Uppgifter - Torsdag 15/9

Här kommer en liten introduktion sedan följer uppgifterna

this

Något som man definitivt uppmärksammar när man jobbar i React är att man väldigt ofta använder nyckelordet `this`. Vissa kanske har tänkt på varför man använder det och vissa kanske bara använder det "för det är så man gör i React". Om du var uppmärksam under JavaScript 1 kursen och när vi gick igenom *ES6 Classes* är detta inget nytt. Om du inte förstod varför vi använde just `this` kan det vara en bra idé att återbesöka material och litteratur från JavaScript 1 (t.ex. Eloquent JavaScript).

Med `this` menar vi att vi referar till just den här (this) instansen eller objektet och i Reacts fall: den här komponenten. En omständighet kring just *Components* är att man är tvungen att binda varje funktion till varje komponent manuellt på detta sätt:

```
//I konstruktorn
this.handleClick = this.handleClick.bind(this);
//Via ett click
<button onClick={this.handleClick.bind(this)} />
```

Detta är inte *React*-specifikt. Det är inte ens *ES6*-specifikt utan har funnits redan sen *ES5*:

> ECMAScript 5 (ES5) introduced the Function.prototype.bind method that is used for manipulating context. It returns a new function which is permanently bound to the first argument of bind regardless of how the function is being used. It works by using a closure that is responsible for redirecting the call in the appropriate context.

Källa: **Understanding Scope & Context in JavaScript**

Länkar

Eloquent JavaScript - The Secret Life of Objects

Understanding Scope & Context in JavaScript

PROPS

Källa: Props Overview

Om vi tänker oss att vi vill skicka vidare vårt state till en *Nested Component* så gör vi det genom att ange ett namn på vad vår property ska heta och detta namn som vi anger är det som vi kan komma åt via `this.props`:

```
class App extends React.Component {
    handleClick(){
        //Do Something
    }
    render(){
```

```
        return <Button myCoolClick={this.handleClick} />
    }
}

class Button extends React.Component {
    constructor(props){
        super(props);
    }
    render(){
        return <button onClick={this.props.myCoolClick} />
    }
}
```

Observera att jag själv satte att vår prop skulle heta `myCoolClick` och jag kommer åt värdet i underkomponenten via det namnet, men värdet jag skickar med är är själva namnet på funktionen jag skapade.

defaultProps & propTypes

Källa: Props Overview samt **Props Validation**

Vi kan även sätta ett default value och bestämma exakt vad för sorts värde man får skicka med via `defaultProps` & `propTypes` på detta sätt:

```
class HelloWorld extends React.Component {
    render(){
        return <h1>{this.props.text}</h1>;
    }
}

HelloWorld.defaultProps = {
    text: "Hello World"
}
HelloWorld.propTypes = {
    text: React.PropTypes.string.isRequired
}
```

Vår komponent kommer nu få ett ursprungligt värde i stil med hur vi satte `this.state={}`. Dessutom kommer React att genera ett felmeddelande om vi t.ex. försöker skicka med en siffra istället för en sträng. `.isRequired` är optional, det kan man lägga till på props om man kräver att de ska finnas.

Refs

Källa: Refs

Om man nu har flera komponenter av samma typ men vill enkelt komma åt en av dem, eller komma åt en specifik komponent så kan man använda `this.refs`.

```
class App extends React.Component {
    render(){
        return(
            <div>
                <Button ref='first' />
                <Button ref='second' />
                <Button ref='third' />
            </div>
        )
    }
}
```

För att sedan komma åt t.ex. den första knappen via refs så är det samma syntax som när man använder state och props: `this.refs.first` för att referera till den första `<Button />`-komponenten.

Reusable Components

Källa: Stateless React Components

Alla komponenter behöver dock inte ha ett state, då är det onödigt att vi skapar en ny klass som har ett state varje gång vi skapar en ny komponent. Såkallade **Reusable Components (Stateless Components)** kan då vara bra att använda. De är även mer kompakta än en vanlig reactkomponent:

```
const HelloWorld = () => {
    return <h1>Hello World</h1>
}

//Om vi vill skicka med props
const HelloWorld = (props) => {
    return <h1>{this.props.text}</h1>
}
//Eller den superkompakta
const HelloWorld = () => <h1>HelloWorld</h1>;
```

Uppgifter

Använd skalprojektet som finns i en zip-fil på studentportalen för att lösa följande uppgifter. Strukturen är liknande som den vi gick igenom under onsdagen.

1. Få projektet att fungera. Appen ska ta det som skrivs in i och skriva ut det värdet i . Just nu måste `this.state` skickas ned och användas som `this.props` i underkomponenterna för att det ska fungera.
2. Gör om komponenterna `<FormInput />` och `<TextOutput` till **Stateless Components** och få projektet att fungera med samma funktionalitet som innan.
3. Använd **defaultProps** så att `<TextOutput />` har en text även om man inte har skrivit in något i `<FormInput />`.

4. Använd **defaultProps** för att ge `<FormInput />` ett placeholder-värde i stil med *"Skriv något här"*
5. Validera båda defaultProps med hjälp av **propTypes** så att man de endast får vara strängar och att de måste finnas.
6. Skapa **3** olika `<FormInput />` i `App`.
7. Använd sedan **this.refs** och gör så att texten enbart uppdateras om man skriver i den mellersta `<FormInput />`. Detta kan man uppnå på lite olika sätt men ett sätt att göra det är med metoden `ReactDOM.findDOMNode()`. Parametern man skickar med är det elementet man letar efter i DOMen.

Extrauppgift

Använd `this.state.data` för att skriva ut alla **name** samt **id** för samtliga personer. Lättast gör man detta med t.ex. **data.map** för att loopa igenom varje värde för att sedan lägga till värdet i ett element. Skriv ut detta i en en lista eller en tabell i din komponent. Tips på hur man kan göra finns här: **ReactJS - Keys**