

EMIL, GABRIEL OCH MICKE

C.A.G CONTACTOR 29/11

KOMPETENSDAG I REACT

MÅL MED DAGEN

- ▶ Bekanta oss med React och Redux och förstå deras roll i att skriva en modern webbapplikation.
- ▶ Skapa en förståelse för hur React fungerar och hur det skiljer sig från andra sätt att skriva frontend-kod.
- ▶ Underlätta för dem som exponeras för React/Redux hos kund eller är sugna på att lära sig det ändå.
- ▶ Ha skoj och koda tillsammans!



AGENDA

- ▶ **1. Grunderna i React**
- ▶ **2. Fördjupning i komponenter**
 - ▶ **Hur hanteras State i React-komponenter**
- ▶ **Lunch på kontoret**
- ▶ **3. Redux**
 - ▶ **Introduktion till Redux**
 - ▶ **Hur React och Redux används tillsammans**
- ▶ **Koda vidare på våra applikationer**



GRUNDERNA I REACT

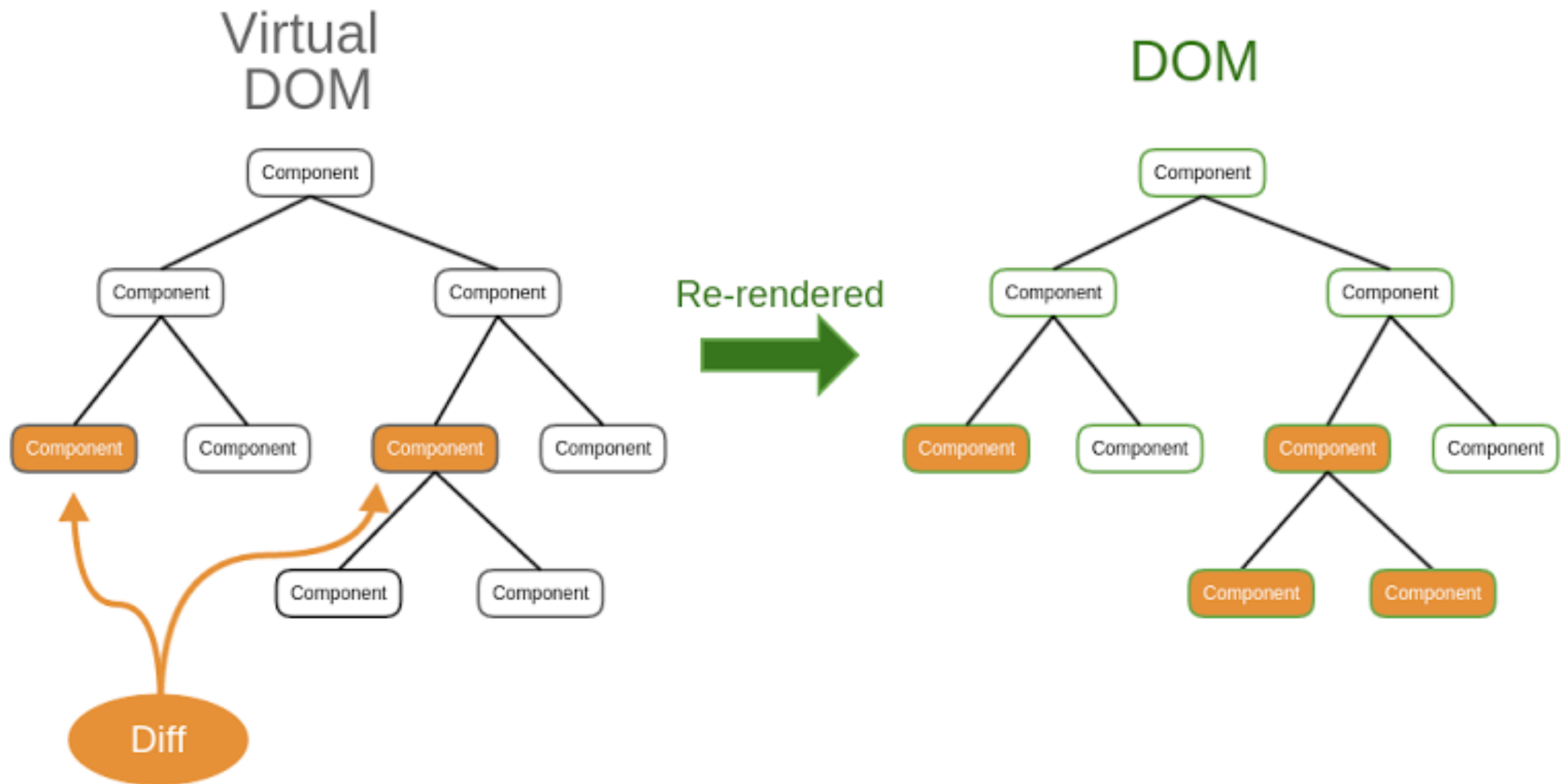
REACT I KORTHET

- ▶ Släpptes först i maj 2013 av Facebook
- ▶ Är huvudsakligen en del av vylagret och används ofta ihop med andra bibliotek, t ex Redux
- ▶ Stödjer rendering både på server-sidan och klient-sidan
- ▶ Vanligt i Single Page Applications
- ▶ Finns även en version för mobila plattformar, React Native

NYCKELKONCEPT

- ▶ Återanvändbara komponenter
- ▶ Virtuellt DOM
- ▶ JSX
- ▶ Envägs databindningar via props
- ▶ Lifecycle-metoder

VIRTUELL DOM



JSX & KOMPONENTSYNTAX

JSX ÖVERSIKT

- ▶ JSX tillhandahåller en HTML-lik syntax
- ▶ JSX kompileras till Javascript
- ▶ React fungerar utan JSX, men det används nästan alltid
- ▶ JSX skyddar mot "injection attack" genom att escapa värden

```
import React from "react"
```

```
class HelloWorld extends React.Component {
  render() {
    return React.createElement("div",
                                { className: "hey" },
                                "Hello World!");
  }
}
```

```
import React from "react"

class HelloWorld extends React.Component {
  render() {
    return (
      <div className="hey">Hello World!</div>
    )
  }
}
```

```
import React from "react"
```

```
function HelloWorld() {  
  return (  
    <div className="hey">Hello World!</div>  
  )  
}
```

```
import React from "react"
```

```
const HelloWorld = () => (  
  <div className="hey">Hello World!</div>  
)
```

```
import React from "react"
```

```
// Klassskomponent
```

```
class HelloWorld extends React.Component {  
  render() {  
    return (  
      <div>Hello World!</div>  
    )  
  }  
}
```

```
// Funktionskomponent
```

```
const HelloWorld = () => (  
  <div>Hello World!</div>  
)
```

EXEMPEL – JSX KOMPILERAS TILL JAVASCRIPT

```
// Det här, som är skrivet i JSX  
<Hello large color="darkgray">  
  Hello, World!  
</Hello>
```

```
// ...kompileras till det här:  
React.createElement(  
  Hello,  
  {large: true, color: 'darkgray'},  
  'Hello, World!'  
)
```



LABB 1

SKAPA EN REACT APP

LABB 1: SKAPA EN ENKEL REACT-APP

- ▶ `npx create-react-app <name>`
- ▶ `npm start`
- ▶ kom ihåg att exportera komponenter som ska komma åt från andra filer
- ▶ props läses från `this.props`
- ▶ `src/index.js` är startpunkten i appen



KOMPONENTER

STATEFULLA KOMPONENTER

- ▶ En komponent kan hålla state
- ▶ Kräver klassbaserade komponenter
- ▶ State initieras när objektet skapas
- ▶ Uppdateras via `setState` och läses från `this.state`
- ▶ Bra att komma ihåg, `setState` är asynkron

```
import React from "react"

export class StatefulComponent extends
React.Component {
  state = {
    name: "C.A.G"
  }

  onClick = () => {
    this.setState({ name: "C.A.G Contactor" })
  }

  render() {
    return (
      <div onClick={this.onClick}>Hello
{this.state.name}</div>
    )
  }
}
```

STATE VS. PROPS

- ▶ State är intern till komponenten
- ▶ Props skickas in i komponenten från föräldern
- ▶ Komponenter kan bara skicka data till sina barn genom properties

KOMPONENT LIVSCYKEL-METODER

- ▶ `render`
- ▶ `componentDidMount/componentWillUnmount`
 - ▶ Starta och stoppa eventuella asynkrona operationer
- ▶ `shouldComponentUpdate/componentDidUpdate`
 - ▶ Hantera nya state och props. Tex skicka nya requests för mer data
- ▶ *`getDerivedStateFromProps`*
 - ▶ Uppdatera state baserat på nya props *innan* `render()`
- ▶ Deprecated: *`componentWillMount`, `componentWillReceiveProps`, `componentWillUpdate`*

DELA UPP KOMPONENTER: CONTAINER VS PRESENTATIONS

- ▶ Single purpose
- ▶ Presentations-komponenter fokuserar på hur saker ser ut
- ▶ Container-komponenter fokuserar på hur saker fungerar, hur data hämtas, skickar data till presentations-komponenter via props
- ▶ Separation av ansvar
- ▶ Mer återanvändbara komponenter

[HTTPS://MEDIUM.COM/@DAN_ABRAMOV/SMART-AND-DUMB-COMPONENTS-7CA2F9A7C7D0](https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0)

```
import React from "react"

export class Clock extends React.Component {
  state = {
    date: new Date()
  }

  tick = () => {
    this.setState({
      date: new Date()
    })
  }

  componentDidMount = () => {
    this.timerId = setInterval(this.tick, 1000)
  }

  componentWillUnmount = () => {
    clearInterval(this.timerId)
  }

  render() {
    return (
      <div>Time is: {this.state.date.toLocaleTimeString()}</div>
    )
  }
}
```



```
import React from "react"
```

Presentation

```
const ClockFace = props => (  
  <div>Time is: {props.time}</div>  
)
```

Container

```
export class Clock extends React.Component {  
  state = {  
    date: new Date()  
  }
```

```
  tick = () => {  
    this.setState({  
      date: new Date()  
    })  
  }
```

```
  componentDidMount = () => {  
    this.timerId = setInterval(this.tick, 1000)  
  }
```

```
  componentWillUnmount = () => {  
    clearInterval(this.timerId)  
  }
```

```
  render() {  
    return (  
      <ClockFace time={this.state.date.toLocaleTimeString()} />  
    )  
  }  
}
```

HÖGRE ORDNINGENS KOMPONENTER (HIGHER ORDER COMPONENTS)

- ▶ Komponenter som wrapper andra komponenter
- ▶ Väldigt populärt pattern i React-världen
- ▶ Används för att förhöja komponenter med mer funktionalitet.
- ▶ Ett vanligt pattern är att namnge dem `with`, tex `withClock(ClockFace)`
- ▶ composable

```
import React from "react"
import { Spinner } from "../Spinner"

// Higher Order Component
function withDataLoader(WrappedComponent) {
  return class DataLoader extends React.Component {

    state = {
      data: undefined
    }

    componentDidMount = () => {
      setTimeout(() => this.setState({data: "Hello"}), 2000)
    }

    render() {
      if (!this.state.data) {
        return <Spinner />
      } else {
        return <WrappedComponent data={this.state.data} />
      }
    }
  }
}

const Component = props => (
  <div>{props.data}</div>
)

export const ComponentWithData = withDataLoader(Component)
```



LABB 2

**BYGG UT MED
STATEHANTERING**

LABB 2: BYGG EN COUNTER MED + OCH - KNAPPAR

- ▶ Skapa en fil med valfritt namn
 - ▶ Skapa en variabel i state som håller reda på counter värdet
 - ▶ Skapa två funktioner, en som ökar och en som minskar värdet på counter
- ▶ Använd `this.setState()` för att ändra värdet på counter
- ▶ Anropa respektive funktion när man klickar på respektive knapp
- ▶ Importera och lägg till Counter i App.js



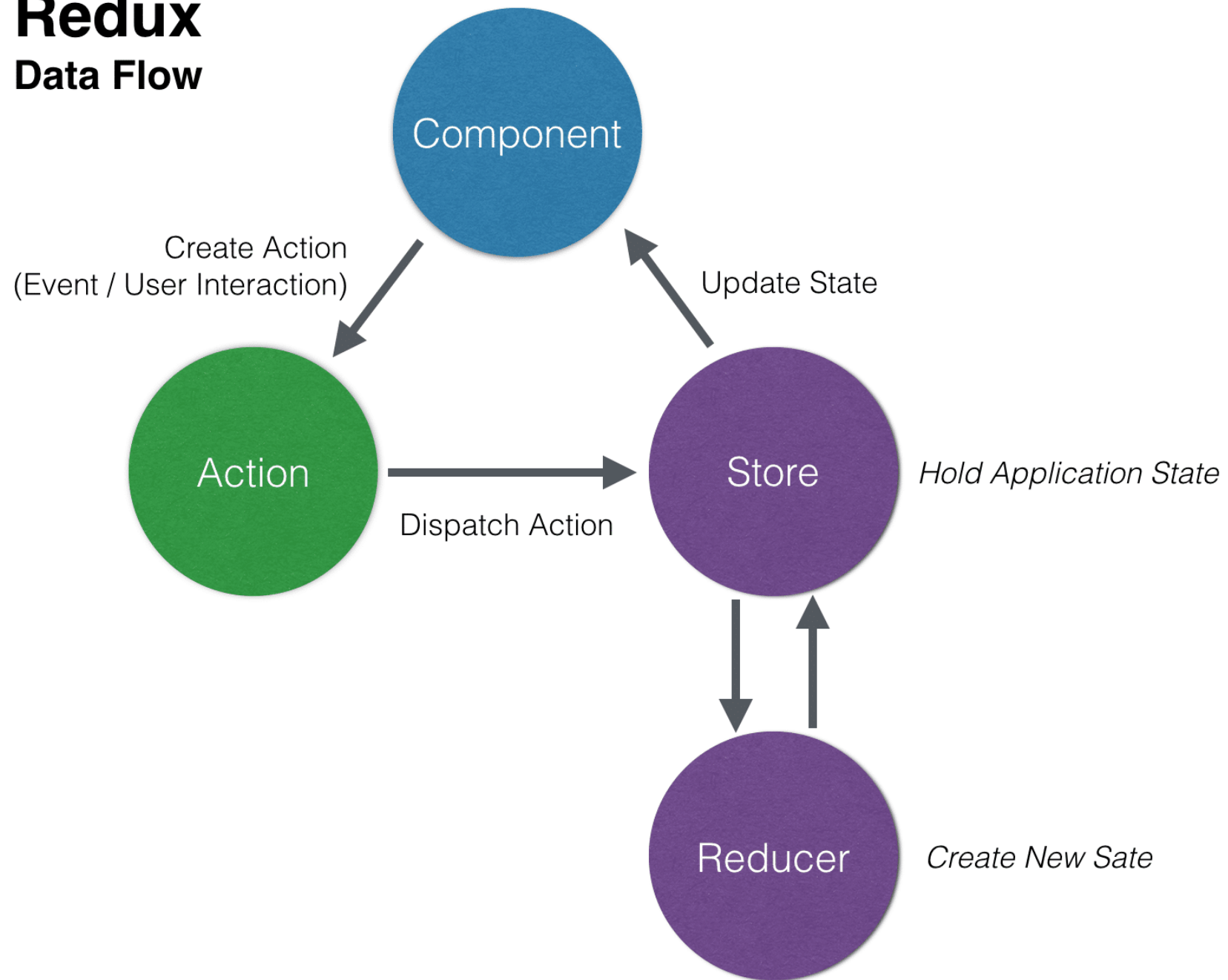
REDUX

BAKGRUND TILL REDUX

- ▶ En implementation av ett koncept som kallas Flux
- ▶ Inspirerats av Elm
- ▶ Kan användas utan React
- ▶ Envägs dataflöde
- ▶ Store håller hela applikations state
- ▶ Reducers uppdaterar state
- ▶ Actions skickas från komponenterna för att beskriva uppgifter som ska utföras

Redux

Data Flow



HUR KOMPONENTERNA KOMMUNICERAR MED REDUX

- ▶ Kopplas ihop med React via react-redux
- ▶ `Provider` är en komponent som gör Redux-storen tillgänglig för applikationen
- ▶ `connect` är en Higher Order Component som kopplar ihop komponenten med Redux
- ▶ `dispatch` används för att skicka *Actions* till Redux

SKAPA ACTION OCH REDUCER

```
const INCREASE_COUNTER = "INCREASE_COUNTER";

export const increaseCounterAction = {
  type: INCREASE_COUNTER
};

const initialState = {
  counter: 0
};

export function updateState(oldState = initialState, action) {
  switch (action.type) {
    case INCREASE_COUNTER:
      return {
        ...oldState,
        counter: oldState.counter + 1
      };
    default:
      return oldState;
  }
}
```

SKAPA CONNECT

- ▶ Ansluta komponenter till Redux Store
 - ▶ `connect(mapStateToProps , mapDispatchToProps)`
- ▶ Läsa ut data från Redux Store
 - ▶ `mapStateToProps`

```
import React from "react";
import { connect } from "react-redux";
import { increaseCounterAction, decreaseCounterAction } from "../state";

const CounterFunc = ({ counterValue }) => (
  <div className="counter">
    <div>
      Counter value: {counterValue}
    </div>
  </div>
);

function mapStateToProps(state) {
  return {
    counterValue: state.counter
  };
}

export const Counter = connect(
  mapStateToProps
)(CounterFunc);
```

SKAPA CONNECT

- ▶ Göra förändringar i Redux Store
 - ▶ `mapDispatchToProps`

```
import React from "react";
import { connect } from "react-redux";
import { increaseCounterAction } from "../state";

const CounterFunc = ({ counterValue, increaseCounter }) => (
  <div className="counter">
    <div>
      <button onClick={increaseCounter}>+</button>
      Counter value: {counterValue}
    </div>
  </div>
);

function mapStateToProps(state) {
  return {
    counterValue: state.counter
  };
}

function mapDispatchToProps(dispatch) {
  return {
    increaseCounter: () => dispatch(increaseCounterAction)
  };
}

export const Counter = connect(
  mapStateToProps,
  mapDispatchToProps
)(CounterFunc);
```



LABB 3

BYGG UT MED REDUX

LABB 3: INFÖRA REDUX

- ▶ Lägg till dependencies
`npm install redux react-redux`
- ▶ Skapa två actions och en reducer i `state.js`
- ▶ Skapa en Store med reduceren som argument
- ▶ Wrappa `<App />` med en `<Provider>`
- ▶ Använd `{connect}` i Counter-komponenten
 - ▶ `mapStateToProps` + `mapDispatchToProps`
 - ▶ Byt ut `this.state` till `this.props`


TEXT

▶ Devtools (React, Redux)

TO DO LIST

☒ Too Many To-Do's

☐ How We're Making Lists

☐ Too Much Time 

☒ ? Unknowns & Changes

LABB 4

EN TODO-LISTA I REACT/REDUX

LABB 4: EN TODO-LISTA I REACT/REDUX

- ▶ Skapa actions för de (state) funktioner som ska finnas
Exempel: add, toggle, setViewFilter
- ▶ Skapa en reducer som uppdaterar state utifrån ovan actions
- ▶ Skapa de komponenter som behövs
Exempel: TodoList, TodoItem, AddTodoForm

SAMMANFATTNING REACT

- ▶ React är en del av vylagret
 - ▶ Virtuellt DOM
 - ▶ Statefulla komponenter
 - ▶ Reacts livscykel-metoder ger kontroll över hur de uppdateras
 - ▶ JSX underlättar render-funktionen
 - ▶ State vs. Props
 - ▶ Container vs Presentations-komponenter
 - ▶ Higher Order Components tillhandahåller composition

SAMMANFATTNING REDUX

- ▶ Tillhandahåller state-hantering till React (och andra ramverk)
- ▶ En *Store* håller i state och tar emot *Actions*
- ▶ *Reducers* används för att uppdatera state baserat på *Actions*
- ▶ *Actions* beskriver handlingar som ska utföras
- ▶ React och Redux kopplas ihop via React-Redux som tillhandahåller `createStore()`, `Provider` samt `connect()`.