane Doe jane@acme.org

```
import Person from './Person';
class PeopleTable extends Component {
  render() {
    return (<table>
      <tbody>
        <Person person={this.props.people[0]} />
        <Person person={this.props.people[1]} />
      </tbody>
    </table>);}}}
```

```
const Person=(props)=>
  <tr>
    <td>{props.person.name}</td>
    <td>{props.person.email}</td>
  </tr>
  /*hieman erilainen syntaksi componentin
  luonnissa */
```

Vaihteleva määrä lapsia

- Usein lasten määrä ei ole etukäteen tiedossa, vaan määrä saadaan vasta ajon aikana
 - Edellisen sivun kovokoodattu kaksi henkilöä ei pitäisi tuntua hyvältä ratkaisulta
- JavaScriptin arrayn `map()`-funktio auttaa
- Huomioitavaa
 - `map()` vaatii uniikin `key`-attribuutin lapsille (Jos ei löydy, löydät varoitukset consolesta)
 - Esimerkiksi tietokannasta saattaa tulla yksilöllinen `id`, mutta alla esimerkki `map()` funktion automaattisesti antamasta `index`-parametrasta. Jos käytät `index`-muuttujaa, niin taulukon sisältöä **ei** saa muuttaa

```
import React, { Component } from 'react';
import Person from './Person';
class PeopleList extends Component {
  render() {
    var peoplerows = this.props.people.map(function(person, index) {
      return ( <Person person={person} key={index}/> );
    });
    return (
      <table><tbody>
        {peoplerows}
      </tbody></table>
    );
  }
}
export default PeopleList;
```

Muuta huomioitavaa

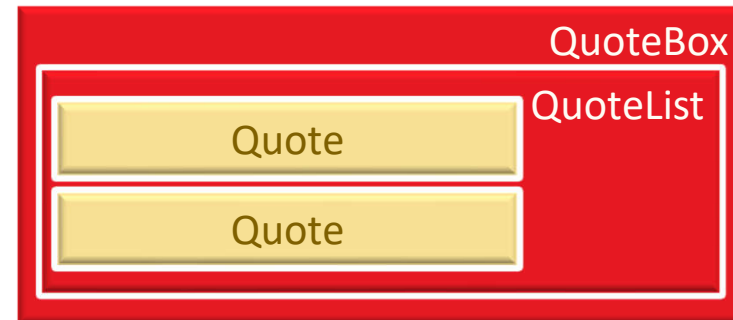
- JavaScriptiä voi kirjoittaa luonnollisesti enemmänkin

```
class PeopleList extends Component {  
  render () {  
    // Määrittele rivi Person-olioita käyttäen map() funktiota,  
    // mutta vasta kun ne on sortattu id'n mukaan  
    var people = this.props.people.sort(function(p1, p2){  
      return p1.id-p2.id}).map(function(person) {  
        return (<Person person={person} key={person.id}/>);  
      });  
    // Lisää Person oliot, eli rivit, tauluun  
    return (  
      <table><tbody>  
        {people}  
      </tbody></table>  
    );  
  }  
});
```

- Voimme myös luoda uusia funktioita, käyttää muuttujia, tapahtumankäsittelyä jne. näistä esimerkkejä myöhemmin

Harjoitus 5: Koosteiset komponentit

- Seuraavaksi tehdään koosteisia komponentteja sisältävä sivu
1. Projektin rakenne create-react-app generoijalla komentoriviltä tai suoraan IntelliJ:ssä
 - Luo uusi projekti, esimerkiksi `c:\course` hakemistoon: `create-react-app sanontaharj`
 2. Harjoituksessa tehdään kolme komponenttia:
 1. QuoteBox - Pääkomponentti, koosteinen - luo ja omistaa sanonta-datan näin aluksi
 2. QuoteList - Lista sanontoja, koosteinen
 3. Quote - Yksittäinen sanonta
 3. Muokkaa App.js komponenttia siten, että se näyttää QuoteBox komponentin
 4. Luo sanonnat QuoteBox moduulissa, esittele komponentin koodin alla taulukko sanontoja {id, author, quotetext} ja välitä ne QuoteList komponentille JSX-osuudessa propsina
 - Saat halutessasi muutaman valmiin sanonnan opettajilta



Harjoitus 5 jatkoa

- Komponenttien toteutus:

1. QuoteBox:

1. Näytä sovelluksen otsikko `<h1>` elementtinä
2. näytä QuoteList otsikon jälkeen
3. Määritä sanonnat globaalina taulukkona QuoteBox.js tiedostoon, ja anna QuoteBox luokan välittää tiedot QuoteList komponentille propsina

2. QuoteList:

1. Käytä sanonta propsia, ja luo `map()` funktiolla Quote olioita niiden perusteella
2. Palauta renderistä generoidut Quotet div-elementin sisällä

3. Quote:

1. Näytä sanonnan teksti, ja keneltä se on peräisin renderissä

4. App.js

1. *Muokkaa* App.js tiedostoa poistamalla renderissä oleva koodi, ja renderöi vain QuoteBox sieltä

QuoteList esimerkki

- Esimerkki alla välittää sanonnan tiedot kahtena attribuuttina Quote komponentille. Koko sanonnan voisi hyvin välittää myös yhtenä attribuuttina.

```
class QuoteList extends Component {
  render () {
    var quoteNodes = this.props.data.map(function(quote) {
      return (
        <Quote author={quote.author} text={quote.quotetext} key={quote.id}>
        </Quote>
      );
    });
    return (
      <div className="quoteList">
        {quoteNodes}
      </div>
    );
  }
}
```

Tilan hallinta

Tila (state), ja alkutila (initial state)
Lomakkeet
Tapahtumien käsittely

Tila

- Kuten props myös state muuttuja on sisäänrakennettu React-komponentteihin
- Tilan muutos aiheuttaa myös komponentin uudelleen piirtämisen (render), eli props muuttujien arvoa ei yleensä muuteta, vaan käytetään tilamuuttujaa
- Alkutila (Initial state), eli tila-muuttujan arvojen alustus on usein pakollista
 - Otetaan esimerkiksi komponentti, joka näyttää dataa joka haetaan verkosta. Data on käytettävissä vasta myöhemmin, mutta jo alussa pitää näyttää jotain
- Triviaali esimerkki (vain osa koodista), joka asettaa tila-muuttujalle alkuarvoja

```
// ES6+:
class Person extends React.Component {
  state = { name: '', email: '' };
  render () {
    return (
      <p>{this.state.name}: {this.state.email}</p>
    );
  }
  ...
}
```

Lomakkeet

- Koska React on vastuussa näkymästä (MVC:n V), niin sen vastuulla on ulkoasu
- Erityisen hyvin tämä käy ilmi kun tehdään ensimmäisiä lomakkeita Reactilla
- Edes tavalliset tekstityyppiset input elementit eivät näytä tekstiä ilman koodia
 1. Luo uusi input komponentti
 2. Anna Reactin huolehtia arvon muuttamisesta
 3. Sido inputin arvo tila-muuttujaan, käytä tapahtumankäsittelyä näkymän muuttamiseen

```
// 2. (HTML: <input name='input' value='A'/>)
const myInput = <input name='input' defaultValue='A' />;
ReactDOM.render(myInput, sample);
```

```
class PersonForm extends React.Component { // 3. render-funktio palauttaa uuden formin, sidonta
  // tapahtumankäsittelijöihin, toteutus myöhemmin
  render () {
    return (
      <form>
        <input type="text" placeholder="Name"
          value={this.state.name} onChange={this.handleChange}/>
        <input type="text" placeholder="email"
          value={this.state.email} onChange={this.handleEmailChange}/>
        <input type="submit" value="Create" onClick={this.handleClick}/>
      </form>
    );
  }
}
```

Tapahtumankäsittely ES6+

- JavaScriptin this-muuttujan on arvo viittaa paikalliseen kontekstiin, oletusarvoisesti funktio, eikä esimerkiksi luokka. This täytyykin saada osoittamaan ympäröivän luokan määrittelemään oloon
- Kaksi mahdollisuutta: nuolisyntaksi (ES7), tai käytä bind-funktiota

- Bind:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {big: false};
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() { this.setState({big: !this.state.big}); }
  render() {
    const msg = 'Here\'s looking at you, ' + this.props.name + '!';
    return( <div onClick={this.handleClick}>
      {this.state.big ? msg.toUpperCase() : msg}
    </div>);
  }
};
```

- Nuoli:

```
class MyOtherComponent extends React.Component {
  state = {big: false};
  handleClick = () => { this.setState({big: !this.state.big}); }
  render() { /* kuten yllä */ }
```

- Huomaa kirjoitusasu onClick, ei onclick

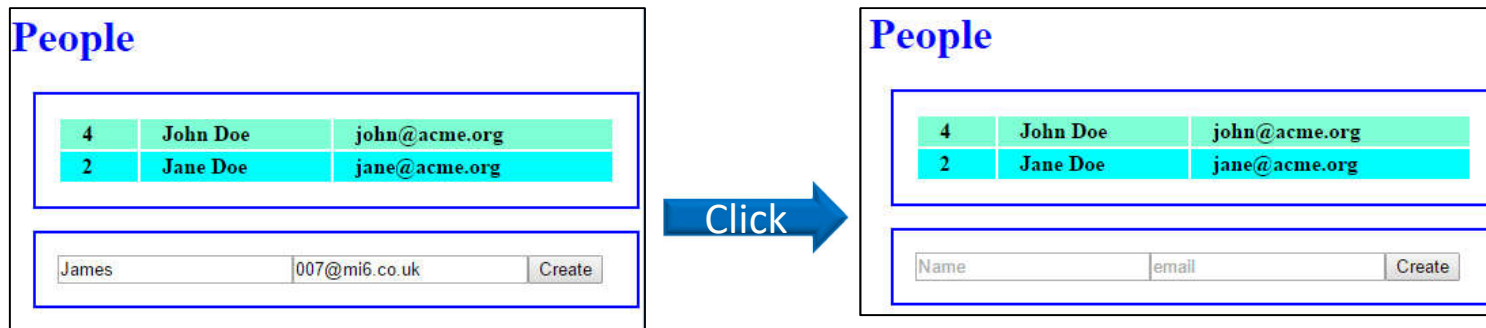
```
Warning: Unknown event handler property onclick.
Did you mean `onClick`?
    in div (created by MyComponent)
    in MyComponent
```

Lomake-esimerkki tapahtumankäsittelijöineen

```
export default class Lomake extends Component {
  state = {name:'', email:''};
  handleEmailChange = (e) => { this.setState({email: e.target.value}); }
  handleNameChange = (e) => { this.setState({name: e.target.value}); }
  handleClickCreate = (e) => {
    e.preventDefault();
    alert(`Nimi: ${this.state.name}, sposti: ${this.state.email}`);
    this.setState({name:'', email:''});
  }
  render() {
    return ( <form>
      <input type="text" placeholder="Name"
        value={this.state.name} onChange={this.handleNameChange} />
      <input type="text" placeholder="email"
        value={this.state.email} onChange={this.handleEmailChange} />
      <input type="submit" value="Create" onClick={this.handleClickCreate} />
    </form>);
  }
}
```

Tilan muutos

- Aiempi esimerkki mahdollisti henkilöiden lisäämisen taulukkoon, mutta muutokset eivät olleet heti näkyvissä



- Syy tähän oli, että vaikka PersonForm päivitti tietoa, se ei kertonut asiasta PeopleList komponentille - joten muutokset listassa eivät tule näkyviin
- Tyypillinen tapa saada myös sisar-komponentit päivittämään itsensä, on välittää juurikomponentilta callback-funktio lomakekomponentille, ja lomakekomponentti kutsuu callbackia - näin juurikomponentti päivittää itsensä ja samalla kaikki lapsensa

Komponentit - lisää

Päivittyvät sivut: REST
Uudelleenkäytettävät komponentit

REST kutsut

- Modernit Web applikaatiot saavat tyypillisesti osan / kaikki näyttämänsä datan palvelimelta ajon aikana dynaamisesti
- REST palvelun käyttäminen on varsin tärkeä osa React-ohjelmointia
- Yksinkertainen perustoteutus:
 - Kuten aiemmin todettiin, käytä **componentDidMount** metodia tehdäksesi REST-kutsun. Kutsun tekninen toteutus valittavissa XHR, jQuery, Fetch, Axios, tai mitä vain
 - Tee REST-kutsu tarpeeksi ylhäällä (kuten datan omistamisessa aiemmin)
 - Tee tapahtumankäsittely sekä onnistuneessa että virheellisessä tilanteessa

```
class Page extends React.Component {
  state = {data: []}
  componentDidMount = () => {
    this.xhr = new XMLHttpRequest();
    this.xhr.onreadystatechange = this.onReadyStateChanged;
    this.xhr.open('GET', '/api/palvelu', true);
    this.xhr.send();
  }
  onReadyStateChanged = () => {
    if (this.xhr.readyState == 4 && this.xhr.status == 200) {
      this.setState({data: JSON.parse(this.xhr.responseText)});
    }
  }
}
```

Variantit

- React komponenttien propsien käyttöä ei pidä tehdä liikaa
- Tapahtumankäsittelyn voi siis tehdä myös anonyymissä funktiossa

```
class Page extends React.Component {  
  state = {data: []}  
  componentDidMount = () => {  
    var xhr = new XMLHttpRequest();  
    xhr.onreadystatechange = function() {  
      if (xhr.readyState == 4 && xhr.status == 200) {  
        this.setState({data: JSON.parse(xhr.responseText)});  
      }  
    }.bind(this); // pakollinen, muuten this.setState ei onnistu  
    xhr.open('GET', this.props.url, true);  
    xhr.send();  
  }  
}
```

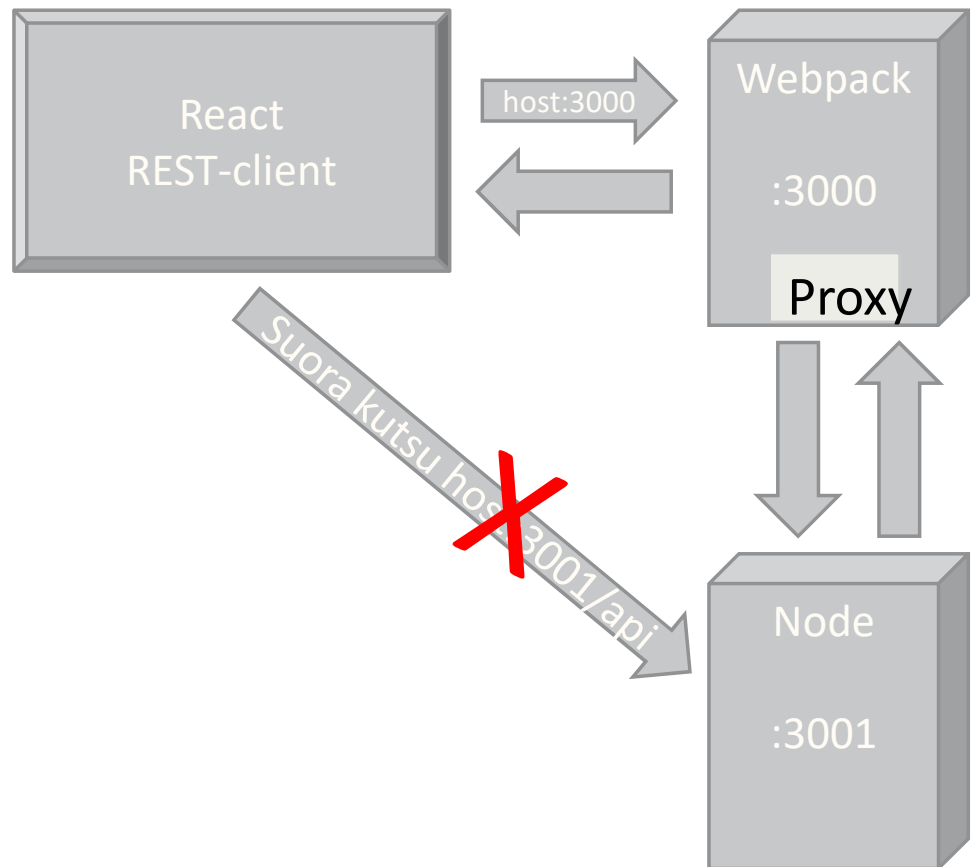
- Muista: käsittelijä on sidottava (bind) oikeaan komponenttiin. Ilman sidontaa this-muuttuja voi viitata väärään olioon

```
✖ ▶ Uncaught TypeError: this.setState is not a function  
    Inline Babel script:35
```

- Jos dataa on haettava jatkuvasti, voit käyttää pollaamista tai Websockettia - muista pysäyttää componentWillUnmount metodissa
 - Pollaaminen esimerkiksi JavaScriptin setInterval funktiolla

Kehitysvaiheen palvelinkonfigurointi

- Kehitysvaiheessa React-sovellus ladataan Webpack-palvelimelta
 - Oletuksena portissa 3000
- REST-palvelin usein eri projekti, esimerkiksi Node-projekti
 - Muista huomioida palvelinten portit! Kahta palvelinta ei voi käynnistää samaan porttiin
- Eri palvelimella, ml. eri portissa, olevaa palvelua voi kutsua vain jos CORS enabloitu
- Proxy-määrittelyksellä voidaan kutsua eri portissa olevaa palvelua ikään kuin se olisi samalla palvelimella
- Tiedostoon package.json rivi:
 - "proxy": "http://localhost:3001/"



Harjoitus 10.7.5

- Ota palvelimeksi joko jokin oma aiempi REST-palvelun toteuttava Node projekti, tai jokin opettajien projekti
- MUISTA porttimääritys, **ei** kahta palvelinta porttiin 3000
- Näytä oliot taulussa (tai listassa) React sivulla
- Tee uusi React applikaatio
- package.json-tiedostoon kannattaa lisätä proxy-rivi, joka ohjaa REST pyynnöt Node-palvelimelle. Nyt kutsun voi tehdä ikään kuin samalla palvelimelle mistä React-sovellus itse on ladattu
 - "proxy": "http://localhost:3001/"
- Toteuta App.js listan hakeminen ja näyttäminen
 - User / Maa / jokin muu komponentti listan alkioille

Harjoitus 10.08 - Node ja Postgres mukana (1/4)

- Toteuta Sanonta-applikaatio React sovelluksena
- Tee oma Node.js palvelin joka käyttää PostgreSQL tietokantapalvelinta sanontojen tallettamiseen
- Opettajilta saatavilla REST palvelu käyttäen Nodea ja SQLite-tietokantaa
 - Voit myös ottaa sen pohjaksi, ja esimerkiksi muokata sen käyttämään PostgreSQL kanta
- Palvelimen toteutus:
 - REST rajapinta sisältää perustoiminnot, seuraavalla sivulla kuvaus
- **Lisätehtävänä:** lisää myös Historia-ominaisuus, jossa talletetaan sanontojen muokkaushistoria

Harjoitus 10.08: Rajapinta (2/4)

- `http://localhost:8080/api/quotes`
 - GET - palauta kaikki sanonnat
 - DELETE - poista kaikki sanonnat
 - POST - luo uusi sanonta, tietokanta generoi id:n
- `http://localhost:8080/api/quotes/:id`
 - GET - palauta sanonta id:n perusteella
 - PUT - muokkaa id:n perusteella löytyvää sanontaa, välitetyn sanonnan id:llä ei ole väliä, polussa oleva ratkaisee
 - DELETE - poista sanonta
- `http://localhost:8080/reset`
 - **GET** - poista kaikki sanonnat, ja alusta muutamalla oletussanonnalla
 - (tämä ihan harjoituksen vuoksi)
- Voit toteuttaa muutakin toiminnallisuutta halutessasi, palvelinkoodaushan on tuttua ja turvallista

Harjoitus 10.08 - React-osuus (3/4)

- Muokkaa aiempien harjoitusten Sanonta-applikaatiota siten, että käytät palvelimen REST rajapintaa aiemman taulukon sijaan
 - Lisää package.json tiedostoon proxyn asetus
- 1. Luo uusi moduuli (eli tiedosto) serviceclient.js johon toteutat REST kutsut
 - Toteuta aluksi ainakin haesanonnat ja luosanonta - yksi kerrallaan luonnollisesti, aloita haesanonnat funktiosta, joka hakee kaikki sanonnat palvelimelta
 - Kukin funktio palauttaa Promisen, tai ottaa parametrinaan callback-funktion, jota kutsut kun haku/luonti on valmis
 - Esimerkki luosanonta-funktiosta (callback variantti siinä kommentteissa)

```
const palveluurl = '/api'; // kun proxy asetettu
export function luosanonta(sanonta /*, callback*/) {
  return fetch(palveluurl+'sanonta/', {
    method: 'POST',
    headers: {'Content-Type': 'application/json' },
    body: JSON.stringify(sanonta)
  })
  .then(function(response) {
    /*callback(response);*/
    return (response);
  });
} // tätä voi käyttää pohjana
```


Harjoitus 10.08 - komponentit (4/4)

1. Muokkaa QuoteBox siten, että siinä ei enää ole kovokoodattua data-taulukkoa, vaan se hakee datan käyttäen serviceclient moduulin funktioita, ja tallettaa tulokset tila- (state) muuttujaan
 1. Ota käyttöön: `import {haesanonnat} from './serviceclient';`
 2. Tee ensin `componentDidMount()` elinkaarimetodi, ja kutsu `haesanonnat` funktiota sieltä
 3. Toteuta myös `callback` funktiot, joita `serviceclientin` funktiot kutsuvat kun haku (tms.) on valmis
 4. Toteuta nyt uusiksi uuden sanonnan luonti (ja sanonnan tuhoaminen) sopivalla kutsulla johonkin `serviceclientin` funktioon, muista `callbackit`
2. Jatka sitten kohta kohdalta REST rajapinnan hyödyntämistä React-applikaatiosta
 - Huomio tiedon omistuksesta: periaatteessa nyt voisi siirtää `serviceclientin` kutsuja alemmas komponenttihierarkiassa, **mutta** silloin oikein tehty uudelleenpiirto (renderöinti) saattaa muodostua hankalaksi - esimerkiksi kun on luotu uusi sanonta
 - Esimerkiksi `this.forceUpdate()` kutsu `QuoteBoxin` koodissa ei saisi `componentWillMount` metodia suoritukseen `QuoteList` komponentissa

Tuotantototeutus

- Tuotantoon viettäessä JavaScript-koodi pitää kääntää palvelimessa valmiiksi paketiksi, samoin css
- Projektin npm run build tekee juuri tämän, samalla se yhdistää tiedostot ja minifioi generoidut tulokset
 - Katso build-hakemiston alta
- Buildin tuloksen voi siirtää tuotantopalvelimelle tiedostoina, ja kaiken pitäisi toimia moitteetta
 - Eli node-palvelimen siihen hakemistoon, joka tarjoaa staattiset tiedostot

Harjoitus 10.09: Build ja deploy

- Katso saatko toteutetun React applikaation käännettyä ja pyörimään REST-palvelun toteuttavan node-projektin sisällä
- Käännä ja minifioi projekti ajamalla build komento
- Kun käännös on valmis, kopioi luodut tiedostot build-hakemiston (myös alihakemisto static) alta REST projektin public (tai static tai files) hakemistoon
- Käynnistä REST-palvelin ja varmista, että projekti löytyy osoitteesta `http://localhost:3000/` (tai 3001)
 - Älä siis käynnistä React-projektin palvelinta lainkaan

React.Component.setState()

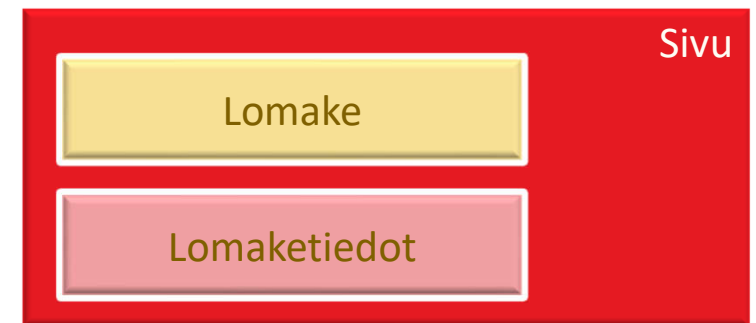
- Komponentin setState kutsu aiheuttaa automaattisesti render funktion kutsun
 - Tämä myös vaikka tila ei varsinaisesti muutu, esimerkiksi this.setState(this.state);
- Eli PeopleList ja toimivat muutokset

```
class App extends React.Component {
  state = {people: this.props.data}
  handlePersonAdded = (p) => {
    this.state.people.push(p);
    this.setState(this.state); // Pakota tämän ja lasten renderöinti
  }
  render () {
    return (<div>
      <h1>People</h1>
      <PeopleList people={this.state.people}/>
      <PersonForm people={this.state.people} newid={this.props.newid}
        addperson={this.handlePersonAdded} />
    </div>);
  }
};
```

```
// Class PersonForm:
handleCreateClick = (e) => {
  e.preventDefault(); // Ei lähetetä formia palvelimelle
  this.props.addperson(this.state); // Muutos tällä rivillä
  var newid = this.state.id+1;
  this.setState({ id: newid, name: '', email: ''});
}
```

Harjoitus 10.06 - lomake

- Toteuta sivu, joka näyttää lomakkeen, sekä lomakkeen alla tiedot jotka lomakkeelle on viimeksi kirjoitettu
- Käytä esimerkiksi Henkilöä, eli vaikka
 - Etunimi ja sukunimi
- Sijoita siis lomakkeelle
 - kaksi input elementtiä
 - submit button
- Lomaketiedot komponentilla näytä viimeksi muuttuneet tiedot
- Sivukomponentti omistaa Henkilö-olion ja välittää sen sekä Lomakekomponentille, että LomakeTiedotkomponentille
- Tee lomakkeelle tapahtumankäsittelijät input-elementtien käsittelyyn. Toteuta Sivukomponentille submit-käsittely, ja välitä siihen viittaus Lomakekomponentille



Muita Form-elementtejä

- Value-attribuuttiin ei aseteta arvoa suoraan, silloin React ei renderöisi muutoksia
- Eli jos arvo halutaan asettaa (esim. text-input/textarea), esimerkiksi aluksi, käytä **defaultValue** attribuuttia. Arvon muutokseen käytä **state**-muuttujaa joka asetetaan **onChange** tapahtuman tullessa
- Radio buttonit ja checkboxit: React käyttää click-tapahtumaa, eikä change (kuten normaalissa JavaScriptissä olisi)
 - Huomioi että näiden kanssa **ei** käytetä preventDefault kutsua

```
<input type="checkbox" value={this.state.rastiruudussa} onClick={this.toggleCheckbox}/>
```

```
toggleCheckbox = (e) => {  
  this.setState({rastiruudussa : !this.state.rastiruudussa});  
},
```

Like button -esimerkki

- Alkuperäinen koodi React dokumentaatiosta (muutettu muutaman kerran)

```
class LikeButton extends React.Component {
  state = {liked: false};
  handleClick = () => {
    this.setState({liked: !this.state.liked});
  }
  render() {
    var text = this.state.liked ? 'liked' : 'haven\'t liked';
    return (
      <div>
        Click to toggle:
        <div onClick={this.handleClick} className="btn">
          You {text} this.
        </div>
      </div>
    );
  }
}

ReactDOM.render(<LikeButton />,
  document.getElementById('example') );
```

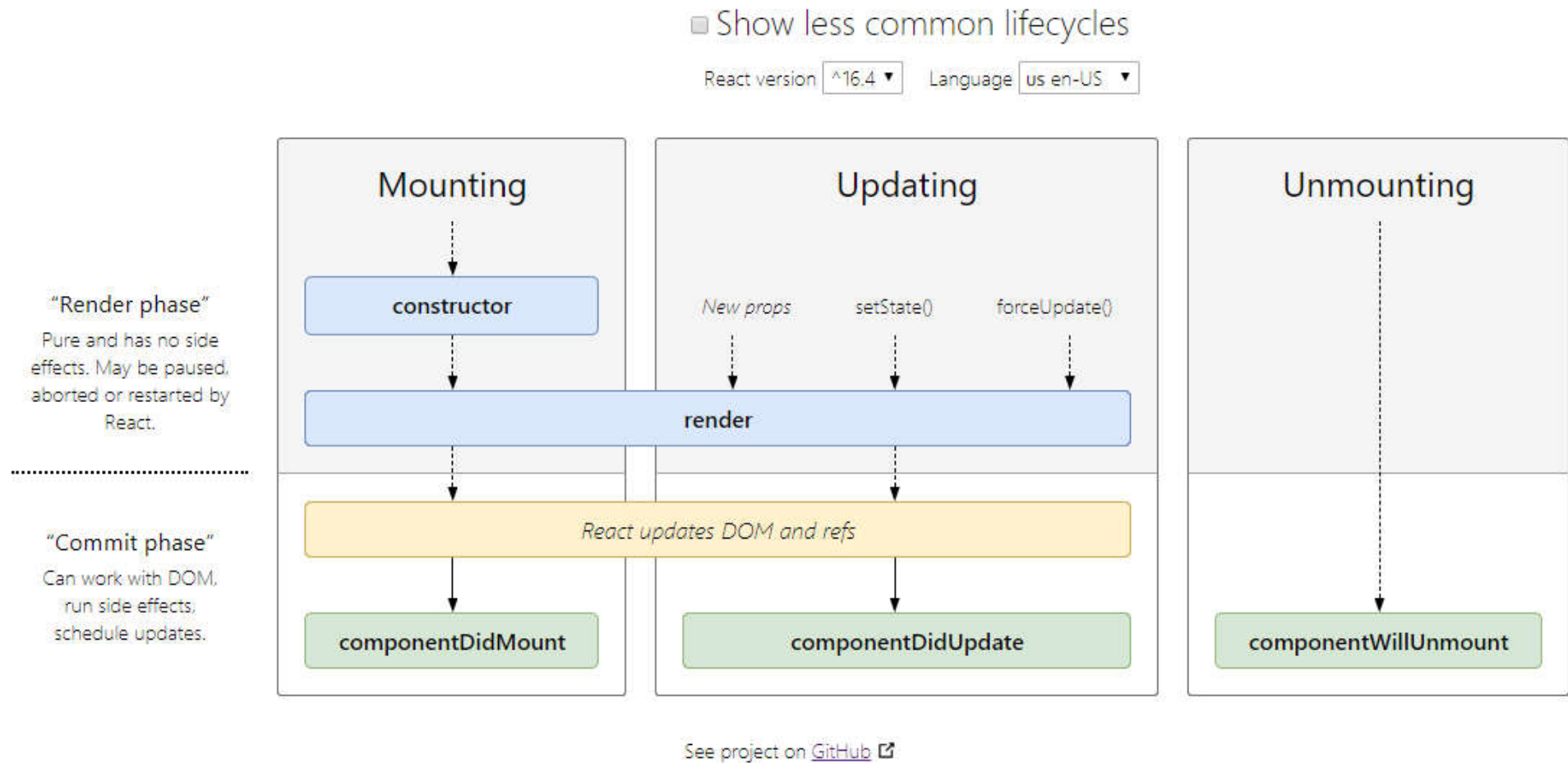
Click to toggle: **You haven't liked this.**

Click to toggle: **You liked this.**

Elinkaari

- Olemme jo nähneet yhden elinkaarimetodin: **render** - varsinaisesti se on Component spesifikaatiosta -mutta speksit ja elinkaarimetodit ovat hyvin lähellä toisiaan
- **render** on pakollinen, jos et halua piirtää: return null tai false
- **getInitialState** poistui ES6+ koodin kanssa
 - Käytä konstruktori / state property alustus
 - Sitten deprekoituja metodeita: `componentWillMount`, `componentWillUpdate`, `componentWillReceiveProps`, ...
- Lifecycle metodeita joita kutsutaan sopivassa vaiheessa komponentin elinkaarta ovat mm.
 - **konstruktori**: kutsutaan juuri ennen kuin komponentti renderöiden ensimmäistä kertaa. tilaa voi vielä vaihtaa täällä - *deprekoitu*
 - **componentDidMount**: kutsutaan heti sen jälkeen kun komponentti on renderöity ensimmäistä kertaa. Huom: lasten `componentDidMount` metodia kutsutaan ennen vanhemman vastaavaa. Tämä metodi on hyvä tehdä (aloittaa) REST-kutsut
 - **componentDidUpdate**: kutsutaan renderöinnin jälkeen, ensimmäistä lukuun ottamatta
 - Muita: **shouldComponentUpdate**, **componentWillUnmount**

Elinkaarimetodit

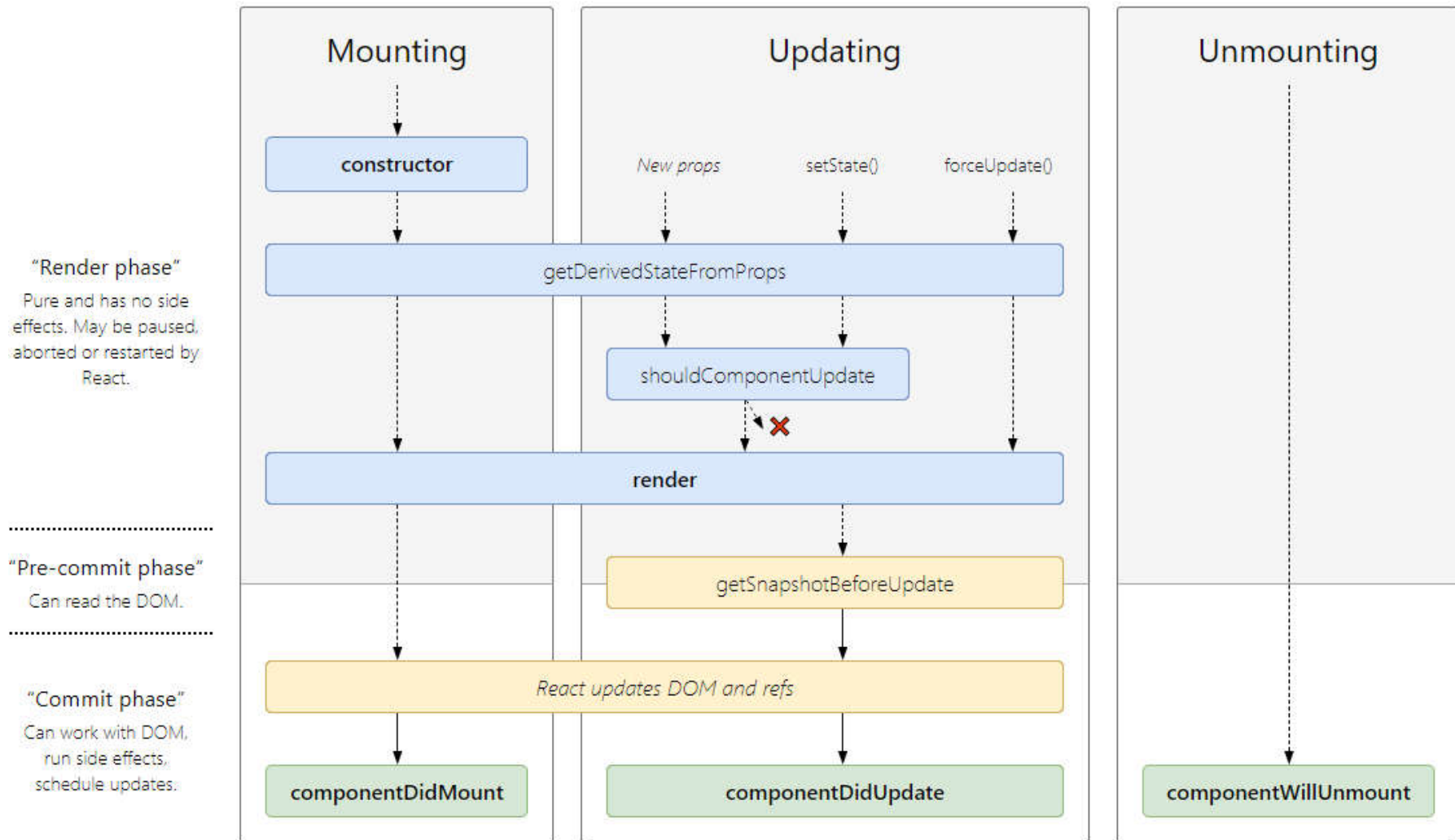


<http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

☑ Show less common lifecycles

React version

Language



See project on [GitHub](#)

Harjoitus 10.07: Sanontalomake

- Lisää käyttäjälle mahdollisuus lisätä sanontoja. Luonnollisesti lomakkeella kun HTML:stä on kyse
- Jatka edellisen harjoituksen parissa, lisää uusi komponentti esimerkiksi sivun loppuun - sisarukseksi QuoteList komponentille
- Tee siis uusi komponentti: QuoteForm
- QuoteForm renderöi lomakkeen ja päivittää tiedot
 - Käytä edelleen taulukossa olevaa dataa, edelleen QuoteBox-komponentin omistuksessa
- Lisää myös QuoteBoxiin tapahtumankäsittelijä uuden sanonnan lisäämiseksi taulukkoon

Harjoitus 10.07: Sanontalomake, toteutusta

1. Muokkaa QuoteBox komponenttia, lisää uusi QuoteForm komponentti

1. Luo uusi tapahtumankäsittelijäfunktio, joka lisää parametrinaan saamansa sanonnan taulukkoon. Tämä on myös hyvä paikka lisätä sanonnalle id, jos käytät sanontojen yhteydessä olevia uniikkeja id-arvoja
2. Välitä viittaus uuteen funktioon attribuuttina QuoteFormille

2. Toteuta QuoteForm komponentti

1. Alusta tila (state) konstruktorissa
2. Tulosta lomake render() funktiossa
 - Seuraavalla sivulla pieni koodiesimerkki, ei tarvitse käyttää, mutta saa
3. Toteuta tapahtumankäsittelijät jotta input-kentät oikeasti näyttävät muokatun tekstin
4. Submit-painikkeen käsittelijän pitäisi kutsua QuoteBoxin välittämää callback-funktiota (anna parametriksi tila-muuttujassa oleva uusi sanonta)

Lomakkeen koodi

```
<form>
  <p>
    <label htmlFor="form_quotetext">Sanonta</label>
    <textarea type="text" placeholder="quote" id="form_quotetext"
      value={this.state.quotetext} onChange={this.handleQuoteChange}
      required="required"></textarea>

  </p>
  <p>
    <label htmlFor="form_author">Sanoja</label>
    <input type="text" placeholder="Name" id="form_author"
      value={this.state.author} onChange={this.handleNameChange} />

  </p>
  <p><input type="submit" value="Create" onClick={this.handleClick} /></p>
</form>
```

Harjoitus 10.07: Sanontalomake, lisätehtävä

1. Poista sanontoja - ei lomakkeen avulla

1. Muokkaa Quote komponenttia siten, että siinä on linkki jolla poistaa kyseinen sanonta
2. Luo delete callback QuoteBox-luokkaan, ja välitä se QuoteListin kautta jokaiselle Quote-komponentille. Callback ottaa parametrikseen poistettavan sanonnan id:n. Poista id:n perusteella sanonta taulukosta (esim. splice-funktiolla).
3. Quote-komponentissa lisää onClick tapahtuman tapahtumankäsittelijä
4. Tapahtumankäsittelijä kutsuu QuoteBoxista välitettyä callback-funktiota oikealla id:llä (luultavasti `this.props.quote.id` tms.)