React.js Training

Rahman Usta

RUsta@luxoft.com

Luxoft
A DXC Technology Company

React.js

A JavaScript library to create re-usable UI components. It stands as V in MVC pattern.

https://facebook.github.io/react/

Who uses

- Facebook
- Instagram
- Airbnb
- Atlassian
- BBC
- Paypal

See. https://github.com/facebook/react/wiki/Sites-Using-React

Getting started

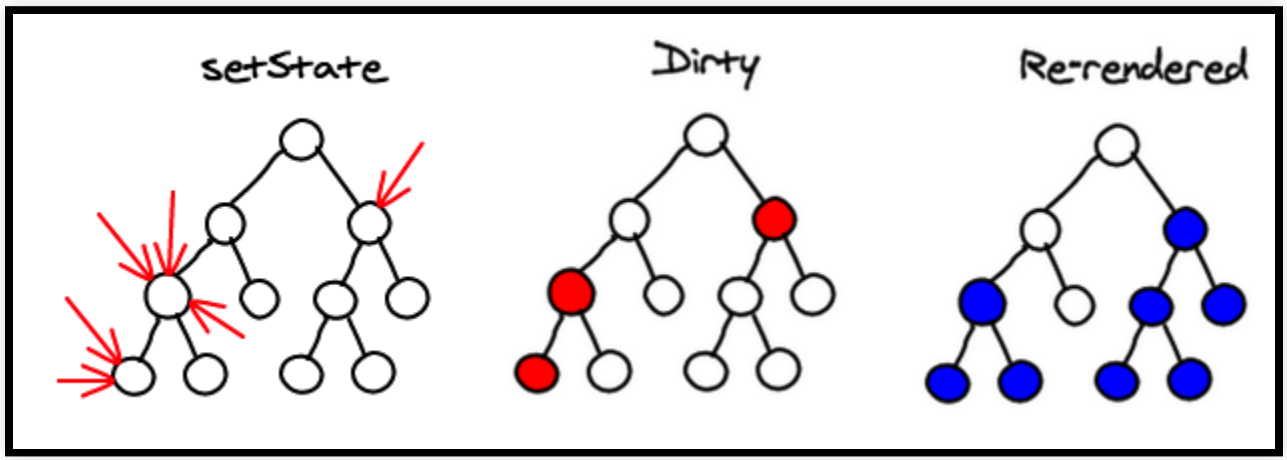
In React we develop view components, and manage them. Required functions are exported over the object named `React`.

Virtual DOM

DOM operations are costly. To reduce this cost, React uses Virtual DOMs.

React keeps a Virtual DOM element for per real DOM element.

DOM diff



Advantages of Virtual DOM

- DOM diff
  - Don’t re-create elements if not required
    - Mount & Unmount
  - Re-use elements as much as possible
    - Change only attributes in matching elements
- Batch update, and delete
  - React may process DOM changes in batch for optimization

React's heuristic diffing algorithm

Assumptions:

1) Two elements of different types will produce different trees.

2) The developer can hint at which child elements may be stable across different renders with a `key` prop.

Elements Of Different Types

```
<div>
  <Counter />
</div>

<span>
  <Counter />
</span>
```

DOM Elements Of The Same Type#1

```
<div className="before" title="stuff" />

<div className="after" title="stuff" />
```

DOM Elements Of The Same Type#2

```
<div style={{color: 'red', fontWeight: 'bold'}} />

<div style={{color: 'green', fontWeight: 'bold'}} />
```

Component Elements Of The Same Type

```
<PricePicker price={99}/>

<PricePicker price={199}/>
```

Recursing On Children#1

```
<ul>
  <li>first</li>
  <li>second</li>
</ul>

<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

Recursing On Children#1

```
<ul>
  <li>Duke</li>
  <li>Villanova</li>
</ul>

<ul>
  <li>Connecticut</li>
  <li>Duke</li>
  <li>Villanova</li>
</ul>
```

Keys#1

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>

<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

Keys#2

- You can use Array's index, if you don't order elements
- Never use Math.random() since it triggers performance degradation.

React.js Core Dependencies

You just require to have `react` and `babel` scripts in your HTML file

React's core dependencies

```
npm init
npm install --save react react-dom
```

React.js in Browser

```
<script src="node_modules/react/umd/react.development.js"></script>
<script src="node_modules/react-dom/umd/react-dom.development.js"></scrip
```

React's createElement

It is a core method to create virtual React elements.

```
React.createElement("<ElTag>",
{<Properties>}, …<ChildComponents>);
```

```
React.createElement("span", {id:"skill"}, "React.js");
```

A simple React.js element

elem-001/index.js

```
// A h1 virtual DOM element
var virtualH1 = React.createElement("h1");

// A p#paragraf virtual DOM element
var virtualP = React.createElement("p", {id: "paragraf"});

// A div>h1+p virtual DOM element
var virtualDiv = React.createElement("div", null, virtualH1, virtualP);

var container = document.querySelector("#placeholder");
ReactDOM.render(virtualDiv, container);
```

React Components

React components wrap other React or HTML components to create reusable structures. React components can keep internal state, can communicate with other components.

- React.Component (Stateful)
  - React.PureComponent (Stateful, Shallow compare)
- Functional Component (Stateless, Stateful with Hooks)
- React.createClass (Stateful, **Deprecated**)

A new React.Component

nojsx/index.js

```
class HelloWorld extends React.Component {
    render() {
        return React.createElement("div", null,
            React.createElement("h3", null, "Hello React"),
            React.createElement("p", null, "Hello World")
        );
    }
}

ReactDOM.render(<HelloWorld/>, document.querySelector("#placeholder"));
```

JSX

JSX is a XML like syntax extension to EcmaScript to define React components inside JavaScript.

JSX ⇒ (Transpilers(Babel.js)) ⇒ JS

See https://facebook.github.io/jsx and https://babeljs.io

Babel.js Dependency

```
npm install --save @babel/standalone
```

Babel.js in Browser

```
<script src="node_modules/@babel/standalone/babel.js"></script>

<script type="text/babel">
    // babel code
</script>
```

A New React.Component (with JSX)

intro-001 / index.js

```
class HelloWorld extends React.Component {

    render() {
        return (<div>
            <h3>This is a title!</h3>
            <p>Hello World</p>
            <img src="../images/earth.jpg" alt="Earth"/>
        </div>);
    }
}

ReactDOM.render(<HelloWorld/>, document.querySelector("#placeholder"));
```

A New Functional Component

functional-comp-001/index.js

```
function HelloWorld() {

    return (<div>
        <h3>This is a title!</h3>
        <p>Hello World</p>
        <img src="../images/earth.jpg" alt="Earth"/>
    </div>);
}

ReactDOM.render(<HelloWorld/>, document.querySelector("#placeholder"));
```

JSX Rules 1

Used component tags have to be closed

```
<Element></Element> // okay
<Element/> // okay

<Element> // not okay
```

JSX Rules 2

Components have to be wrapped or be used Fragments

```
// okay
(<Element>
    <SubElement1/>
    <SubElement2/>
</Element>)

// not okay
(<Element1/>
<Element2/>)
```

Fragments be discussed later.

JSX Rules 3

JSX components may include HTML attributes, or non HTML attributes called `props`

Few Exceptions for reserved keyword

class ⇒ className

for ⇒ htmlFor

```
(<div id="luxoft">
   <h1 className="header">Hello Luxoft</h1>
   <p>
     <a href="https://luxoft.com">Luxoft.com</a>
   </p>
</div>)
```

JSX Rules 4

A child element can be

- Plain text
- A React component
- A HTML element
- [] form of the above
- <></> A fragment

JSX Rules 5

React tags have to be started in **Capital case**. HTML elements should be used in **Lower case**.

A new Component

Any JS value, even React components can be assigned to a variable. Variables can be used in template between { }

intro-002/index.js

```
class HelloWorld extends React.Component {
    render() {

        var clazz = "hello";
        var content = (<strong>Hello world!</strong>);

        return (<p id="hello" className={clazz}>
            {content}
        </p>);
    }
}
ReactDOM.render(<HelloWorld/>, document.querySelector("#placeholder"));
```

Composition in React Components

intro-003/index.js

```
class Name extends React.Component {
    render() {
        return (<strong>Rahman</strong>);
    }
}

class Surname extends React.Component {
    render() {
        return (<u>Usta</u>);
    }
}

class FullName extends React.Component {
    render() {
        return (<p>
            <Name/>
            {" "}
            <Surname/>
```

Composition in Functional Component

intro-004/index.js

```
const Name = () => <strong>Rahman</strong>;

const Surname = () => <u>Usta</u>;

const FullName = () => (<p>
    <Name/>
    {" "}
    <Surname/>
</p>)

ReactDOM.render(<FullName/>, document.querySelector("#placeholder"));
```

When Components Rendered?

- When internal state is changed
  - From User action, side-effects, timers
- When new props are received
  - From parent component.

Props Are (Stateless data)

- Data which can be passed to React components (parent to child)
  - Data could be:
    - Simple values, objects, functions, or other React components
- Just read-only values passed from top to down.
- React props can be accessed via special property called `props`.

Assigning Props

`props` can be assigned as String `""` or in other types with `{}`.

Accessing Props in Class Components

Props can be accessed in Class components via `this.props` expression

props-001/index.js

```
class StrongElement extends React.Component {
    render() {
        return (<strong>{this.props.content}</strong>);
    }
}

ReactDOM.render(<StrongElement content="Hello World"/>,
    document.querySelectorAll(".placeholder")[0]);

var content = "Hello World";

ReactDOM.render(<StrongElement content={content}/>,
    document.querySelectorAll(".placeholder")[1]);
```

Accessing Props in Functional Components

Props can be accessed in Functional components via `props` function argument

props-002/index.js

```
function StrongElement(props) {
    return (<strong>{props.content}</strong>);
}

ReactDOM.render(<StrongElement content="Hello World"/>,
    document.querySelectorAll(".placeholder")[0]);

var content = "Hello World";

ReactDOM.render(<StrongElement content={content}/>,
    document.querySelectorAll(".placeholder")[1]);
```

Conditional Rendering

props-003/index.js

```
function Conditional(props) {

    if (props.message) {
        return (<h1>{props.message}</h1>);
    } else {
        return <h1>Message not found</h1>;
    }

}

ReactDOM.render(<Conditional/>,
    document.querySelector("#placeholder"));
```

Rendering if Matches

props-004/index.js

```
function Conditional({message}) {

    return (<div>
        <h1>Prints if message exist</h1>
        {message && <p>You have a new message! => {message}</p>}
    </div>)

}

ReactDOM.render(<Conditional message={"Your message"}/>,
    document.querySelector("#placeholder"));
```

Default Props

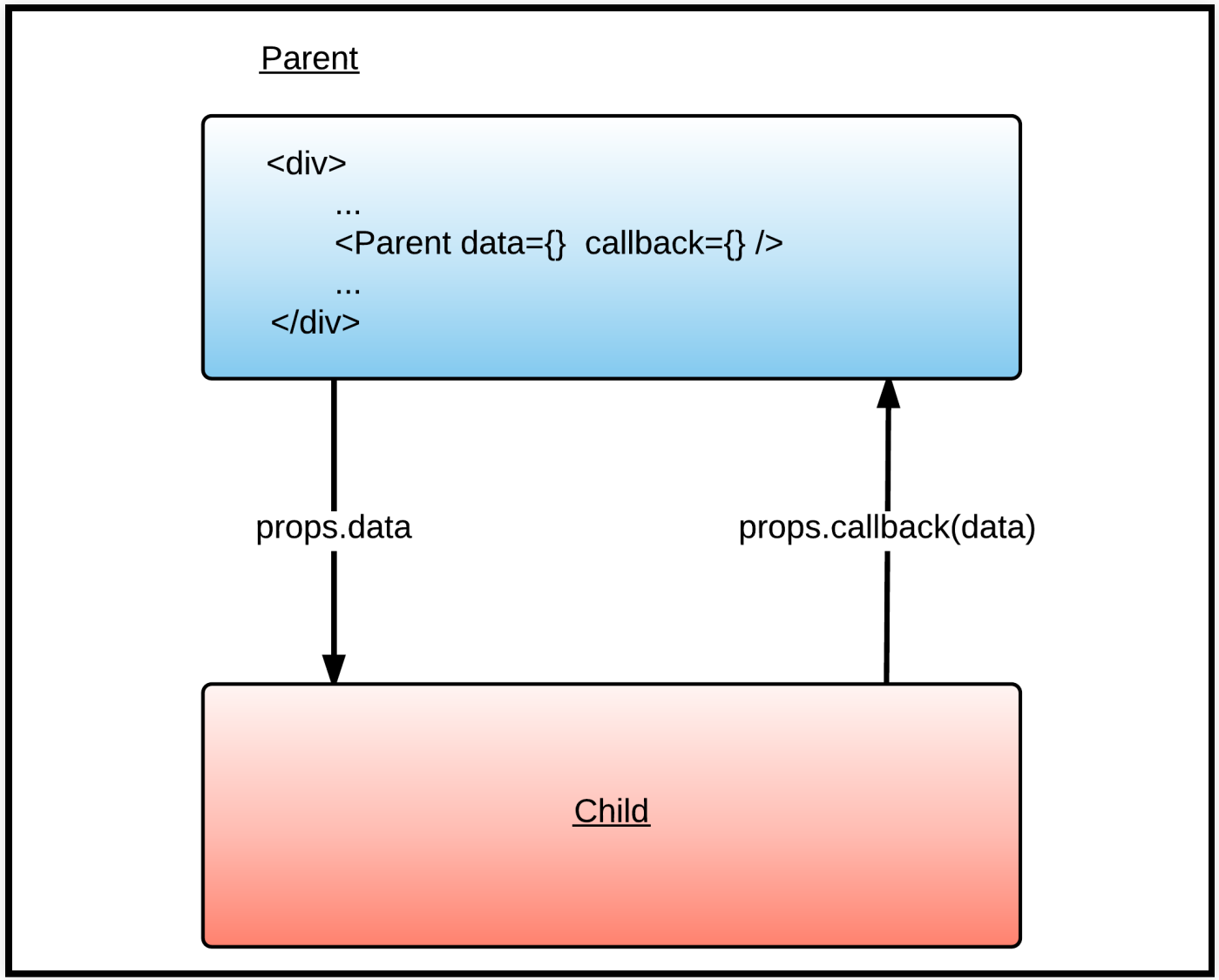
We can define default props to any React component just using the `defaultProps` property.

default-props-001/index.js

```
class MoneyText extends React.Component {

    render() {
        return (<div>
             {this.props.symbol} {this.props.value}
        </div>);
    }
}

MoneyText.defaultProps = {
    symbol: "$"
}

ReactDOM.render(<MoneyText value="100"/>,
document.querySelector("#placeholder"));
```

Passing data between components

Data goes down, Action goes up


Parent

...
<Parent data={} callback={} />
...

props.data
props.callback(data)
Child

State (Stateful data)

- State refers to internal state of a component
- Using state is not mandatory for every component
- Component state can be initialized
- State is kept until it is changed
- State can be changed during the lifecycle a component
- State change triggers re-**render**ing of a component

Initial State of a Component

Initial state can be assigned in the constructor of a React class component.

```
class Counter extends React.Component{
    constructor(props){
        super(props)
        this.state= { value: 0 };
    }

}
```

State 1

state-001/index.js

```
class Counter extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            value: 0
        };
    }

    _updateValue() {
        this.setState({
            value: this.state.value + 1
        });
    }

    componentDidMount() {
        setInterval(() => this._updateValue(), 1000);
    }
```

State 3

state-003/index.js

```
class ProgressBar extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            position: 0
        };
    }

    _proceed() {

        let newState = {};
        let currentPosition = this.state.position;
        if (currentPosition >= 100) {
            newState.position = 0;
        }
        else {
            newState.position = currentPosition + this.props.step;
```

References

Sometimes we may require to access real DOM element. In that case we can use `refs` .

refs-001/index.js

```
class Merhaba extends React.Component {
    componentDidMount() {
        console.log(this.myNode.innerText);
    }

    render() {
        return (<strong ref={e => this.myNode = e}>
            Hello world
        </strong>);
    }
}

ReactDOM.render(<Merhaba/>, document.querySelector("#placeholder"));
```

Events Handling#1

Event	React
onclick	onClick
onkeypress	onKeypress
ondblclick	onDblclick

See. https://reactjs.org/docs/events.html#supported-events

SyntheticEvent

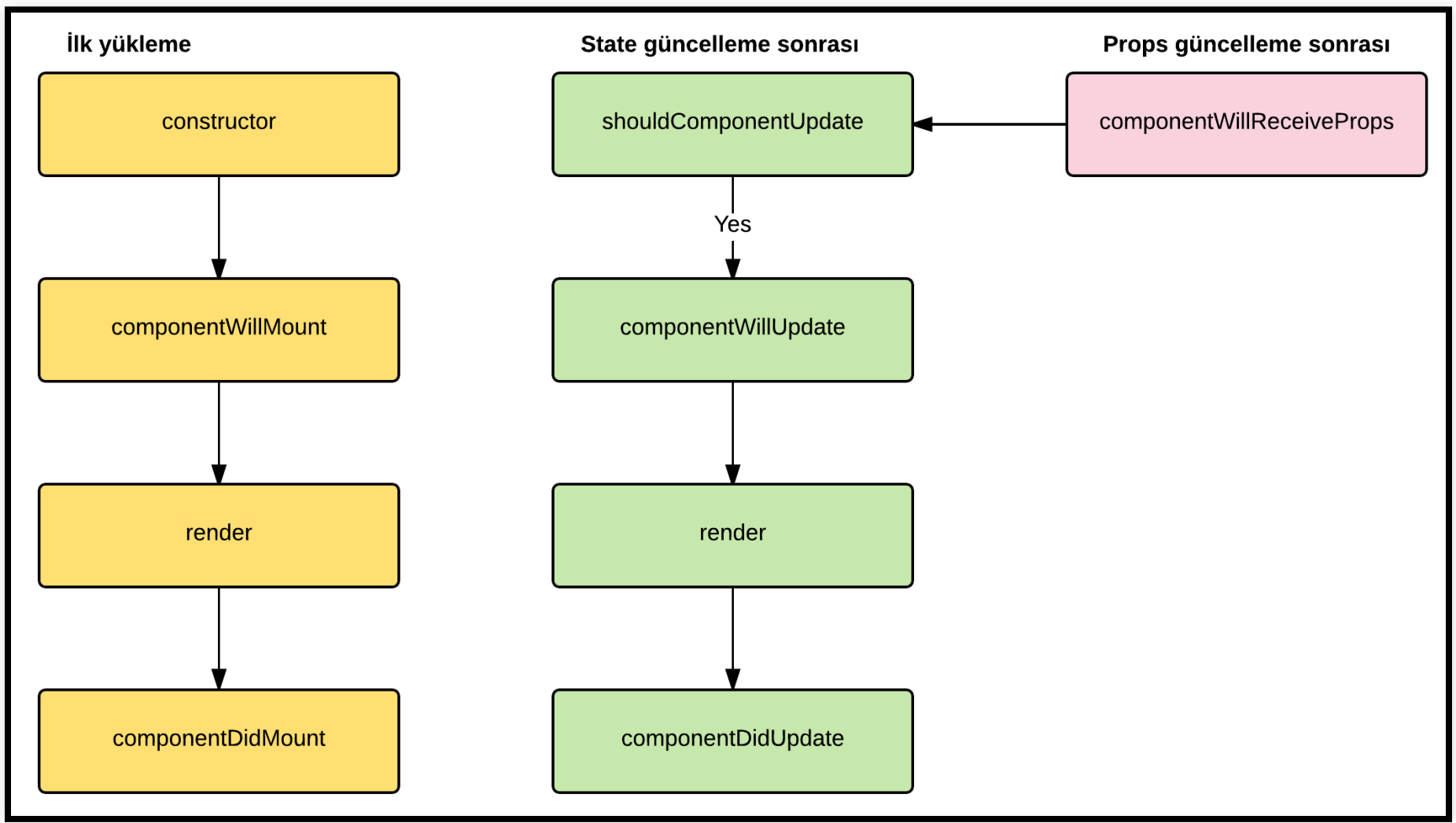
Your event handlers will be passed instances of SyntheticEvent, a cross-browser wrapper around the browser's native event.

Events Handling#2

event-001/index.js

```
class EventComp extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            counter: 0,
            keypressValue: "",
            keyenterValue: ""
        };
    }

    _onClick() {
        this.setState({
            counter: this.state.counter + 1
        });
    }

    _onKeyUp(event) {
```

React Component's Lifecycle#1



React Component's Lifecycle#2

Phase	Description
componentWillMount	Called just before a React component is bound to real DOM.
render	Called during first `render` or after `state` update.
componentDidMount	Called just after a React component was bound to real DOM.

React Component's Lifecycle#3

Phase	Description
shouldComponentUpdate	It is a checkpoint to skip possible re-render
componentWillUpdate	Called just before re-rendering.
componentDidUpdate	Called just after re-rendering.

Should it be re-rendered?

Bir React bileşeni `render` edilmeden önce bu soruyu sorar.

```
shouldComponentUpdate: function (nextProps, nextState) {
    return this.state.abc !== nextState.abc;
}
```

Props validation 1

If you design a component, you may want to force its users regarding data types of the `props`. For that purpose the `prop-types` project can be used.

Props validation 2

validation-001/index.js

```
class Person extends React.Component {
    render() {
        return (<div>
            <p><strong>{this.props.name}</strong></p>
            <p><strong>{this.props.age}</strong></p>
        </div>);
    }
}

Person.propTypes = {
    name: PropTypes.string,
    age: PropTypes.number.isRequired,
    fullName: PropTypes.func
};

ReactDOM.render(<Person name="Rahman Usta" age={29} fullName={console.lo
document.querySelector("#placeholder"));
```

Styling in React

style-001/index.js

```
var boxStyle = {
    color: "blue",
    padding: "5",
    border: "1px solid",
    borderRadius: "5",
    height: "100",
    width: "150"
};

const HelloWorld = () => {
    return (<div style={boxStyle} className="box">
        <strong>Hello World</strong>
    </div>);
}

ReactDOM.render(<HelloWorld/>,
    document.querySelector("#placeholder"));
```

https://github.com/JedWatson/classnames

Dynamic Components

We may want dynamically generate components in React.

iteration-001/index.js

```
const Item = (props) => {
    return (<li style={{color: props.color}}>{props.color}</li>)
}

let colors = ["blue", "red", "pink", "black"];
const Colors = (props) => {
    return colors.map(color => <Item color={color}/>)
}


var container = document.querySelectorAll(".placeholder");
ReactDOM.render(<Colors/>, container[0]);
```

Do you recall Keys section?

We have to provide a unique key for each element.

iteration-002/index.js

```
const Item = (props) => {
    return (<li style={{color: props.color}}>{props.color}</li>)
}

let colors = ["blue", "red", "pink", "black"];
const Colors = (props) => {
    return colors.map(color => <Item key={color} color={color}/>)
}


var container = document.querySelectorAll(".placeholder");
ReactDOM.render(<Colors/>, container[0]);
```

Do you recall the Keys section ?

Returning Multiple Elements?

fragments-001 / index.js

```
function List() {
    return (
        <li>Blue</li>
        <li>Red</li>
        <li>Yellow</li>
    );
}

function UnOrdered() {
    return (<ul>
        <List/>
    </ul>)
}

ReactDOM.render(<UnOrdered/>, document.querySelector("#placeholder"));
```

Fragments

Fragments <></> allows you to wrap sibling elements without using a real DOM element.

fragments-002/index.js

```
function List() {
    return (
        <>
            <li>Blue</li>
            <li>Red</li>
            <li>Yellow</li>
        </>
    );
}

function UnOrdered() {
    return (<ul>
        <List/>
    </ul>)
}

ReactDOM.render(<UnOrdered/>, document.querySelector("#placeholder"));
```

Don’t Mutate Arrays & Objects

Bir React bileşeninin `state` nesnesi değiştiğinde `render` metodu çağrılır. Değişimi anlaması için varolan nesnede değişiklik değil, yeni nesne ile `state` değiştirilmelidir.

Don’t Mutate Array#1

Wrong usage

```
var state = [1, 2, 3, 4];

// change state
state.push(5);
var newState = state; // wrong X
```

Don’t Mutate Array#2

Correct usage

```
var state = [1, 2, 3, 4];

// change state
var newState = state.slice(0); // clone it
newState.push(5);
```

Don’t Mutate Array#3

Alternatively spread syntax can be used

Correct usage

```
var state = [1, 2, 3, 4];

// change state
var newState = [...state, 5]
```

Don’t Mutate Objects#1

Wrong usage

```
var state = { a: "1", b: "2"};

// change state
state.c = "3"
var newState = state; // wrong X
```

Don’t Mutate Objects#2

Correct usage

```
var state = { a: "1", b: "2"};

// change state
var newState = {};
newState.a = state.a;
newState.b = state.b;
newState.c = "3";

// or

//change state
var newState = Object.assign({},state,{c: "3"});
```

Don’t Mutate Objects#3

Alternatively spread syntax can be used

Correct usage

```
var state = { a: "1", b: "2"};

//change state
var newState = {...state, c: 3}
```

ES6 Classes

es6-comp-001/index.js

```
class HelloWorld extends React.Component {

    constructor(props) {
        super(props)
        console.log(arguments);
    }

    render(){
        return (<strong>
            Hello World
        </strong>);
    }
}

ReactDOM.render(<HelloWorld/>, document.querySelector("#placeholder"));
```

ES6 Bind This

es6-comp-002/index.js

```
class HelloWorld extends React.Component {

    constructor(props) {
        super(props)
        // this.printName = this.printName.bind(this)
    }

    printName() {
        alert(this.name.value);
    }

    render() {
        return (<div>
            <input ref={name => this.name = name}
                   type="text" placeholder="Enter your name"/>
            <button onClick={this.printName}>Submit 1</button>

            <button onClick={this.printName.bind(this)}>Submit 2</button>
```

Functional Components

```
const Merhaba = (props) => (
                <h1>Merhaba {props.name}</h1>
            )
```

Create React App

React.js App creator by Facebook

https://github.com/facebook/create-react-app

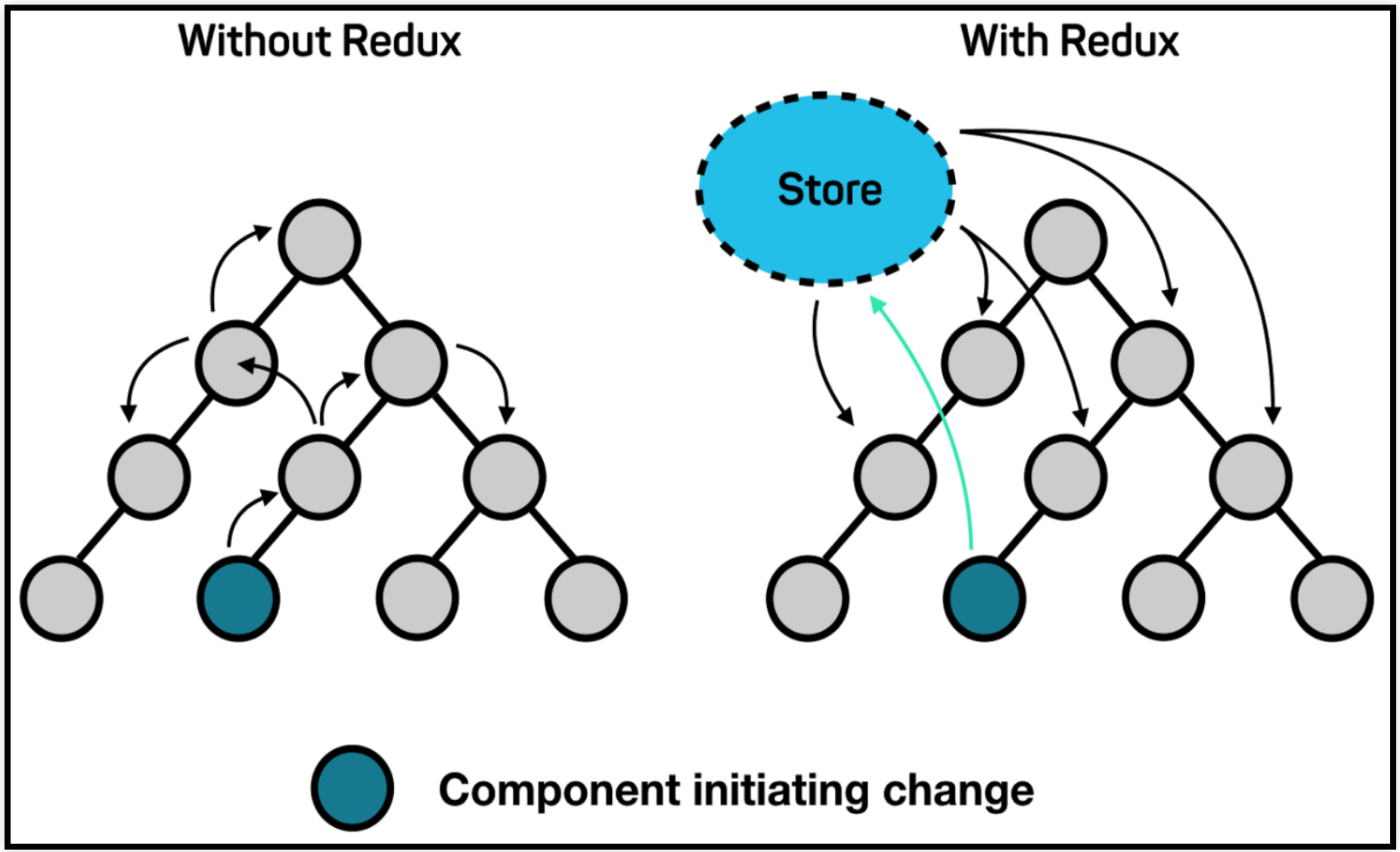
```
npx create-react-app my-app
cd my-app
npm start
```

Redux

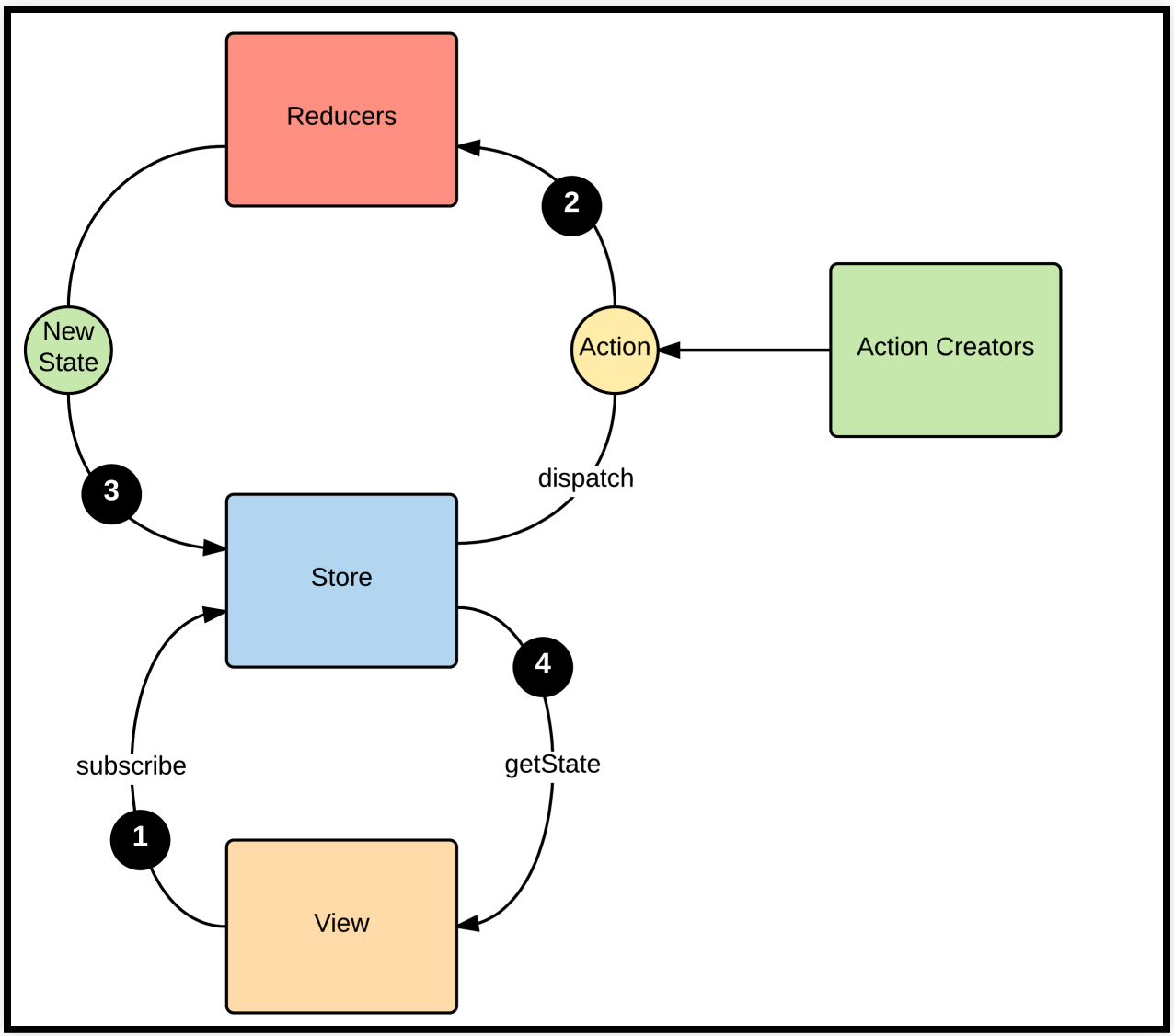
Central `state` manager for React components.

Redux components: Actions, Reducers, Store

Why Redux?


Without Redux
With Redux
Store
Component initiating change

Redux Structure



Actions

Every action triggers a state change. Actions are just plain JS objects. **Action**s are dispatched, and used to decide next state.

```
const SAY_HELLO="SAY_HELLO";

function sayHello(name){
  return {
    type: SAY_HELLO,
    name
  };
}
```

Reducers

Receives dispatched actions, and decides to the next `state`.

```
var defaultState= {
    hello: "Hello World"
};
function helloReducer(state = defaultState, action){
    switch(action.type) {
        case SAY_HELLO:
            return {
                hello: "Hello " + action.name;
            }
        default:
            return state;
    }
}
```

Stores

Stores hold state information. Stores can publish new Action's and subscribe to state changes.

```
var helloStore = createStore(helloReducer);

helloStore.subscribe(()=>{
    let latestState= helloStore.getState();
    // {hello: "Hello Rahman"}
});

helloStore.dispatch(sayHello("Rahman"));
```

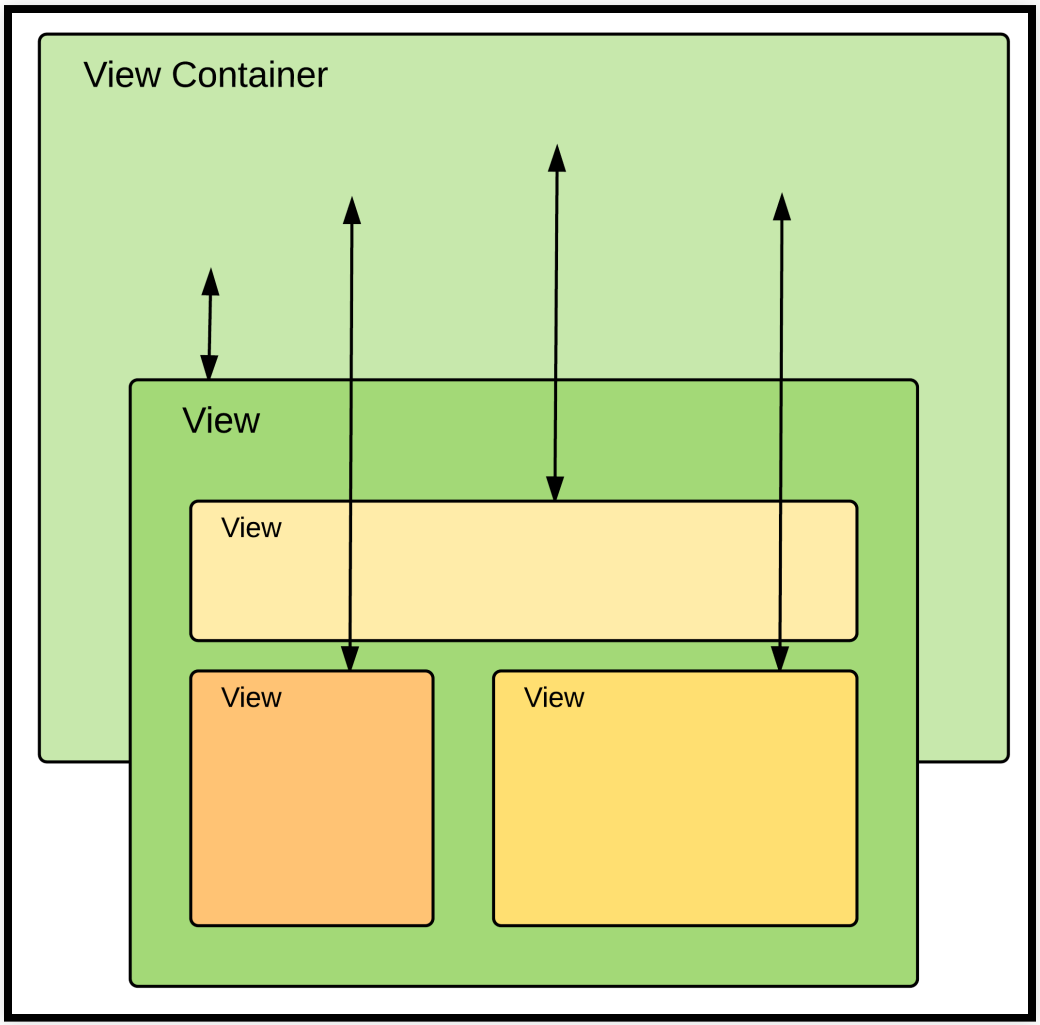
Getting started with Redux

Just install the `redux`, and use the dependency.

```
// add redux dependency
npm install --save redux

// and access to the `createStore` fonksiyon
import { createStore } from "redux";
```

Redux Containers

Redux containers is an extension to React. It brings easy communication inside a view container. == Redux Containers #0


View Container
View
View
View
View

Redux Containers #1

CounterComponent

```
let {increase, counterValue} = this.props;

<div>
    <button onClick={increase}>Counter {counterValue}</button>
</div>
```

Redux Containers #2

CounterContainer

```
import { connect } from 'react-redux'

const mapStateToProps = (state) => {
    return {
        counterValue: state.counter.value
    }
}

const mapDispatchToProps = (dispatch) => {
    return {
        increase: ()=> {
            dispatch(increaseCounter())
        }
    }
}

export default connect(mapStateToProps, mapDispatchToProps)(CounterCompo
```

React Redux Provider

```
export default (<Provider store={appStore}>
    <YourApp/>
</Provider>);
```

Redux's React extension

```
npm install --save react-redux
```

Redux Thunk

```
function sayHello() {
 return function(dispatch, getState){

    dispatch(startHello());

    setTimeout(()=>{
       dispatch(sayHello("Usta"));
    },1000);
 }

}
```

```
npm install --save redux-thunk
```

```
import {createStore,applyMiddleware} from "redux"
import thunk from "redux-thunk"

var store = createStore(counterReducer, applyMiddleware(thunk));
```

Merging reducers

Reducers can be merged together.

```
function aReducer(state, action){

}

function bReducer(state, action){

}

let reducers= combineReducers({
    aReducer, bReducer
);

let store= createStore(reducers);

state => {
    aReducer: {...},
    bReducer: {...}
}
```

```
import { combineReducers } from 'redux'
```

React and Navigations

React.js doesn't have a standard solution for page navigation. `react-router` is a widely used library for navigation purpose.

React Router

React Router can be used in both React Web and React Native projects.

React Router installation

```
npm install react-router-dom --save
```

How to use React Router

```
import {
    BrowserRouter,
    Route,
    Link,
    NavLink,
    Switch
} from "react-router-dom";
```

Router types

React Router supports several router solutios.

- BrowserRouter (Web, HTML 5 History Api)
- HashRouter (Web, Hash param)
- MemoryRouter (Test, React Native)
- NativeRouter (React Native)
- StaticRouter (Server side)

Router Components

- Router
- Switch
- Link
- NavLink
- history, match, withRouter

BrowserRouter

Router components creates a navigation context. Every components inside the router component can use the router features.

```
<BrowserRouter>
    ...
</BrowserRouter>
```

Route

It shows a spesific component for a matching URL path.

```
<BrowserRouter>
  <div>
    <Route exact path="/" component={Home}>
    <Route path="/about" component={About}>
    <Route strict path="/contact/" component={Contact}>
    <Route component={NotFound}>
  </div>
</BrowserRouter>
```

Example 1

path	url	shows
/path	/path	yes
/path	/path/	yes
/path	/path/subpath	yes

Example 2

When exact is `true`

path	url	shows
/path	/path	yes
/path	/path/	yes
/path	/path/subapth	no

Example 3

When strict is `true`

path	url	shows
/path/	/path/	yes
/path/	/path	no

Link

Generates route aware HTML links

```
<Link to={"/"}>Home</Link>
<Link to={"/about"}>About</Link>
<Link to={"/contact"}>Contact</Link>
```

NavLink

Generates route aware HTML links

```
<NavLink to={"/"} activeClassName="active">Home</Link>
<NavLink to={"/about"}>About</Link>
```

Navigation with code

```
class Page extends React.Component{
 //
    goToHome(){
       this.props.history.push("/");
       // push, goBack, go, goForward
    }
 //
}

<Route path="/page" component={Page}>
```

Child Route

```
<BrowserRouter>
  <div>
    <Route exact path="/" component={Home}>
    <Route path="/about" component={(props)=>{
      <h1>Hakkında sayfası</h1>
      <Route path=`${props.match.url}/me` component={Me}/>
      <Route path=`${props.match.url}/you` component={You}/>
    }>
  </div>
</BrowserRouter>
```

Path parameterss

```
<BrowserRouter>
  <Route path="/user/:name" render={(props)=>{
     return (<h1>
            Merhaba {props.match.params.name}
         </h1>)
  }}>
</BrowserRouter>
```

Path conflicts

Which one will be rendered?

```
<BrowserRouter>
  <div>
    <Route path="/user/rahmanusta" ...>
    <Route path="/user/:name" ...>
  </div>
</BrowserRouter>
```

Switch

```
<BrowserRouter>
    <Switch>
      <Route path="/user/rahmanusta" ...>
      <Route path="/user/:name" ...>
    </Switch>
</BrowserRouter>
```

The End

Thank you