React

Prepared by: Aamir Pinger

fb.com/AamirPingerOfficial

linkedin.com/in/AamirPinger

github.com/AamirPinger

React Book

https://softchris.github.io/books/react/

React

- The initial React release was 2013 by Facebook.
- React is a library made over javascript
- In recent years single page applications (SPA) have become popular.
- React is not an SPA framework but a “view” library.
- It is the V in the MVC (Model-View-Controller architectural pattern).

React

- Model-View-Controller (MVC) is an application model comprised of three dependent layers.
  - The model (data)
  - The view (user interface)
  - The controller (processes that handle input).
- React only enables you to render components as viewable elements in a browser.

React

- A Single Page Application (SPA) differs from a normal web application
- In SPA that you remain on the same URL and thereby the same page, hence the name.
- Traditionally in HTML we create multiple HTML files for multiple page website
- In SPA we create one HTML page and create Routes on them to show different pages

Why React

Why React

Because of :

- Its compositional model
- Its declarative rather than imperative
- Unidirectional Data flow/binding
- React is simply javascript

Compositional Model

Compositional model

- Composition is an act or mechanism to combine simple functions to build more complicated ones.
- Why Composition?
- Two things to remember:
  - Simple functions
  - Combination of simple functions to make another function

Compositional model

- Let's look at an a simple JS function example:

```
function getTwitterProfile (username) {
    return 'https://twitter.com/' + username
}
```

It’s a very simple function

Compositional model

- Similarly, the simple getTwitterUserPic function to return url for user twitter profile picture:

```
function getTwitterUserDP (username) {
    return ‘https://avatars.io/twitter/’ + username + ‘/medium'
}
```

Another very simple function

Compositional model

- Let’s combine both functions

```
function getTwitterProfileData (username) {
    return {
        twitterProfile: getTwitterProfile(username),
        profilePic: getTwitterUserDP(username)
    }
}
```

That is composition!

QUESTION ARISES?

Why three function instead of one directly?

Compositional model

What we did

```
function getTwitterProfile (username) {
     return 'https://twitter.com ' + username
}
```

Function 1

```
function getTwitterUserDP (username) {
   return ‘https://avatars.io/twitter/’ + username + ‘/medium'
}
```

Function 2

```
function getTwitterProfileData (username) {
   return {
       twitterProfile: getTwitterProfile(username),
       profilePic: getTwitterUserDP(username)
   }
}
```

Composite Function

Compositional model

How about

```
function getTwitterProfileData (username) {
   return {
     twitterProfile: 'https://twitter.com ' + username,

     profilePic: ‘https://avatars.io/twitter/’ + username + ‘/medium'
   }
}
```

Non composite way

```
function getTwitterProfileData (username) {
   return {
       twitterProfile: getTwitterProfile(username),
       profilePic: getTwitterUserDP(username)
   }
}
```

Composite Function

Compositional model

How about

```
function getTwitterProfileData (username) {
   return {
     twitterProfile: 'https://twitter.com ' + username,

     profilePic: ‘https://avatars.io/twitter/’ + username + ‘/medium'
   }
}
```

Non composite way

- Two separate functions with one composite function is better as it increase reusability.
- There is always a one rule for a good function, a rule of "**DOT**", Do one thing!

React & Composition

- In React we rely on composition, heavily!
- In React we create Components to build different sections of a website
- Components are building blocks in React.
- For example following are three different components:
    - <LandingPage />
    - <AboutUs />
    - <ContactUs />
- Currently they are independent component

React & Composition

- In React, we can have a composite component as simple as

```
<LandingPage>
    <AboutUs />
    <ContactUs />
</LandingPage>
```

- <LandingPage> become parent component and other become child component
- This way we can use individual component blocks to build a big website.

Declarative Nature

Declarative nature

- Most of JavaScript is imperative code.
- We spoon feed each and every step to make javascript aware of how to get desired result.
- Let’s take a example water tank level
    - Manual - (Imperative)
    - Auto - (Declarative)

Declarative nature

Imperative code!

```
const teachers = ['Zia', 'Irfan', 'Muneeb', 'Aamir']
const titles = []

for (let i = 0; i < teachers.length; i++) {
     titles[i] = 'Mr. ' + teachers[i]
}

console.log(titles)
```

RESULT: ['Mr. Zia', 'Mr. Irfan', 'Mr. Muneeb', 'Mr. Aamir']

Declarative nature

Declarative code!

```
const teachers = ['Zia', 'Irfan', 'Muneeb', 'Aamir']

Const titles = teachers.map(name => ‘Mr. ‘ + name )

console.log(titles)
```

RESULT: ['Mr. Zia', 'Mr. Irfan', 'Mr. Muneeb', 'Mr. Aamir']

React - Declarative Nature

In React we will soon going to see declarative code like following

```
<PrintOnBrowser name=’Aamir’ />
```

```
<PrintOnBrowser name=’Aamir Pinger’ />
```

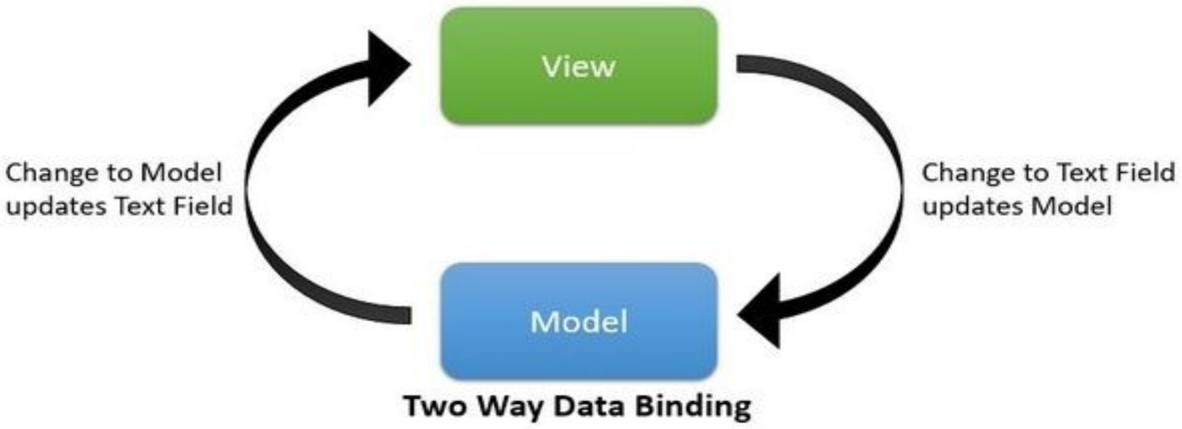
RESULT on Browser:

Aamir

Aamir Pinger

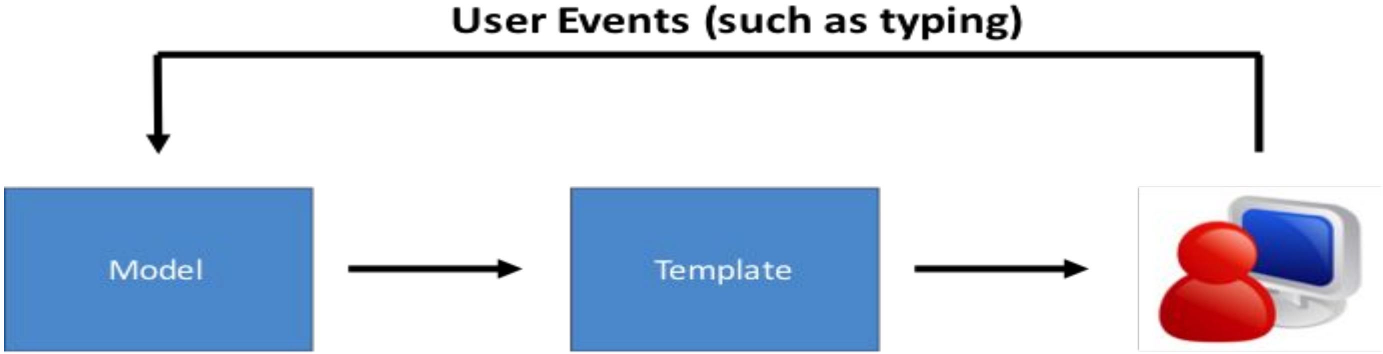
Unidirectional Data Binding

Two way data binding


View
Change to Model updates Text Field
Change to Text Field updates Model
Model
Two Way Data Binding

- Many front-end framework like Angular uses two way data binding.
- Two-way data binding look really great, but when application grows it is hard to determine where the data is actually being updated.

Unidirectional Data Binding



- One-way data binding only propagates changes from the model to the UI, not vice versa.
- It is known as **“single source of truth”**.

Source: https://www.accelebrate.com/blog/two-way-data-binding-angular-2-and-React/

React is simply Javascript

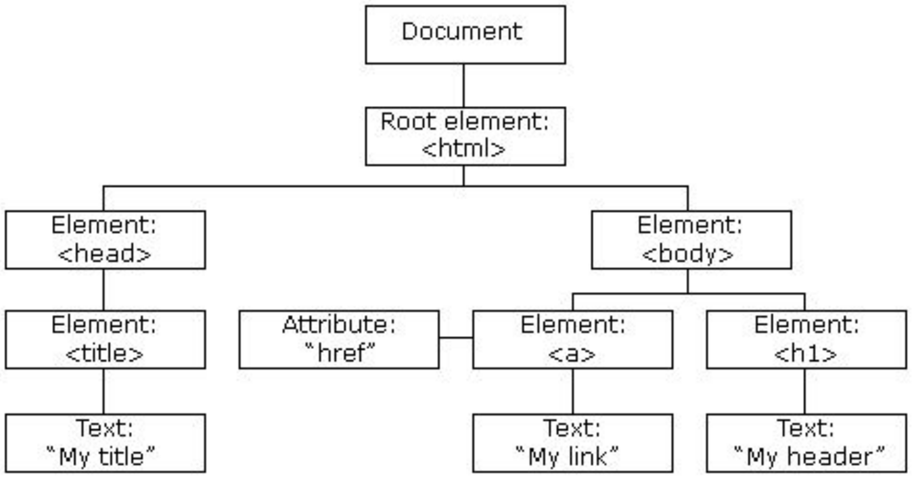
React is simply Javascript

- Uptil now we haven’t seen any code other then javascript code
- React is small library based on Javascript
- Even components in React are JavaScript class or function
- Arrow functions, .map() and .filter() will be seen used extensively in any React code

DOM vs Virtual DOM

DOM

- DOM is Document Object Model
- It’s a programming interface for HTML and XML documents
- When a web page is loaded, the browser creates a Document Object Model of the page.



Document

Root element: <html>

Element: <head>

Element: <body>

Element: <title>

Attribute: "href"

Element: <a>

Element: <h1>

Text: "My title"

Text: "My link"

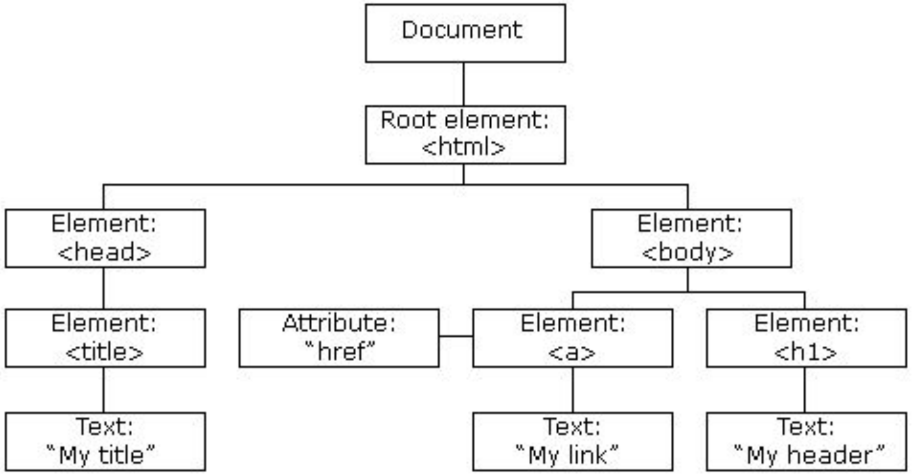
Text: "My header"

DOM Tree

https://www.w3schools.com/js/js_htmldom.asp

DOM

- The DOM is an object-oriented representation of the web page.
- Scripting language such as JavaScript can modify the document structure, style, and content.



DOM Tree

https://www.w3schools.com/js/js_htmldom.asp

Virtual DOM

- React introduced Virtual DOM (VDOM)
- The VDOM is a programming concept where a virtual representation of a UI is kept in memory
- It’s is a tree based on JavaScript objects created with React that resembles a DOM tree
- A process called Reconciliation is used to sync Real DOM with VDOM
- React uses ReactDOM Library updates VDOM and render it on actual DOM

QUESTION ARISES?

Why Virtual DOM is needed?

Why Virtual DOM is needed?

- Making changes in memory (VDOM) is quite faster than updating a complete browser screen (Real DOM)

- React creates first VDOM when application launches and then put everything on browser

Why Virtual DOM is needed?

- Once app need to update browser screen, React creates new updated VDOM

- Through reconciliation process React find out difference between new and old VDOM

- Lastly only updates the difference to browser (Real DOM)

JSX

JSX

- JSX stands for JavaScript XML.
- Browser uses HTML tags to render the content of the webpage
- .html files are used to write HTML tags
- React is javascript and it uses .js files.
- JSX allows us to write HTML tags in .js files.
- JSX makes it easier to write and add HTML in React.

JSX Example

- In HTML we write

```
<div> Hello world </div>
```

- In React

```
React.createElement(‘div’, null, ‘Hello World’)
```

JSX Example

- In HTML we write

```
<div>
    <h1> Hello world <h1>
</div>
```

- In React

```
React.createElement(
    ‘div’,
    null,
    React.createElement(h1’, null, ‘Hello World’)
)
```

JSX

- For nested tags React gives us easy way to write, for example

In HTML

```
<div>
   <h1>Some title</h1>
   <div>Some content</div>
</div>
```

In React

```
var myElement = (
        <div>
            <h1>Some title</h1>
            <div>Some content</div>
        </div>
)

ReactDOM.render(<myElement />,
         document.getElementById('root’)
)
```

JSX

- Important thing to remember, JSX needs to have one parent.

This will throw error

```
var myElement = (
        <h1>Some title</h1>
        <div>Some content</div>
)

ReactDOM.render(<myElement />,
 document.getElementById('root)
)
```

```
var myElement = (
    <div>
      <h1>Some title</h1>
      <div>Some content</div>
    </div>
)

ReactDOM.render(<myElement />,
 document.getElementById('root)
)
```

JSX

- We can also use React.Fragment instead of **<div>**

```
var myElement = (
      <React.Fragment>
            <h1>Some title</h1>
            <div>Some content</div>
      </React.Fragment>
)
ReactDOM.render(<myElement />, document.getElementById('root'))
```

Better way

Setting up React

Setting up React

- Easiest way to setup React environment is by using **create-react-app**
- **Create-react-app** scaffold a basic React application with ease and will get you up and running in no time.

To install create-react-app

```
aamir@ap-linux:~$ npm install create-react-app -g
```

Setting up React

To scaffold React application

```
aamir@ap-linux:~$ npx create-react-app my-app

aamir@ap-linux:~$ cd my-app

aamir@ap-linux:~/my-app$ npm start
```

http://localhost:3000

Our first React component

Our first React component

- There few types of component from which we will be working with following two type in this course
    - Class based Component
    - Function based Component

Class based Component

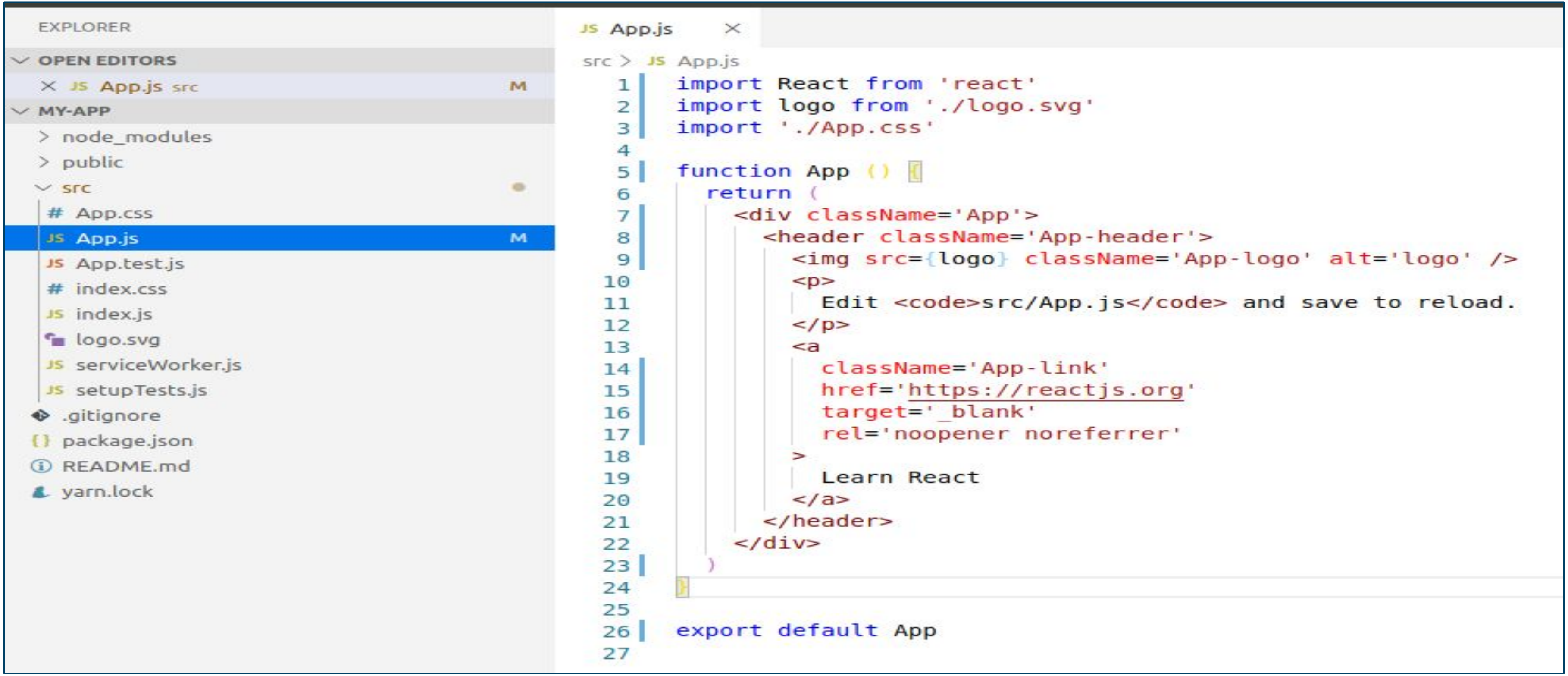
```
class MyComponent extends React.Component {
  render() {
    return (
      <div>
            <h1> Hello world </h1>
      </div>
    )
  }
}
ReactDOM.render(

    <MyComponent />,

    document.getElementById(‘root’)

    )
```

Functional based Component

```
function MyComponent () {
     return (
       <div>
             <h1> Hello world </h1>
       </div>
     )
}


ReactDOM.render(
     <MyComponent />,
     document.getElementById(‘root’)
     )
```

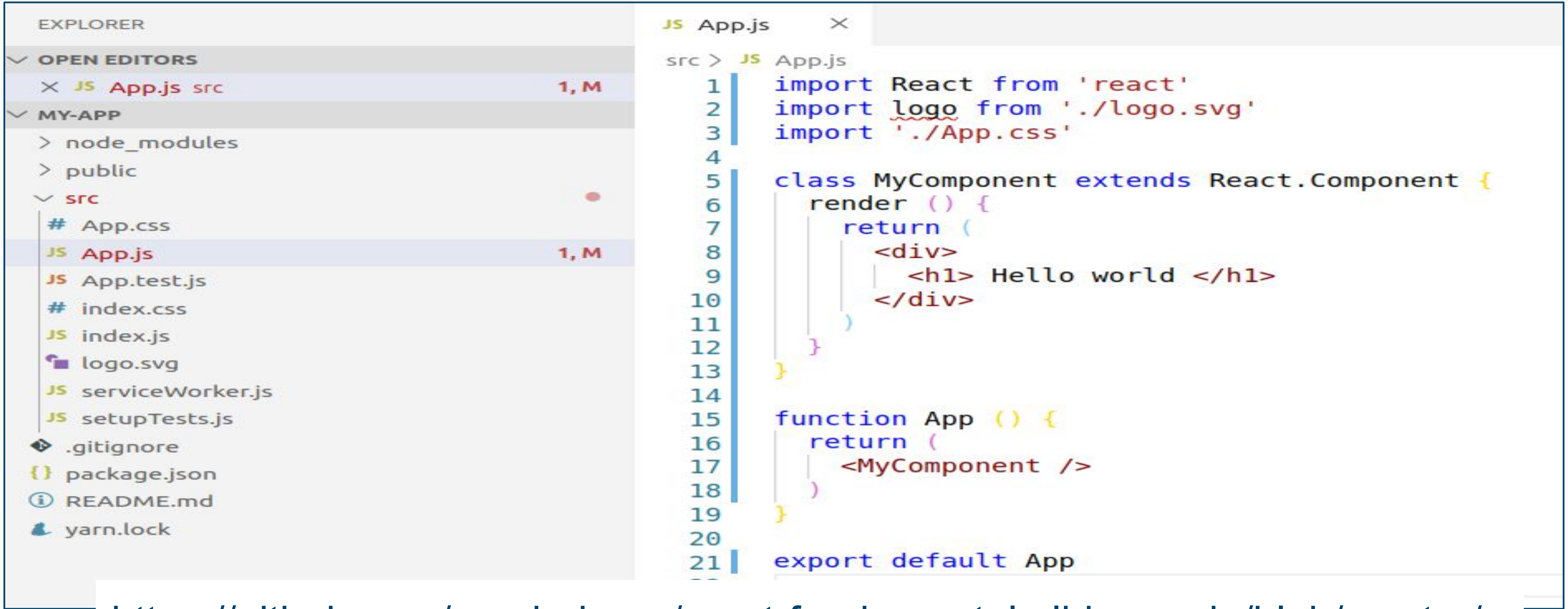
Create-react-app generated files

Directory and files after create-react-app



```
import React from 'react'
import logo from './logo.svg'
import './App.css'

function App () {
  return (
    <div className='App'>
      <header className='App-header'>
        <img src={logo} className='App-logo' alt='logo' />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className='App-link'
          href='https://reactjs.org'
          target='_blank'
          rel='noopener noreferrer'
        >
          Learn React
        </a>
      </header>
    </div>
  )
}

export default App

```

Our first React component



EXPLORER

OPEN EDITORS

- × App.js src 1, M

MY-APP

- node_modules
- public
- src
  - App.css
  - App.js 1, M
  - App.test.js
  - index.css
  - index.js
  - logo.svg
  - serviceWorker.js
  - setupTests.js
- .gitignore
- package.json
- README.md
- yarn.lock

App.js ×

src > App.js

```
import React from 'react'
import logo from './logo.svg'
import './App.css'

class MyComponent extends React.Component {
  render () {
    return (
      <div>
        <h1> Hello world </h1>
      </div>
    )
  }
}

function App () {
  return (
    <MyComponent />
  )
}

export default App
```

https://github.com/aamirpinger/react-fundamental-slides-code/blob/master/myFirstComponent.js

Component with .map method

```
import React from 'react'

class MyComponent extends React.Component {
 render () {
   const cityArray = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']

   return (
     <ul>
       {
         cityArray.map(city => <li key={city}> {city} </li>)
       }
     </ul>
   )
 }
}

function App () {
 return (<MyComponent />)
}

export default App
```

ComponentWithDotMap.js

Result

- Karachi
- Lahore
- Peshawar
- Quetta

Component with .filter method

```
import React from 'react'

class MyComponent extends React.Component {
 render () {
   const cityArray = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']

   const shortListedCities = cityArray.filter(city => city.length > 6)
   return (
     <ul>
       {
         shortListedCities.map(city => <li key={city}> {city} </li>)
       }
     </ul>
   )
 }
}

function App () {
 return (<MyComponent />)
}

export default App
```

ComponentWithDotFilter.js

Result

- Karachi
- Peshawar

Component Reusability

Component Reusability

```
import React from 'react'

class MyComponent extends React.Component {
 render () {
   const cityArray = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']

   const shortListedCities = cityArray.filter(city => city.length > 6)
   return (
     <ul>
     { shortListedCities.map(city => <li key={city}> {city} </li>)}
     </ul>
   )}
}

function App () {
 return (
   <div>
     <MyComponent />
     <MyComponent />
   </div>
)}
export default App
```

ComponentWithDotFilter.js

Result

- Karachi
- Peshawar

- Karachi
- Peshawar

props

props

- React allow us to send data or functions from parent component to it’s child component
- These data or functions are called props
- Adding props, or properties to your component means we add attributes to our component element.

props

Component call WITHOUT props

```
<myElement />
```

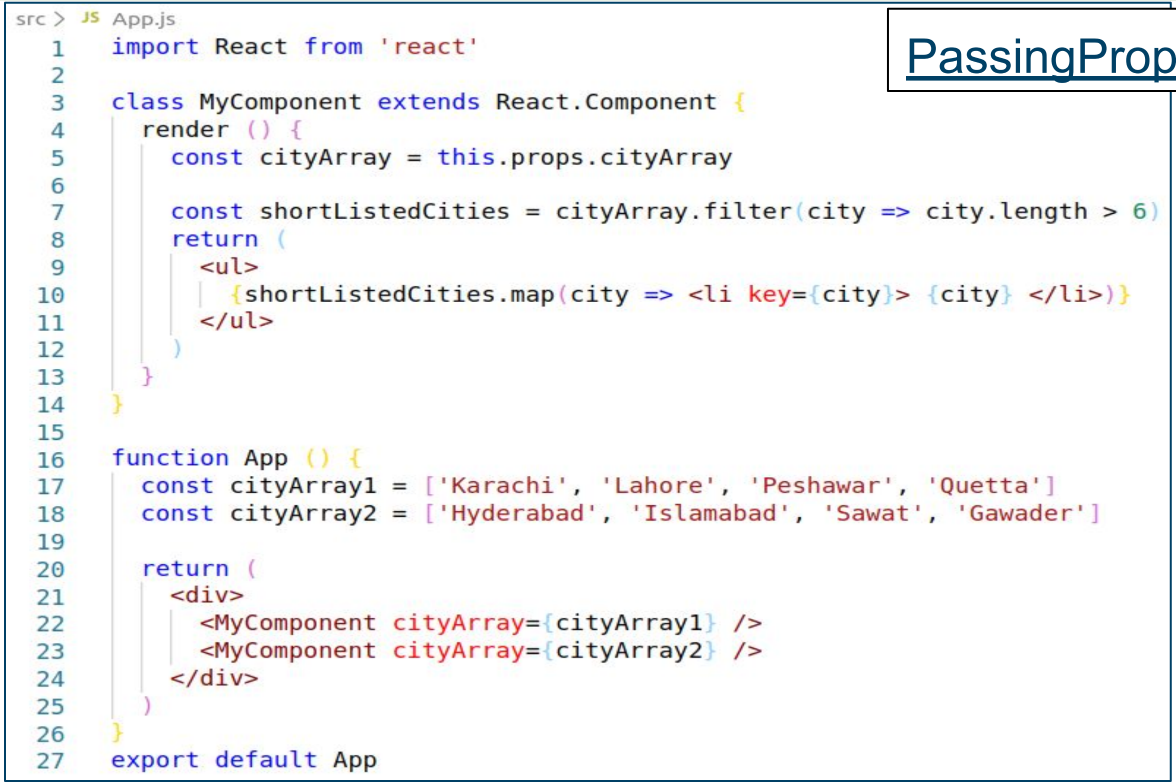
Component call WITH props

```
<myElement
      name=’Aamir’
      myfunc={() => console.log(‘hello world’)}
/>
```

props

- This means we can pass data from an outer component to an inner component.
- That data can be either data that we want to render a or a function that we want to invoke.

props



src > JS App.js

```
import React from 'react'

class MyComponent extends React.Component {
  render () {
    const cityArray = this.props.cityArray

    const shortListedCities = cityArray.filter(city => city.length > 6)
    return (
      <ul>
        {shortListedCities.map(city => <li key={city}> {city} </li>)}
      </ul>
    )
  }
}

function App () {
  const cityArray1 = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']
  const cityArray2 = ['Hyderabad', 'Islamabad', 'Sawat', 'Gawader']

  return (
    <div>
      <MyComponent cityArray={cityArray1} />
      <MyComponent cityArray={cityArray2} />
    </div>
  )
}
export default App
```

PassingProps.js

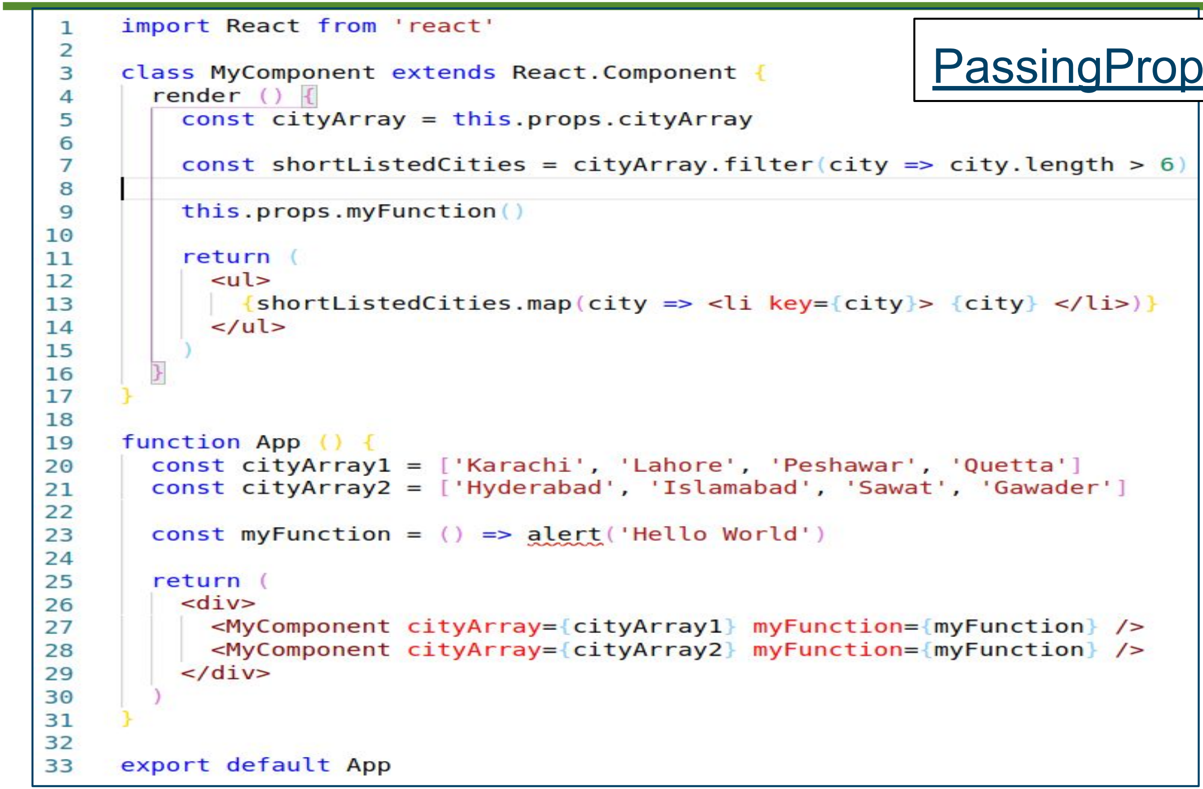
Result

- Karachi
- Peshawar

- Hyderabad
- Islamabad
- Gawader

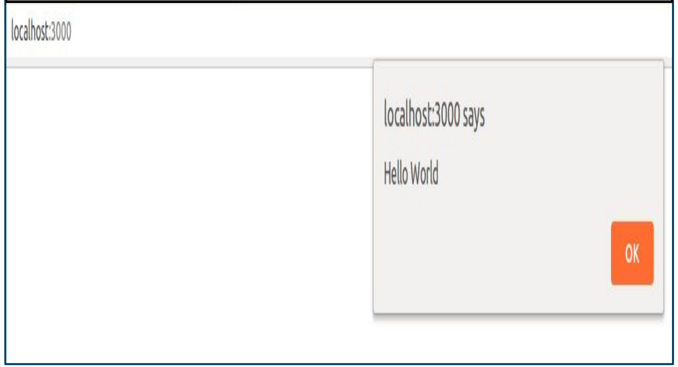
props

PassingPropsAndFunction.js



```
import React from 'react'

class MyComponent extends React.Component {
  render () {
    const cityArray = this.props.cityArray

    const shortListedCities = cityArray.filter(city => city.length > 6)

    this.props.myFunction()

    return (
      <ul>
        {shortListedCities.map(city => <li key={city}> {city} </li>)}
      </ul>
    )
  }
}

function App () {
  const cityArray1 = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']
  const cityArray2 = ['Hyderabad', 'Islamabad', 'Sawat', 'Gawader']

  const myFunction = () => alert('Hello World')

  return (
    <div>
      <MyComponent cityArray={cityArray1} myFunction={myFunction} />
      <MyComponent cityArray={cityArray2} myFunction={myFunction} />
    </div>
  )
}

export default App
```

Result



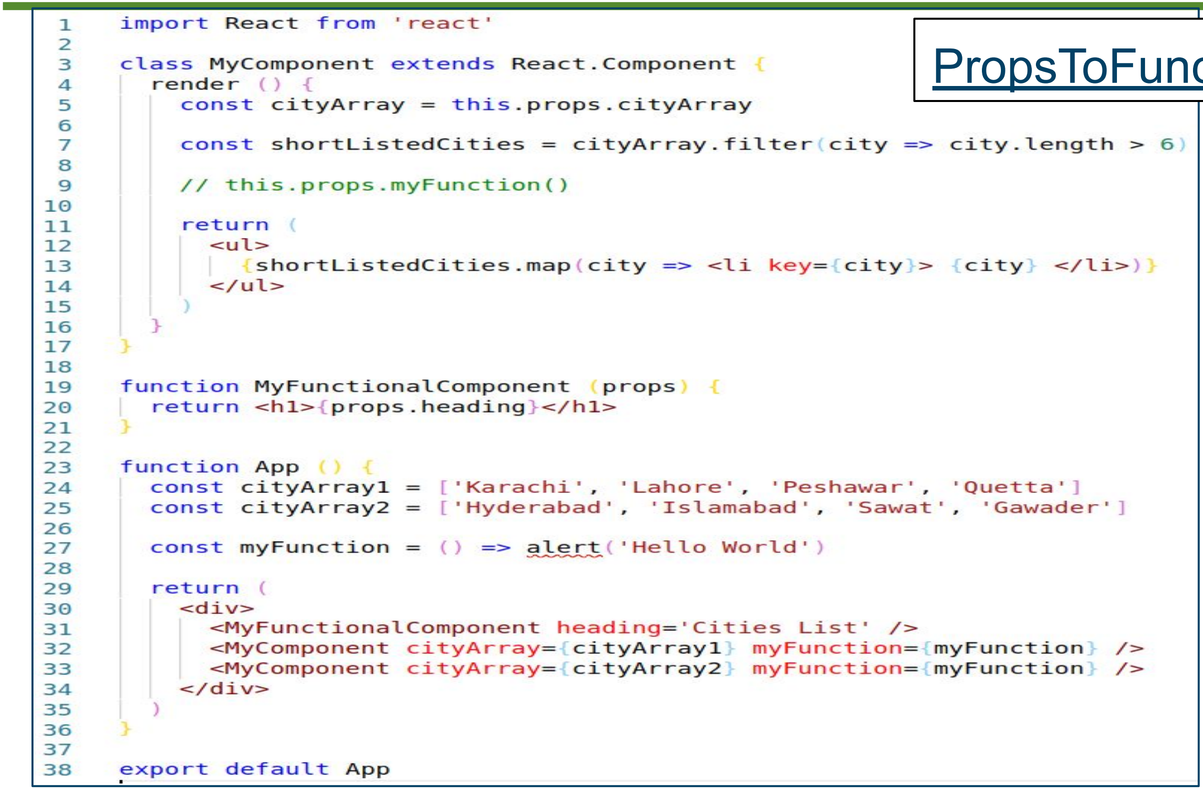
localhost:3000 says

Hello World

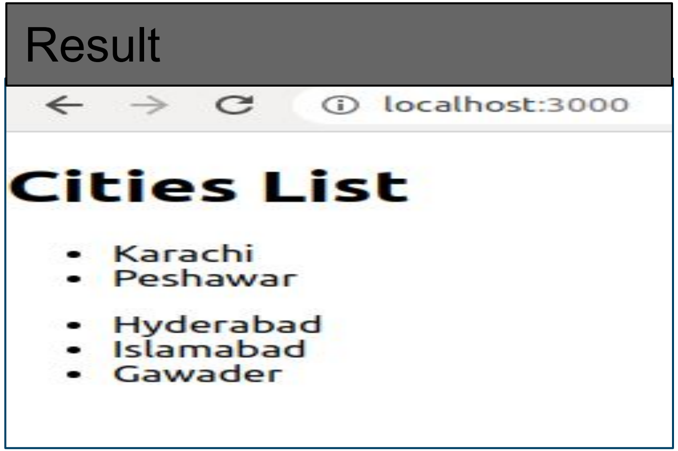
Passing Props to Functional component

Props to Functional Component

PropsToFunctionalComp.js



```
import React from 'react'

class MyComponent extends React.Component {
  render () {
    const cityArray = this.props.cityArray

    const shortListedCities = cityArray.filter(city => city.length > 6)

    // this.props.myFunction()

    return (
      <ul>
        {shortListedCities.map(city => <li key={city}> {city} </li>)}
      </ul>
    )
  }
}

function MyFunctionalComponent (props) {
  return <h1>{props.heading}</h1>
}

function App () {
  const cityArray1 = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']
  const cityArray2 = ['Hyderabad', 'Islamabad', 'Sawat', 'Gawader']

  const myFunction = () => alert('Hello World')

  return (
    <div>
      <MyFunctionalComponent heading='Cities List' />
      <MyComponent cityArray={cityArray1} myFunction={myFunction} />
      <MyComponent cityArray={cityArray2} myFunction={myFunction} />
    </div>
  )
}

export default App
```



Result

Cities List

- Karachi
- Peshawar

- Hyderabad
- Islamabad
- Gawader

PropTypes

PropTypes

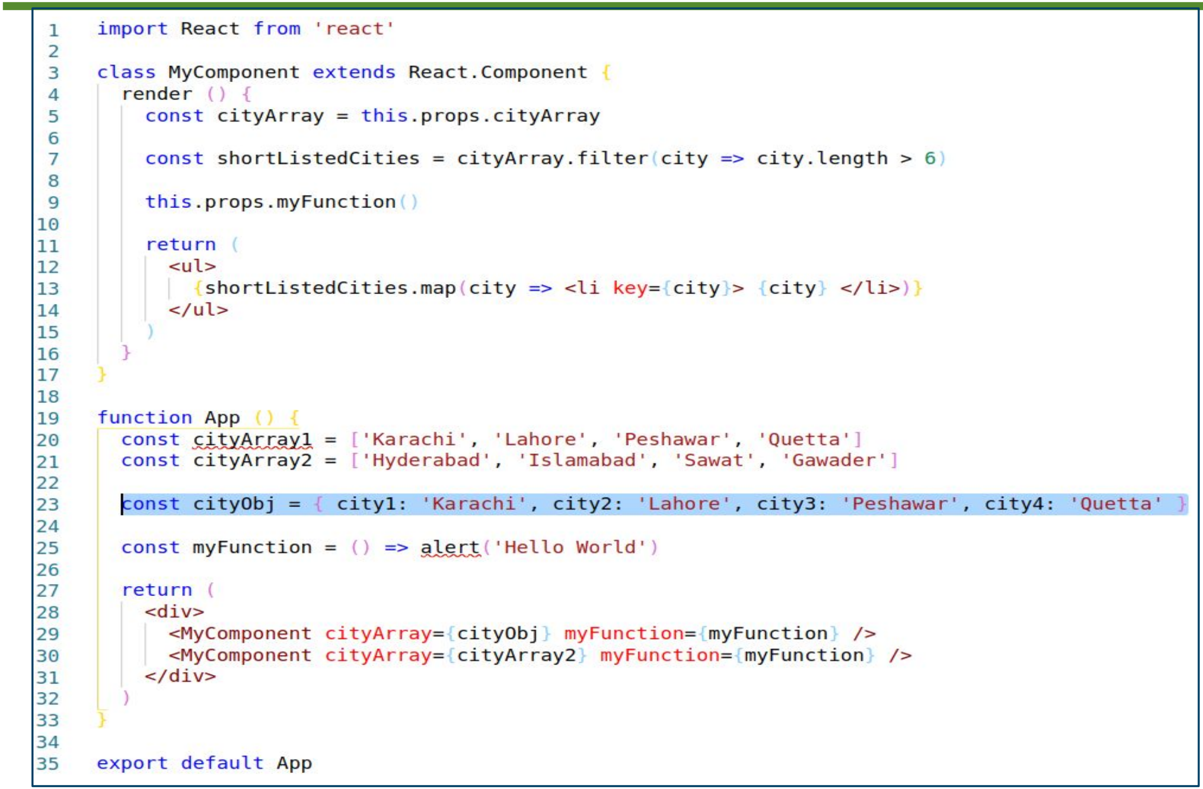
- React components are meant to be reuse
- In a bigger application soon teams will work on a project and reuse other's components
- Props is integral part of component

QUESTION ARISES?

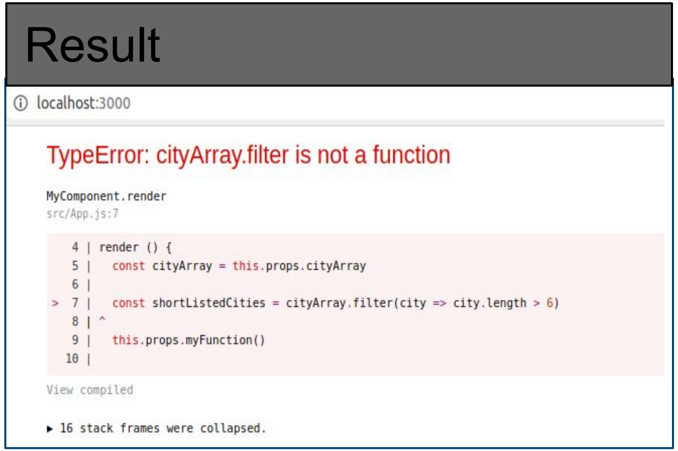
What if wrong props are passed to our component

e.g. array instead of object

Wrong props



```
import React from 'react'

class MyComponent extends React.Component {
  render () {
    const cityArray = this.props.cityArray

    const shortListedCities = cityArray.filter(city => city.length > 6)

    this.props.myFunction()

    return (
      <ul>
        {shortListedCities.map(city => <li key={city}> {city} </li>)}
      </ul>
    )
  }
}

function App () {
  const cityArray1 = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']
  const cityArray2 = ['Hyderabad', 'Islamabad', 'Sawat', 'Gawader']

  const cityObj = { city1: 'Karachi', city2: 'Lahore', city3: 'Peshawar', city4: 'Quetta' }

  const myFunction = () => alert('Hello World')

  return (
    <div>
      <MyComponent cityArray={cityObj} myFunction={myFunction} />
      <MyComponent cityArray={cityArray2} myFunction={myFunction} />
    </div>
  )
}

export default App
```



Result

```
TypeError: cityArray.filter is not a function

MyComponent.render
src/App.js:7

   4 | render () {
   5 |   const cityArray = this.props.cityArray
   6 |
>  7 |   const shortListedCities = cityArray.filter(city => city.length > 6)
   8 | ^
   9 |   this.props.myFunction()
  10 |

View compiled

▶ 16 stack frames were collapsed.
```

PropTypes

- By using **PropTypes** library, we can define the data type we expect if someone uses our component and passes props to it.

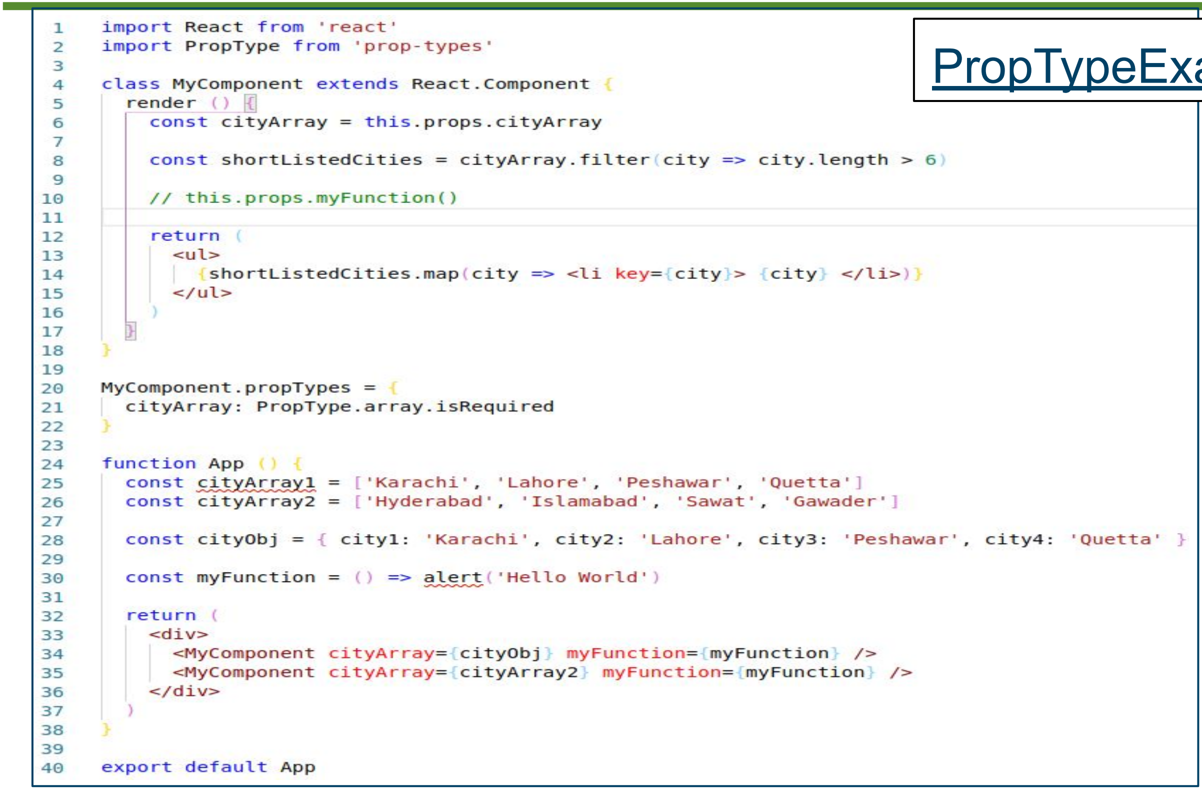
- It will warns us during development in the console of browser if unexpected data type passed

To install PropTypes

```
aamir@ap-linux:~$ npm install prop-types
```

More details: https://github.com/facebook/prop-types

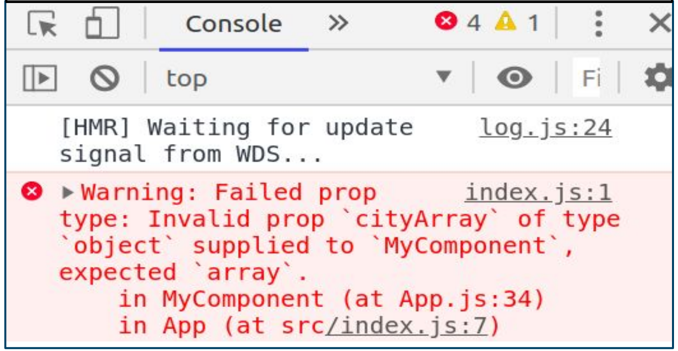
PropTypes



```
import React from 'react'
import PropType from 'prop-types'

class MyComponent extends React.Component {
  render () {
    const cityArray = this.props.cityArray

    const shortListedCities = cityArray.filter(city => city.length > 6)

    // this.props.myFunction()

    return (
      <ul>
        {shortListedCities.map(city => <li key={city}> {city} </li>)}
      </ul>
    )
  }
}

MyComponent.propTypes = {
  cityArray: PropType.array.isRequired
}

function App () {
  const cityArray1 = ['Karachi', 'Lahore', 'Peshawar', 'Quetta']
  const cityArray2 = ['Hyderabad', 'Islamabad', 'Sawat', 'Gawader']

  const cityObj = { city1: 'Karachi', city2: 'Lahore', city3: 'Peshawar', city4: 'Quetta' }

  const myFunction = () => alert('Hello World')

  return (
    <div>
      <MyComponent cityArray={cityObj} myFunction={myFunction} />
      <MyComponent cityArray={cityArray2} myFunction={myFunction} />
    </div>
  )
}

export default App
```

PropTypeExample.js

Result

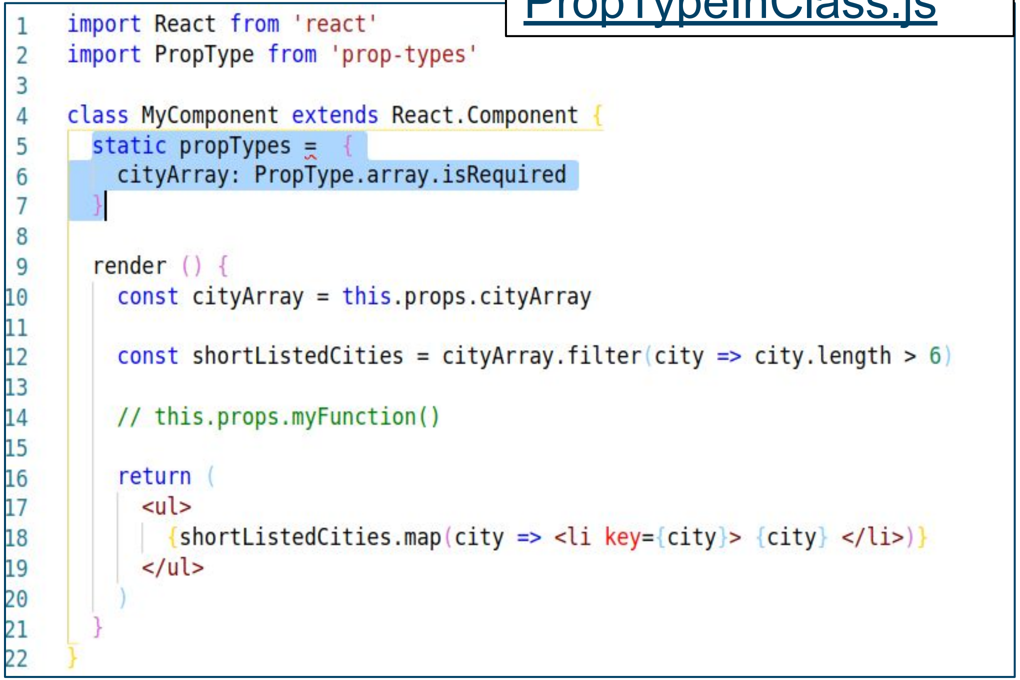


```
[HMR] Waiting for update        log.js:24
signal from WDS...
✖ ▸Warning: Failed prop          index.js:1
type: Invalid prop `cityArray` of type
`object` supplied to `MyComponent`,
expected `array`.
    in MyComponent (at App.js:34)
    in App (at src/index.js:7)
```

PropTypes

- It is considered a best practice to always use this library
- It is also recommended to make it part of the class

PropTypeInClass.js



```
import React from 'react'
import PropType from 'prop-types'

class MyComponent extends React.Component {
  static propTypes =  {
    cityArray: PropType.array.isRequired
  }

  render () {
    const cityArray = this.props.cityArray

    const shortListedCities = cityArray.filter(city => city.length > 6)

    // this.props.myFunction()

    return (
      <ul>
        {shortListedCities.map(city => <li key={city}> {city} </li>)}
      </ul>
    )
  }
}
```

State

State

- The heart of every React component is its “state”
- It is simply an Javascript object that determines how that component renders & behaves
- In other words, “state” is what allows you to create components that are dynamic and interactive

State

- Props from parent component are immutable
- State is mutable
- Change in state re-renders your component at browser
- Every change in state provide updated data to the user

State

- By rules State can only be declared for class components
- That’s what makes you decide whether to use class component or functional component
- By using React Hooks you can now have React state in the functional component

Place to declare State

Where state should be declared

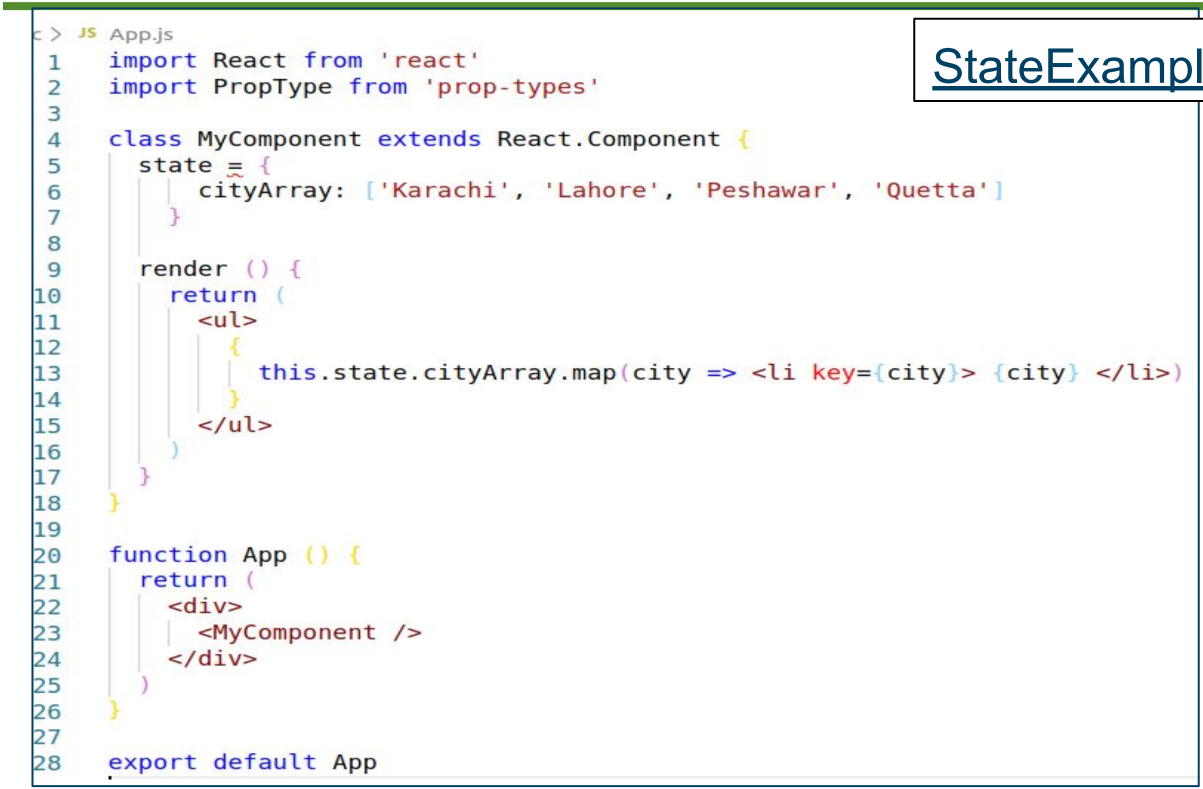
- State can only be mutate by owner component
- You can make one parent component with state and pass the state value to its child components as a props
- Props cannot change they are immutable
- If child component need to make any changes in state it can request its parent (owner) component to make changes

Where state should be declared

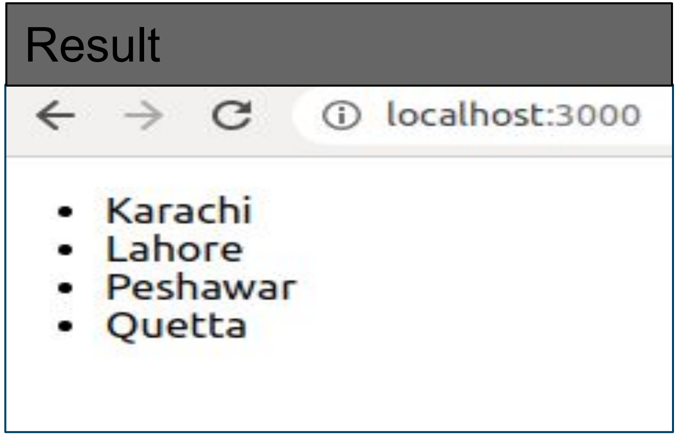
- Parent component has to pass a function as a props which will be used by child component to request a state change
- Once changes are done by parent, react automatically pass updated state value to all the child component where parent component have passed state values as a props
- This way React insures single source of truth and unidirectional data model

State

StateExample.js



```
import React from 'react'
import PropType from 'prop-types'

class MyComponent extends React.Component {
  state = {
      cityArray: ['Karachi', 'Lahore', 'Peshawar', 'Quetta']
    }

  render () {
    return (
      <ul>
        {
          this.state.cityArray.map(city => <li key={city}> {city} </li>)
        }
      </ul>
    )
  }
}

function App () {
  return (
    <div>
      <MyComponent />
    </div>
  )
}

export default App
```



- Karachi
- Lahore
- Peshawar
- Quetta

Destructuring

Destructuring - Array

- The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables

```
let myArray = ["Aamir","Pinger",2]

let [firstName, lastName, degrees] = myArray

console.log(firstName)
console.log(lastName)
console.log(degrees)

let sentence = `${firstName} ${lastName} has ${degrees} Masters degrees.`

console.log(sentence)
```

RESULT

```
Aamir
Pinger
2
Aamir Pinger has 2 Masters degrees.
```

[DestructuringArray1.js]

Destructuring - Array

```
let myArray = ["Aamir","Pinger",2]

let [firstName, lastName] = myArray

console.log(firstName)
console.log(lastName)
console.log(degrees)

let sentence = `${firstName} ${lastName} has ${degrees} Masters degrees.`

console.log(sentence)
```

RESULT

```
Aamir
Pinger
degrees is not defined
```

DestructuringArray2.js

Destructuring - Object

```
let myObject = {firstName: "Aamir", lastName: "Pinger", degrees: 2}

console.log(myObject.firstName)
console.log(myObject.lastName)
console.log(myObject.degrees)

let {firstName, lastName, degrees} = myObject

console.log(firstName)
console.log(lastName)
console.log(degrees)

let sentence = `${firstName} ${lastName} has ${degrees} Masters degrees`

console.log(sentence)
```

RESULT

```
Aamir
Pinger
2
Aamir
Pinger
2
Aamir Pinger has 2 Masters degrees.
```

Destructuring - Object

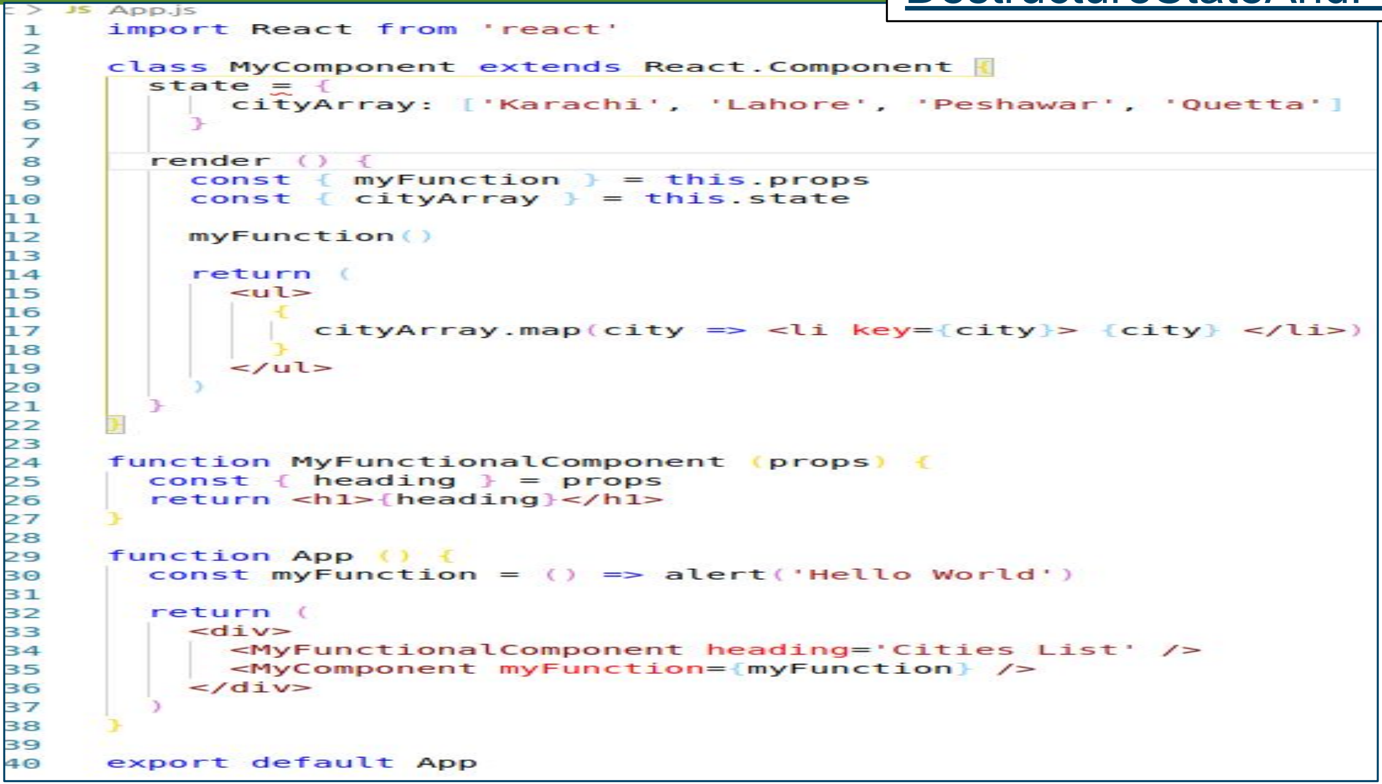
```
let myObject = {firstName: "Aamir", lastName: "Pinger", degrees: 2}

console.log(myObject.firstName)
console.log(myObject.lastName)
console.log(myObject.degrees)

let {firstName, lastName} = myObject

console.log(firstName)
console.log(lastName)
console.log(degrees)

let sentence = `${firstName} ${lastName} has ${degrees} Masters degrees`

console.log(sentence)
```

RESULT

```
Aamir
Pinger
2
Aamir
Pinger
degrees is not defined
```

Destructure State and Props

DestructureStateAndProps.js



```
import React from 'react'

class MyComponent extends React.Component {
  state = {
    cityArray: ['Karachi', 'Lahore', 'Peshawar', 'Quetta']
  }

  render () {
    const { myFunction } = this.props
    const { cityArray } = this.state

    myFunction()

    return (
      <ul>
        {
          cityArray.map(city => <li key={city}> {city} </li>)
        }
      </ul>
    )
  }
}

function MyFunctionalComponent (props) {
  const { heading } = props
  return <h1>{heading}</h1>
}

function App () {
  const myFunction = () => alert('Hello World')

  return (
    <div>
      <MyFunctionalComponent heading='Cities List' />
      <MyComponent myFunction={myFunction} />
    </div>
  )
}

export default App
```

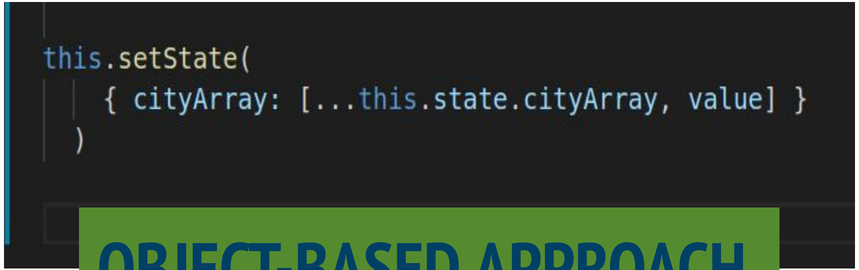
Changing State

Changing State

- To change state we use **setState** method
- Beauty of setState method is no matter if your state got many key value pairs, you can just pass specific keys and their new values you want to update
- Once setState method updates the state it rerenders that component and your browser gets updated data

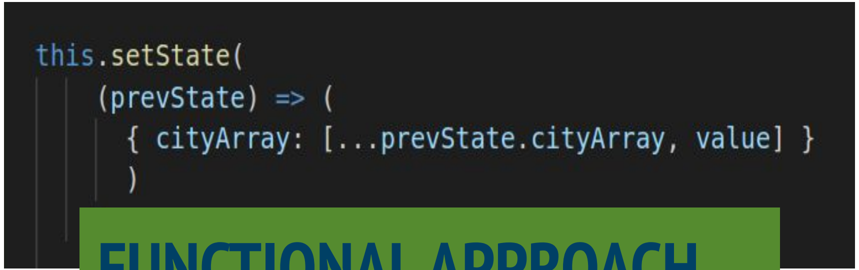
setState Method

- setState function in React is an asynchronous function
- There are two variations of using setState:
  - Object-based approach
  - Functional approach



```
this.setState(
    { cityArray: [...this.state.cityArray, value] }
  )
```

OBJECT-BASED APPROACH



```
this.setState(
    (prevState) => (
      { cityArray: [...prevState.cityArray, value] }
    )
```

FUNCTIONAL APPROACH

setState Method

Add city name example Code:
AddCityInState.js

Remove city name example Code:
RemoveCityFromState.js

Forms

Forms

- Form is not new in react, it is same as we used in HTML
- Form is helpful to group multiple input fields and validate on submit
- These input fields can be any of the HTML input field types. Examples are:
    - Textbox
    - Checkbox
    - Radio
    - TextArea
    - Date input
    - Select

Form Example

Form example Code:

FormExample.js

Conditional rendering

Conditional rendering

- Conditional rendering is about deciding what to render and when
- It is simply rendering something if provided condition is matched
- For example:
    - Maximum limit message component
    - Render an error message instead of alert in case of no city name provided on submit

Conditional rendering

- There are different approaches we can have here:
  - render if true (if else)
  - ternary expression (condition ? true : false)
  - Short circuit evaluation (&&, ||)

Conditional rendering Example

Conditional rendering example Code:

ConditionalRendering.js

Styling

Styling

- Styling is also a important aspect to look when developing a web app
- No matter how efficient you have made you web app, if you did not made a good UI/UX, user may not use it
- React provide us multiple ways to use styling
- Some of the widely used practices are
  - CSS files
  - Inline Styling
  - 3rd party libraries like *Styled Components*

Styling with CSS files

Styling with CSS files

CSS styling example Code:

CSS-Styling/

Inline Styling

Inline Styling

Inline styling example Code:
InlineStyling.js

List of all supported HTML attributes that can be passed along to the React element
https://facebook.github.io/react/docs/dom-elements.html#all-supported-html-attributes

Styled Component

Styled Component

- CSS files have few issues like these are global in scope
- Similar class name could cause side effects to other part of website
- Inline styling though not recommended to use but it is sometimes required to make small dynamic styling
- Styled components is a library that give us solution for above all
- Using style components you don’t need css files, you create styled HTML element in JS files and use them wherever you need them

Styled Component

Styled Component example Code:

CSS-Styling/styled-component-example.js

To install styled-components

```
aamir@ap-linux:~$ npm install styled-components
```

Images in React

Images

- Rendering images in a react is little different
- In HTML you write

```
<img src="smiley.gif" alt="Smiley face" height="42" width="42">
```

- In React you have to import a file right in a JavaScript module
- This tells webpack to include that file in the bundle during the project start.

img tag example Code:
CSS-Styling/imgTagExample.js

Lifecycle Events

Lifecycle Events

- When developing in React, every Component follows a cycle from
  - When it’s created and Mounted on the DOM
  - Got updated during its presence on DOM
  - To when it is unmounted from DOM and destroyed
- This is what we refer to as the Component lifecycle

Lifecycle Events

- The React component which extends React.Component goes through the following phases:
  - Mounting
  - Updating
  - Unmounting
- React offer us various methods which are invoked at different phases of the lifecycle of a component.

Lifecycle Events

- Mounting phase is initialization of the component when the instance of the component is being created and inserted into the DOM

- Updating phase comes when component is re-rendered due to change in props or state etc

- Unmounting phase is when unmounting or removal of component from the DOM

Why lifecycle events?

Why lifecycle events?

- Render method should be pure and should not have any side effects
- Pure method means you get the same output every time when provided same input.
- Component’s render() function should be pure, meaning that it does not modify component state

Why lifecycle events?

- render() Is meant for Rendering, Only!
- As we said its pure and should return same result when provided same input so that means should not make
  - Any HTTP requests
  - Fetch data that's used to display the content
  - Modify the DOM
  - Or call any function that does the above 3

Lifecycle Events

- There are special methods bound to the every component
- Those methods are triggered on some certain time during component lifecycle
- We can use those methods to do thing which we don’t do in render()

Lifecycle Events

- React offer us various methods which are invoked at different phases of the lifecycle of a component.

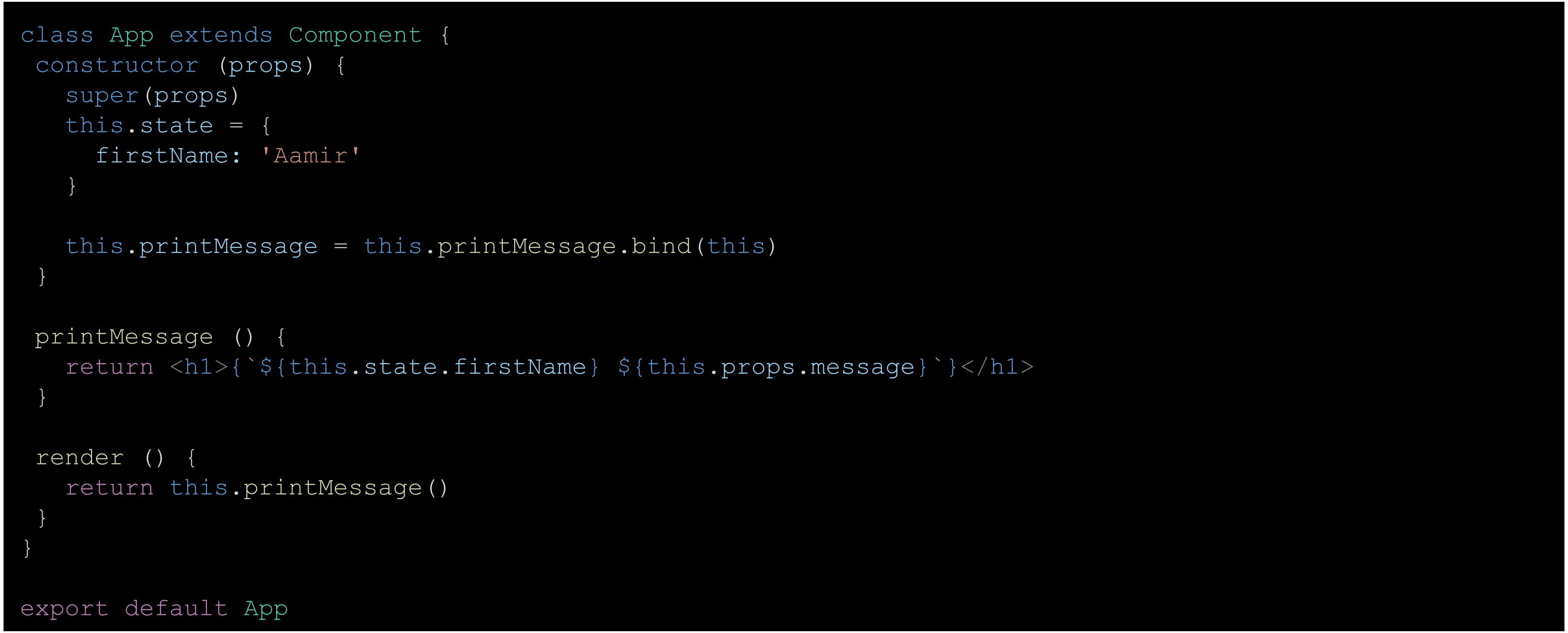
Mounting	Updating	Unmounting
1. constructor()	1. static getDerivedStateFromProps()	1. componentWillUnmount()
2. static getDerivedStateFromProps()	2. shouldComponentUpdate()	
3. render()	3. render()	
4. componentDidMount()	4. getSnapshotBeforeUpdate()	
	5. componentDidUpdate()	

constructor()

constructor()

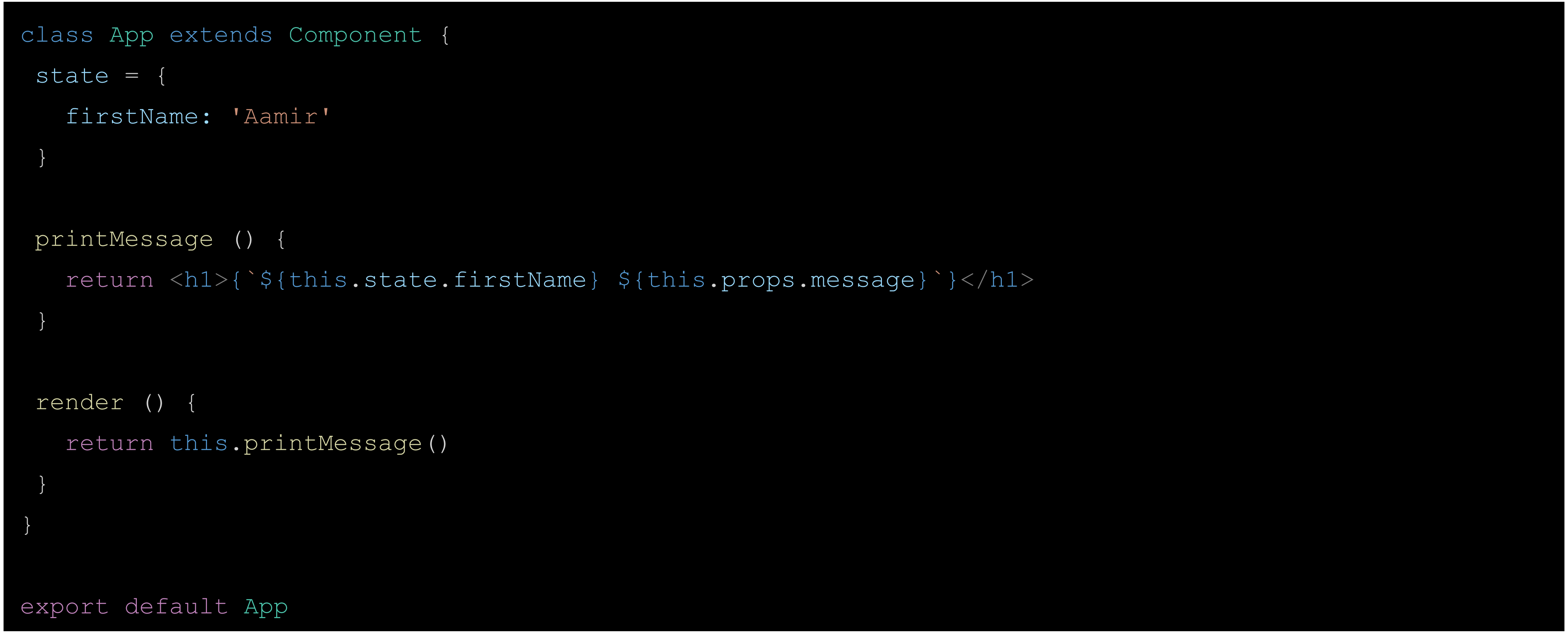
- Constructors are not react specific or even javascript specific
- They are specific to the inheritance in OOP
- When the component class is created in memory, the constructor is the first method called, so it’s the right place to initialize everything – state included
- Earlier we need to define constructor method explicitly but babel now transpile state directly without writing it in constructor()

constructor()



```
class App extends Component {
 constructor (props) {
   super(props)
   this.state = {
     firstName: 'Aamir'
   }

   this.printMessage = this.printMessage.bind(this)
 }

 printMessage () {
   return <h1>{`${this.state.firstName} ${this.props.message}`}</h1>
 }

 render () {
   return this.printMessage()
 }
}

export default App
```

constructor()



```
class App extends Component {
 state = {
   firstName: 'Aamir'
 }

 printMessage () {
   return <h1>{`${this.state.firstName} ${this.props.message}`}</h1>
 }

 render () {
   return this.printMessage()
 }
}

export default App
```

componentDidMount()

componentDidMount()

- componentDidMount() is invoked only ONCE immediately after a component is mounted (users will have a display on there browsers)
- If you need to load data from a remote endpoint (API Calls), this is a good place
- Initializing state also should be done here
- Setting state in this method will trigger a re-render to update the data everywhere

componentDidMount()

componentDidMount() example Code:
componentDidMountExample.js

static getDerivedStateFromProps()

static getDerivedStateFromProps()

- Invoked right before calling the render method, both on the initial mount and on subsequent updates

- It should return an object to update the state, or null to update nothing.

- This method exists for rare use cases where the state depends on changes in props over time

static getDerivedStateFromProps()

getDerivedStateFromProps() example Code:

getDerivedStateFromPropsExample.js

shouldComponentUpdate()

shouldComponentUpdate()

- It returns true by default.
- Whenever a component's state or props (its parent's state) is updated, the component re-renders we can stop updating the state if we want
- This method is not called for the initial render or when forceUpdate() is used
- shouldComponentUpdate() allows us to say: only update if the props you care about change

shouldComponentUpdate()

shouldComponentUpdate() example Code:
shouldComponentUpdateExample.js

componentDidUpdate()

componentDidUpdate()

- It is called right after the component update takes place, except the first time when the component is rendered.
- componentDidUpdate() takes two arguments as parameters, prevProps and prevState
- This method is useful if you want to take any actions on DOM after received updated props from parent or re-render caused by setState
- This is also a suitable place when we want to log the new changes

componentDidUpdate()

componentDidUpdate() example Code:
componentDidUpdateExample.js

componentWillUnmount()

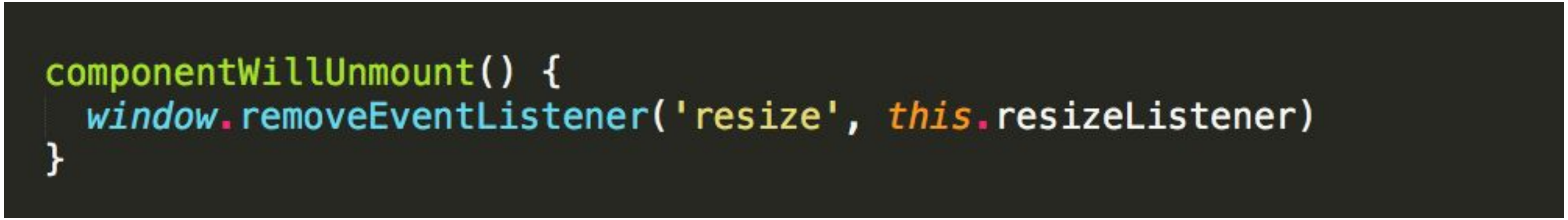
componentWillUnmount()

- This event (method) is invoked immediately before a component component is being removed from the DOM
- Recommended to perform any necessary cleanup in this method, such as
    - Invalidating timers,
    - Canceling network requests, or
    - cleaning up any subscriptions that were created in componentDidMount()

componentWillUnmount()

- setState() will not effect if called in componentWillUnmount() because the component cannot re-rendered once unmounted

componentWillUnmount() example:



```
componentWillUnmount() {
  window.removeEventListener('resize', this.resizeListener)
}
```

Components file/directory structure

Components file/directory structure

- Uptil now we have made a single App.js file and made all our component into it
- This is not at all a good approach
- Components should always made in different folders and .js files
- Keep app supporting function into utils folder
- Keep all images to one static resources folder etc

Components file/directory structure

Component splitted to file/folders example Code:
SampleFolderStructure/

Assignment

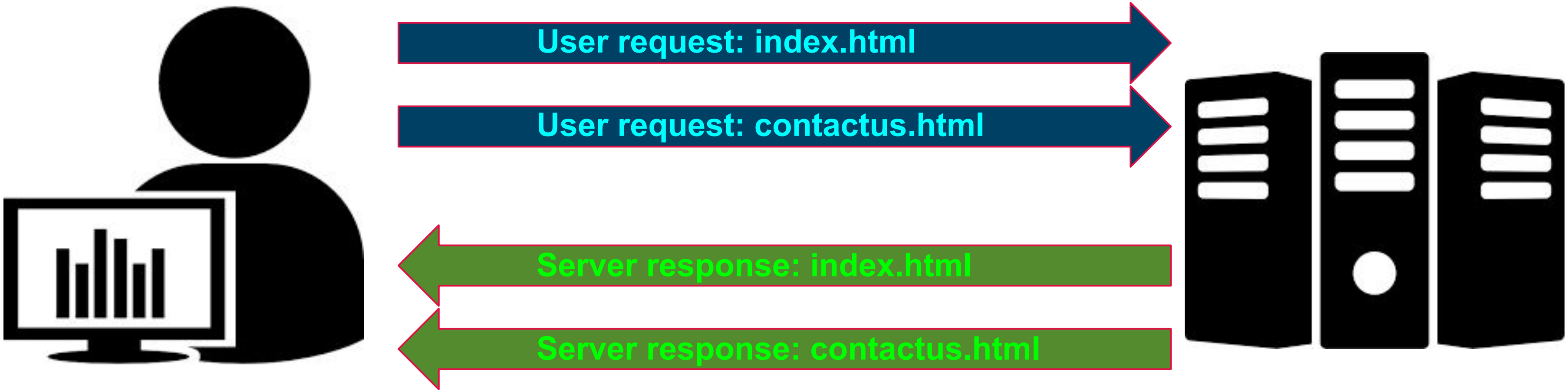
Create a todo application that

1. Fetch the todo records just once from https://jsonplaceholder.typicode.com/
2. Store them in local component state
3. Add a new todo functionality
4. Delete any todo functionality
5. List all todo functionality
6. Search any todo on Title field

Please note: Call API only once to get the records and then use/modify it locally using state

React Router

React Router



User request: index.html

User request: contactus.html

Server response: index.html

Server response: contactus.html

Normal flow of user request on browser and response from server

React Router

- Single-page applications can work in different ways
- In react we have only index.html
- React routers that user move to different pages in your web app without actually changing any page, even URL will show change of page
- One way is you can do have conditional rendering and move to different screens (components) on the browser
- Above way is not easy to manage and does not give browser back button

React Router

- Routing in a Single Page Application is the way to introduce some features to navigating the app through links, which are expected in normal web applications like:
  - The browser should change the URL when you navigate to a different screen
  - Deep linking should work: if you point the browser to a URL, the application should reconstruct the same view that was presented when the URL was generated
  - The browser back (and forward) button should work like expected

React Router

- React-router-dom is a library for routing in React SPA
- It provides all the previously discussed navigation features to SPA

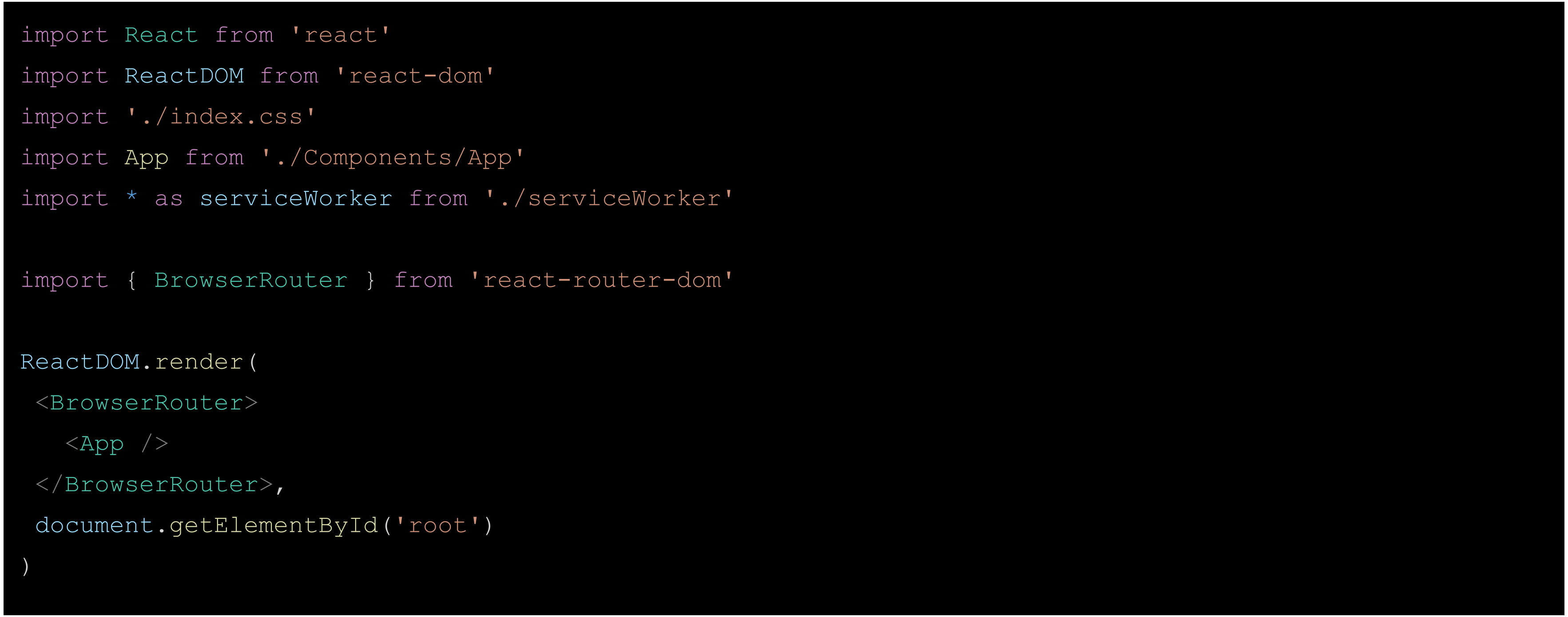
To install react-router-dom

```
aamir@ap-linux:~$ npm install react-router-dom
```

React Router - <BrowserRouter>

- React-router-dom library provides **<BrowserRouter />** component
- The purpose of this component is to listen to URL changes
- When URL changes this component will make sure the correct screen (page) of your web app get render on the screen
- For React Router to work properly, we must have to wrap our whole app into a BrowserRouter component

<BrowserRouter> - index.js



```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './Components/App'
import * as serviceWorker from './serviceWorker'

import { BrowserRouter } from 'react-router-dom'

ReactDOM.render(
 <BrowserRouter>
   <App />
 </BrowserRouter>,
 document.getElementById('root')
)
```

<Route>

<Route>

- Our routes will be defined in <Route> element
- We specify which path they will match and what component that should respond
- Let’s say we want
  - **www.ourDomain.com/** to go to City list page

    ```
    <Route exact path='/' component={CityList} />
    ```

  - **www.ourDomain.com/aboutus** to go to About us page

    ```
    <Route exact path='/aboutus' component={AboutUs} />
    ```

<Route>

- Let’s say you want to pass a some dynamic value to any route at runtime
- You can achieve this by for example

```
<Route exact path='/cityList/:city' component={CityList} />
```

- For components where we passes props, **component** attribute will be replaced by **render=() => (<your component>)**

```
<Route exact path='/cityList' render={() => (<CityList city={['Karachi']}>)} />
```

React Router

React Router example Code:

RoutesExample/

<switch>

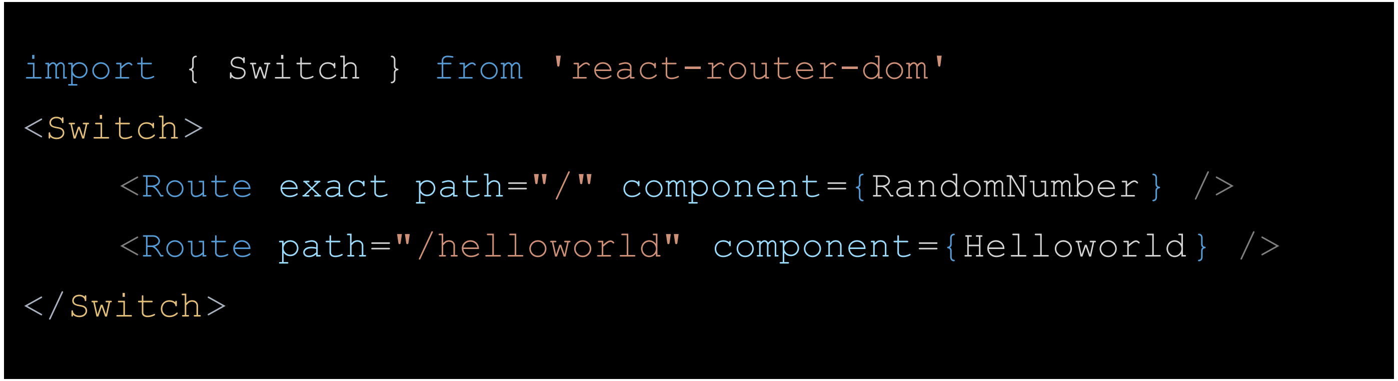
<switch>

- We can add something like following to our code

```
<Route render={() => <AnyComponent /> } />
```

- Only problem with this is it will render with every route no matter what path is provided
- To solve this problem we must wrap all above routes with <Switch></Switch>
- By doing this we actually are instructing that as URL path matches with path of any route first render that component and then jump out of switch block and do not go and check next routes

React Router



```
import { Switch } from 'react-router-dom'
<Switch>
    <Route exact path="/" component={RandomNumber} />
    <Route path="/helloworld" component={Helloworld} />
</Switch>
```

React Router example Code:

RoutesExample/

Programmatic navigation

Programmatic navigation

- In many cases you will be needing a way to programmatically navigate using React Router v4
- This can be achieved by using a <Redirect /> component

<Redirect to='/dashboard' />

- You can also pass

```
   <Redirect to={{
   pathname: "/login",
   state: {
     referrer: this.props.location.pathname,
     humanType: "Cat Person",
     age: 12,
   }
}} />
```

To access these values this.props.location.state

React Router

<Link />

- It is a component of react-router-dom library

<Link to="/helloworld">Click here</Link>

same as

<a href=”/helloworld”>Click here</a>

- The purpose of this component is add provided route to URL
- Based on changes in URL <BrowserRouter> will sense the change in URL and render new UI
- Above all Back / Forward button of browser will also work

React Router

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom'

 class App extends Component {
   render(){
     return(
       <div>
       <Link to="/helloworld">Goto to Hello World Page</Link>
     </div>
)}
 export default App
```

Thank you