



React JS & Redux Hands On Training

Module : React JS

Training Duration: 4 days

Pankaj Sharma | Sr. Technical Trainer

- Be On-Time & Refrain from taking frequent breaks
- Only nominated participants are allowed & in case of any other, needs to contact program coordinator
- It is mandate to attend all sessions and any deviation to be notified to program coordinator
- Have notepad and pen handy, to make notes of the session
- Mobiles in silent mode & refrain from any other gadgets or distractions
- Listen actively & Participate to the fullest of your ability
- Speak from your experience instead of generalizing
- Have right questions, follow manners & body language while you communicate around
- In case of any support needed please contact program coordinator
- Switch off LCD, lights and rearrange chairs post completion of session
- In case of any unusual behaviours or technical challenge, please do reach to session coordinator

Full Name	Pankaj Sharma
Designation or Role	Sr. Technical Trainer
Place	Indore
Overall Experience (years & description):	12 Years (AS a Corporate trainer and Developer)
Relevant Experience (years & description):	10 Years
Hobbies/Interests	Playing classical flute
Contact Number	8827611875
Email ID	Sharma.Pankaj@yash.com , pankajsimmc@gmail.com

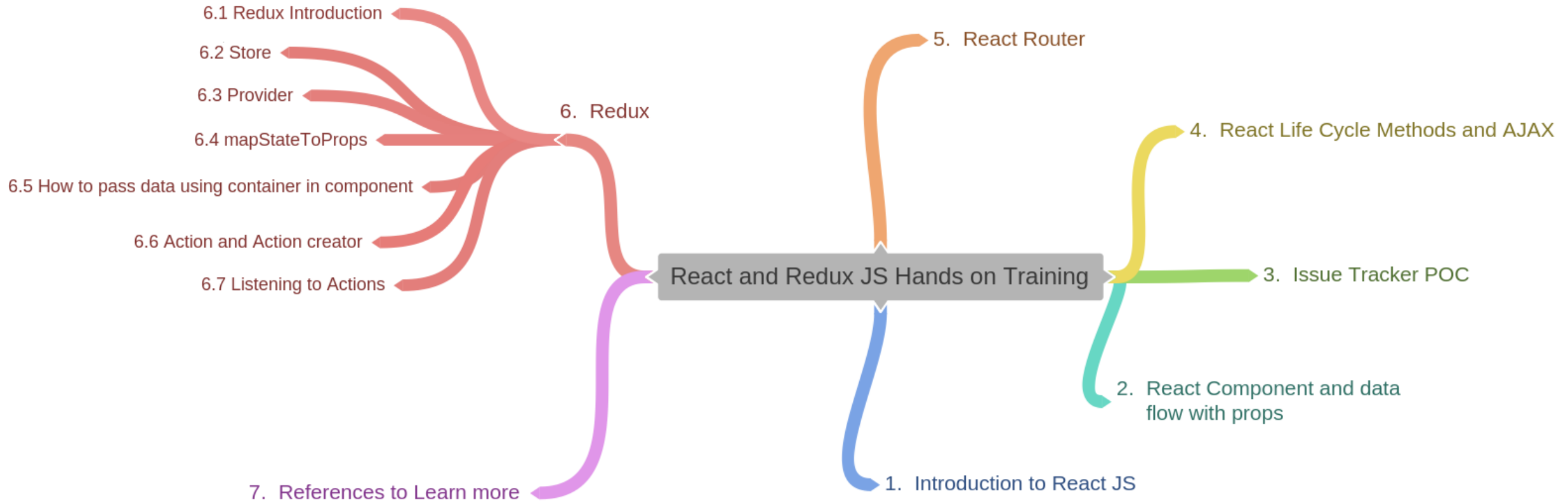
1. Will create POCs that will help to understand working of React and Redux
2. Hands on training
3. Less theory and more practical
4. Practical Assignments to get more understanding on concepts



Day -1

React JS Basic Concepts

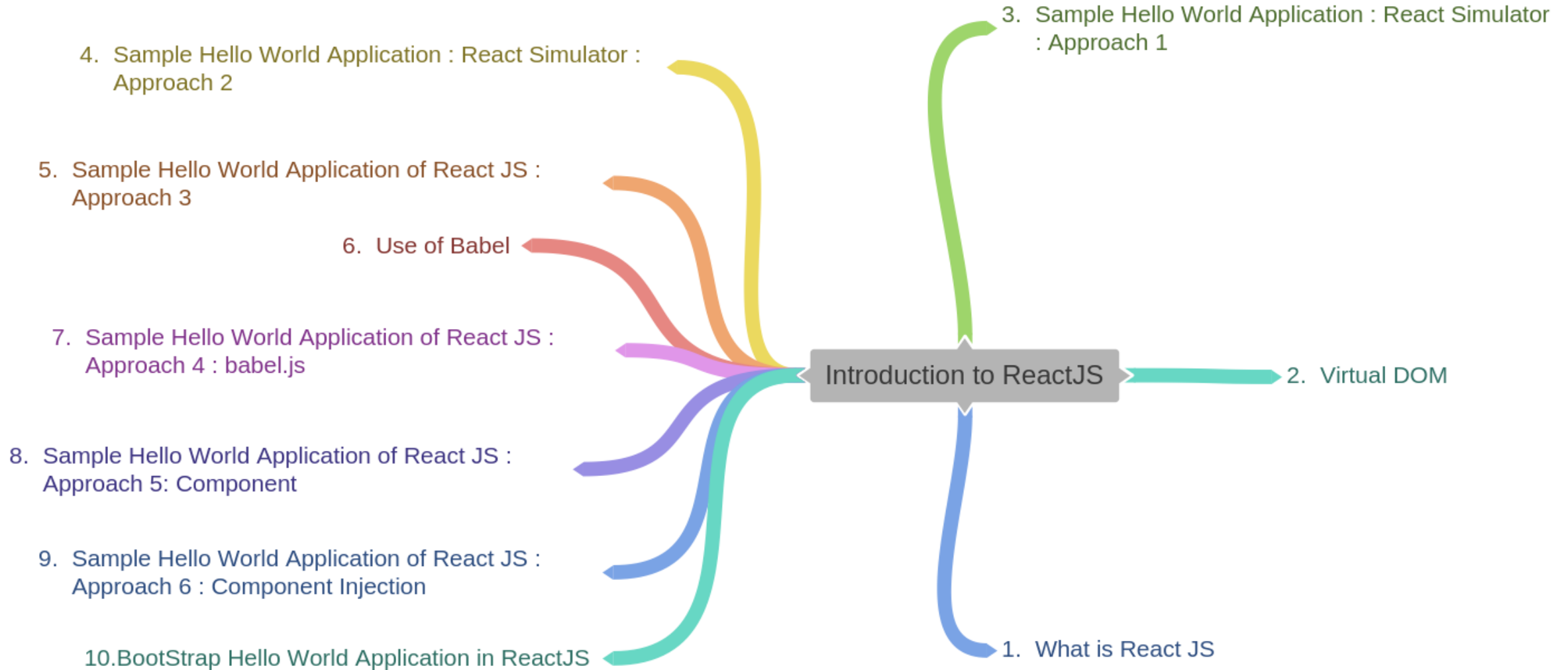
1. HTML, CSS Basic Understanding
2. JavaScript – Good Level of understanding
3. ES6 – Good Level of understanding



- Sublime Text or Bracket
- Visual Studio Code
- NodeJS



Introduction To React JS





What is React JS

Q : Is it a framework? [Ref : <https://reactjs.org/>]

There are certain points that need to be remembered while talking about ReactJS?

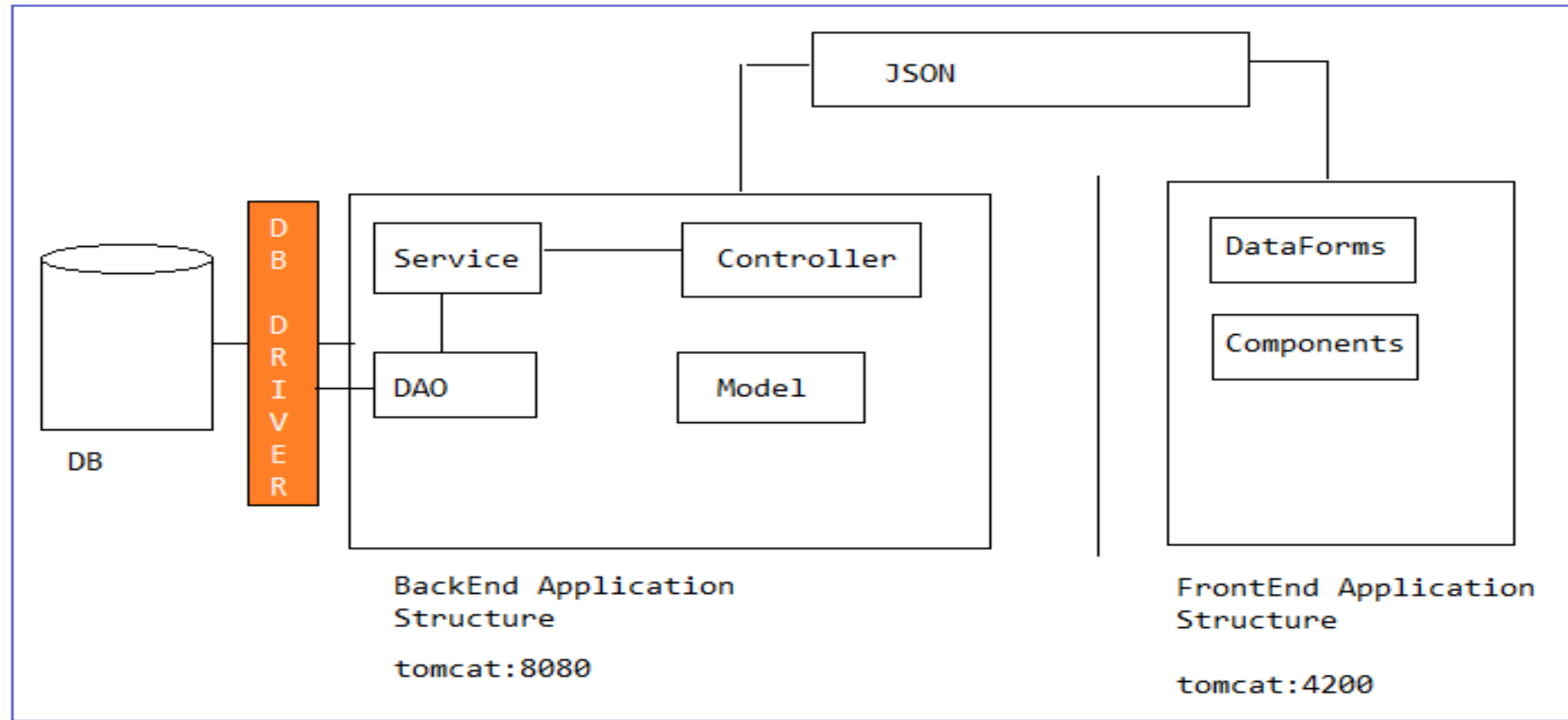
- ✓ It is a view in the application
- ✓ It recommends breaking down the pages or features in the small pieces of components
- ✓ Breaking down features in small pieces will offer reusability
- ✓ It is a declarative library rather than imperative
- ✓ It has a virtual DOM feature that makes the DOM rendering fast.
- ✓ It offers a component based model

>> What is ReactJS –[View in the application]

13

MVC – [Model View Controller]

Model and Controller is available at backend side only view is exposed to React side



“It recommends breaking down the pages or features in the small pieces of components”

The screenshot shows the ReactJS homepage. The top navigation bar includes the React logo, links to Docs, Tutorial, Community, and Blog, a search bar, and version information (v16.4.1) and GitHub link. The main section features the React logo and the tagline 'A JavaScript library for building user interfaces', with buttons for 'Get Started' and 'Take the Tutorial'. Below this, three columns describe React's features: Declarative, Component-Based, and Learn Once, Write Anywhere.

React
A JavaScript library for building user interfaces

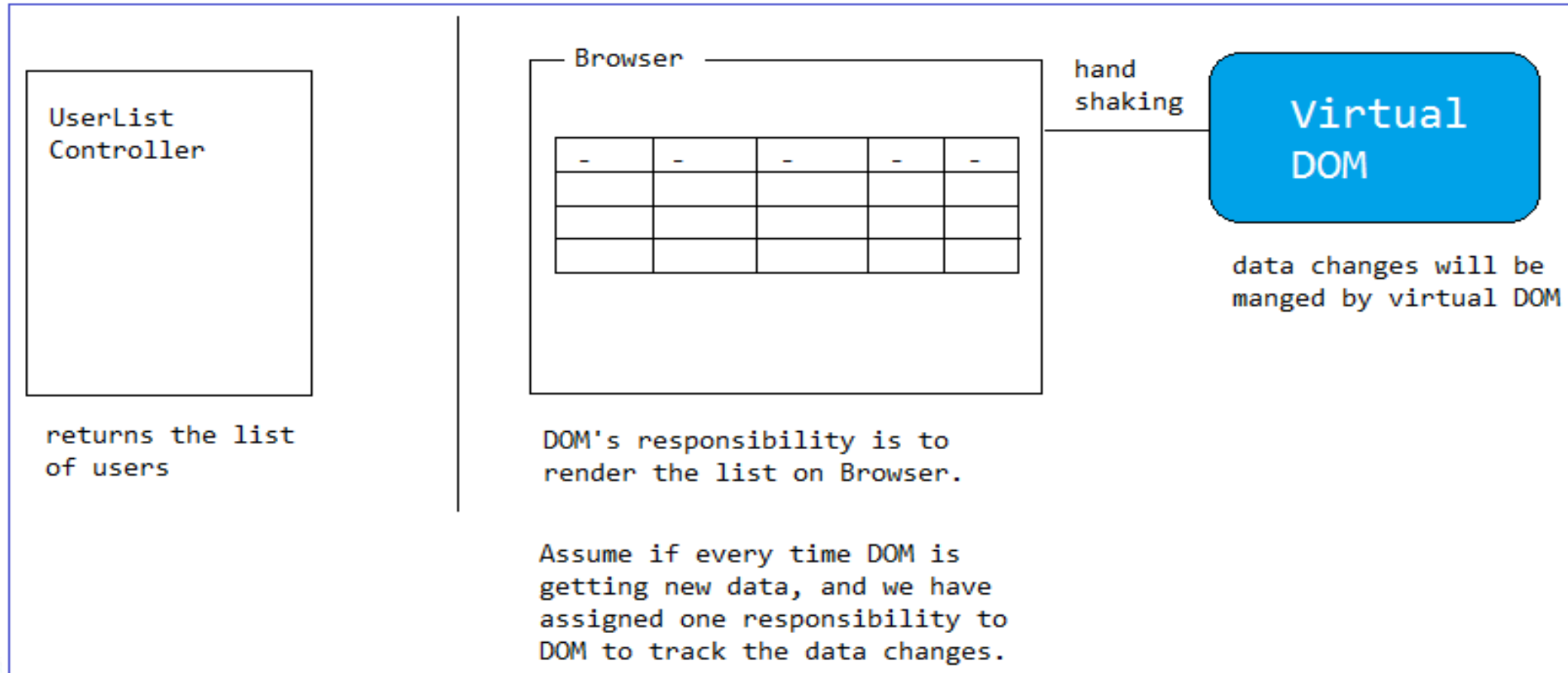
[Get Started](#) [Take the Tutorial >](#)

Declarative
React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.
Declarative views make your code more predictable and easier to debug.

Component-Based
Build encapsulated components that manage their own state, then compose them to make complex UIs.
Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere
We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.
React can also render on the server using Node and power mobile apps using [React Native](#).

“Virtual DOM is the helper to DOM” – [View This](#)



Look for a sample app on <https://reactjs.org>



Time Allotted : 5 mins

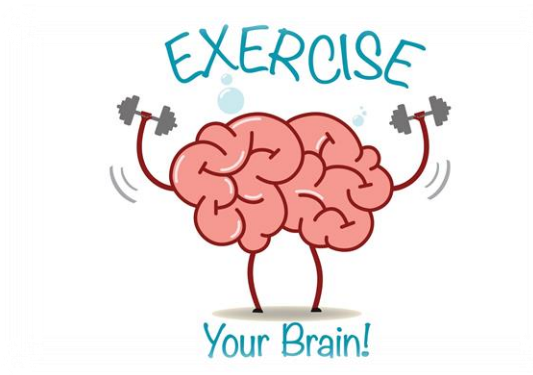
Start

You need to create a sample application to simulate ReactDOM.render() functionality. In approach 1 you need to create render functionality only?

Your challenge is:

- Not to include any dependency of ReactJS
- Use Pure Java Script to achieve this functionality

Note : Follow Trainer for Basic Structure : day01/challenge/index01.html



Time Allotted : 10 mins

Start

You need to create a sample application to simulate ReactDOM.render() functionality. In approach 2 you need to create dom.render() functionality?

Your challenge is:

- Not to include any dependency of ReactJS
- Use Pure Java Script to achieve this functionality

Note : Follow Trainer for Basic Structure : day01/challenge/index02.html

Here in this code snippet we will be actually using the react libraries to demonstrate the functionality that we have already created in last two approaches.

For achieving this we need to download [react.js](#) and [react-dom.js](#)

Refs: <https://reactjs.org/docs/cdn-links.html>



Time Allotted : 5 mins

Start

You need to create a sample application and now you will be including react.js and react-dom.js libraries.

Create a separate folder with “js” name and include both the libraries there.

Your challenge is:

- Use react and react-dom.js libraries
- Use ReactDOM.render() method to achieve the functionality

Note : Follow Trainer for Basic Structure : day01/challenge/index03.html

- Babel is a JavaScript compiler
- It will compile your ES6 or ES2015 code into equivalent ES5 code
- In React you will be writing ES6 code, that will be automatically compiled in ES5

How to use

```
// HTML Code
<script src="babel.min.js"></script>
<script type="text/babel">
  // ES6 code
</script>
```

You need to add babel.js library in your application's js folder.

Ref: <https://github.com/babel/babel-standalone>

Go to Installation section and copy and paste the babel.min.js link in browser and save it to the desired location by babel.min.js name.



Time Allotted : 5 mins

Start

Open index04.html file. In the render method enclose what to render parameter in h3 tag.

Your challenge is:

- Include babel.min.js in your file
- Script should understand babel js

Note : Follow Trainer for Basic Structure : [day01/challenge/index04.html](#)



Time Allotted : 5 mins

Start

Open index05.html file. Achieve the same functionality without using babel.

Your challenge is:

- Do not include babel.min.js
- Achieve the same result without babel conversion

Note : Follow Trainer for Basic Structure : [day01/challenge/index05.html](#)

- Creating component using Component
- Creating component using functional approach

According to Reactjs.org “Components are like javascript functions. They accept input (called props) and return React element describing what should appear on the screen”.

Ref: <https://reactjs.org/docs/components-and-props.html>

Different approaches for creating components as per the requirement:

- Using Component i.e. provided by ReactJS
- By creating function

Component helps to split the UI into independent and reusable pieces, and think about each piece in isolation.



Time Allotted : 5 mins

Start

Open index06.html file.

You need to create a Component and ReactDOM.render() method should render component.

Your challenge is:

- See the challenge in **index06.html** file

Note : Follow Trainer for Basic Structure : [day01/challenge/index06.html](#)



Time Allotted : 5 mins

Start

Open index07.html file.

You need to create a functional component and ReactDOM.render() method should render component.

Your challenge is:

- See the challenge in **index07.html** file

Note : Follow Trainer for Basic Structure : [day01/challenge/index07.html](#)

- Calling one component inside another component.

```
Hello  
return "<h2>Hello</h2>"
```

```
World  
return "<h2>World</h2>"
```

```
HelloWorld  
return "<div>  
    <Hello/>  
    <World/>  
</div>"
```

Output

Hello
World



Time Allotted : 5 mins

Start

Open index08.html file.

You need to create HelloWorld top component, Hello and World two more separate component.

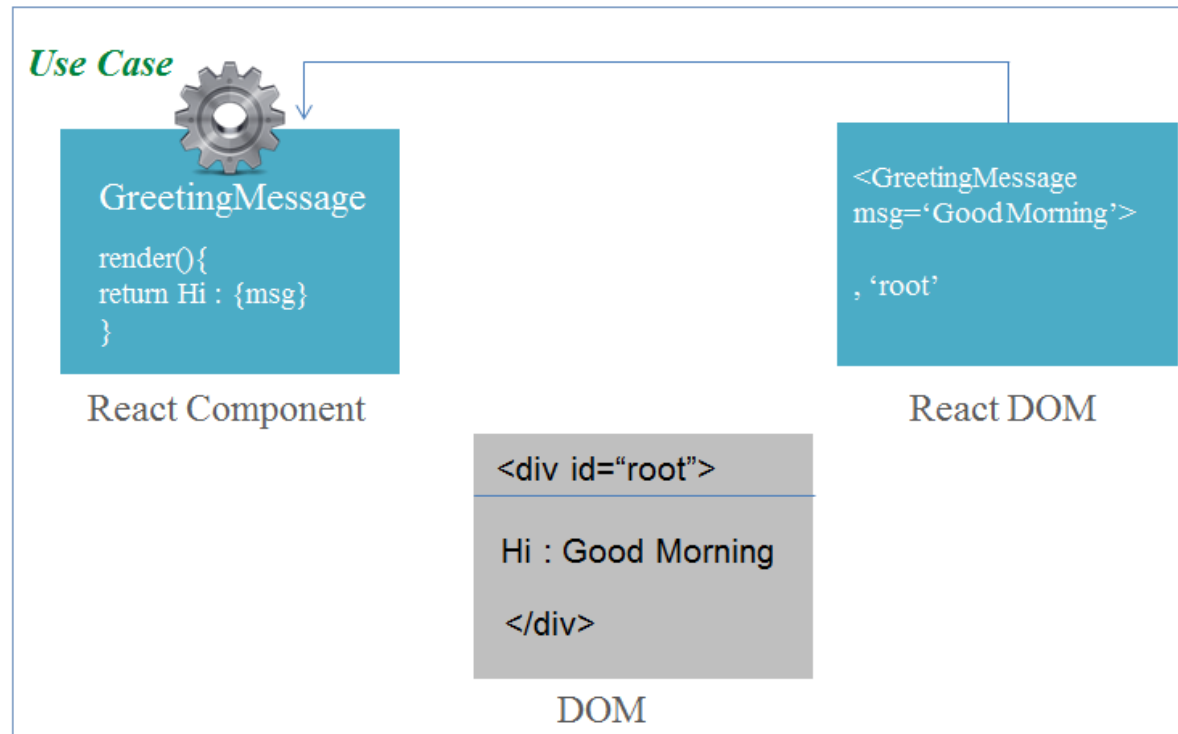
Your challenge is:

- See the challenge in **index08.html** file

Note : Follow Trainer for Basic Structure : [day01/challenge/index08.html](#)

React Component props is an argument to React Component

- Class Components may have state and props
- Functional components will have props not state





Time Allotted : 5 mins

Start

Open index09.html file.

You need to create GreetingComponent, that should return the props that is provided to this component.

Your challenge is:

- See the challenge in **index09.html** file

Note : Follow Trainer for Basic Structure : [day01/challenge/index09.html](#)



Bootstrapping helloworld app

31

Create React App is a new officially supported way to create single-page React applications. It offers a modern build setup with no configuration.

Note : working with React using create-react-app you will need node and npm in your machine. Download and install if you don't have.

Installation

```
npm install -g create-react-app
```

Creating an app

```
create-react-app hello-world
```

Starting the server

```
npm start
```

Let's understand the file structure



Time Allotted : 5 mins

Start

Open **hello-world** project in bracket.
Make below changes in your application

Changes in hello-world app

- App.css : let only .App selector style in App.css, remove other code
- App.js: return **<h1>Welcome in React JS HelloWorld App</h1>** in top most <div> tag

Note : observe the changes



Time Allotted : 15 mins

Start

Open assignment.md file from day01/challenge/assignment.md

- read the instructions
- working file is assignment01.html

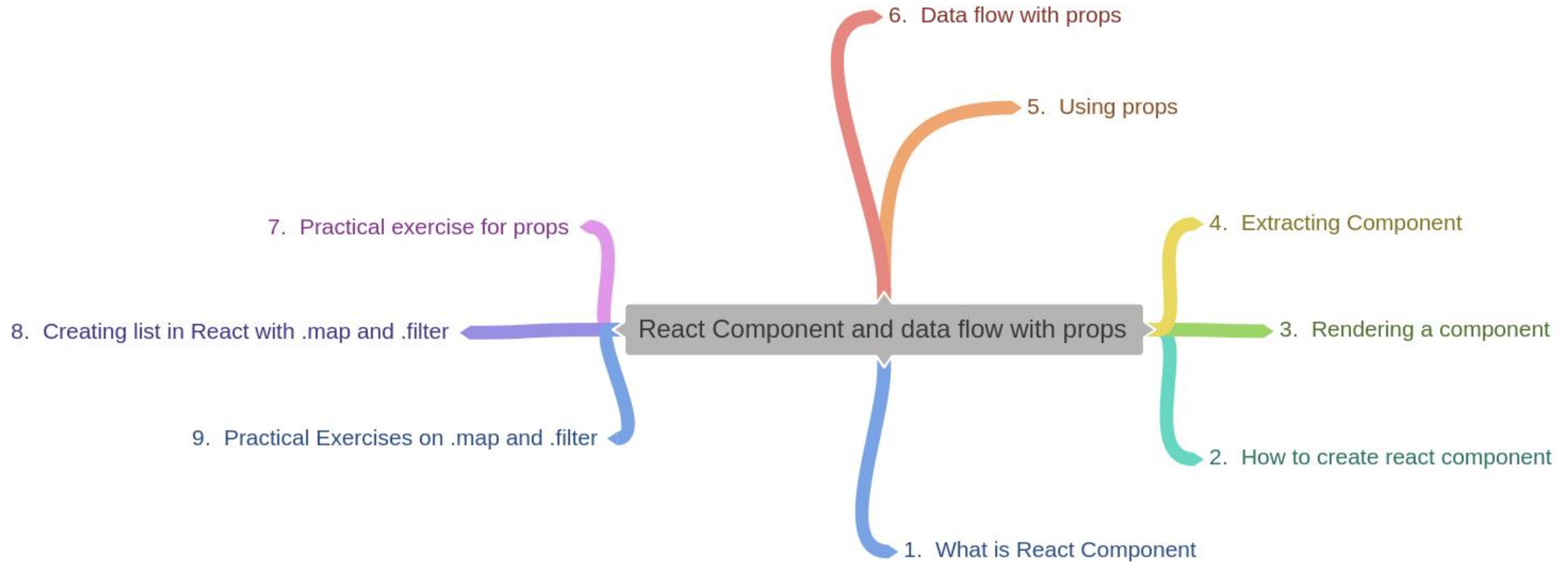
Changes in hello-world app

- Use all the knowledge you got in previous sessions, and complete the assignment.
- No need to create a separate file structure, styling is not needed.
- Practice for component creation.

Bonus points for:

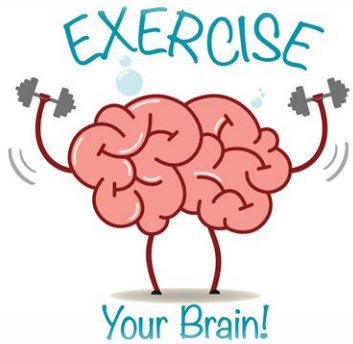
- Different type of components as per the requirements

Note : Follow Trainer for Basic Structure : day01/challenge/assignment01.html



We will see the nature of React Component and will cover the details around creating React Component

- React Component is typically a single view of a user interface that is divided up into logical parts or branches.
- The tree becomes the starting component (i.e. layout component)
- Each branch will become the sub-component that can be divided further into sub-components.
- Advantages will be organized UI and allows data and state changes to logically flow from the tree to branch, then sub branches.



Time Allotted : 10 mins

Start

LOGO

HOME | ABOUTUS | CONTACT

id	name	task	assignedby	status	operation
1	varun	UI imp	pankaj	pending	Edit Delete

#1 : Open hello-world app in visual Studio code.

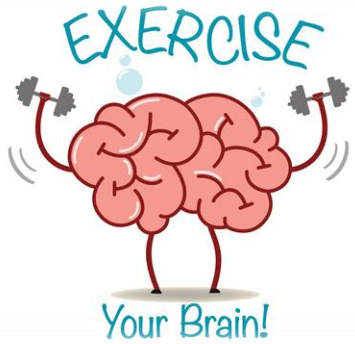
#2 : As you can see in diagram, We may have different parts like Header (Logo, Navigation), Search Form and data table.

#3 : First Create placeholder and check for output.

#4 : Code in App.js to return the UI

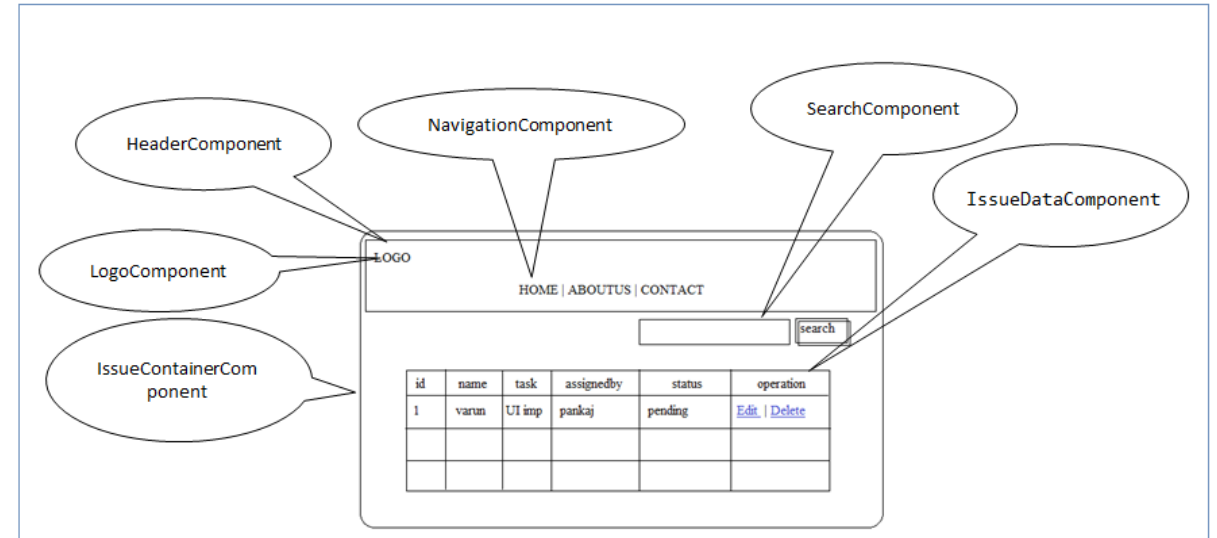
#5 : render App component in index.js file

Note : day01/hello-world/ App.js



Time Allotted : 10 mins

Start



#1 : Visualize each part as a separate component.

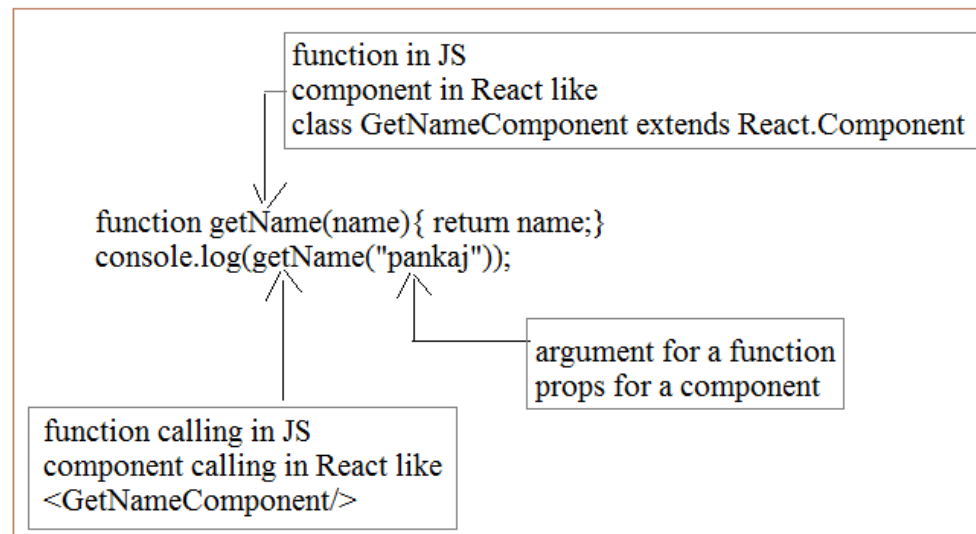
#2 : Create Separate Component, extract the appropriate code from the App component

#3 : Replace the extracted code with appropriate component.

Note : day01/hello-world/ [different components]

- We know how to create component
- Now we will see how to pass information to our component.
- Passing information to component is just like passing information to function.
- React is good at managing state, because there is a simple system for passing data from one component to another child component and that system is through props.
- **Props work for the component same as arguments work for function**

#Assignment : Create a function in JavaScript to accept name of the user and return the provided name, verify your function in console.log(). Use Browser Console for the same.



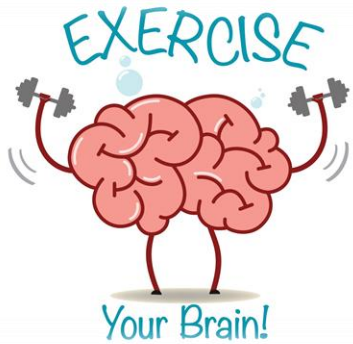
Create **GetNameComponent**, that should display the Hello! <User Name> as the name of user is provided.

```
import React from 'react';
class GetNameComponent extends React.Component{
  render(){
    return(
      <h2>Hello ! {this.props.name}</h2>
    );
  }
}
export default GetNameComponent;
```

When we use the component, we pass the name as attribute. This attribute can be accessed inside the component using **this.props.name**

>> Assignment 02 : props assignment

40



Time Allotted : 10 mins

Start



Name : Pankaj Sharma

UserName : pankaj

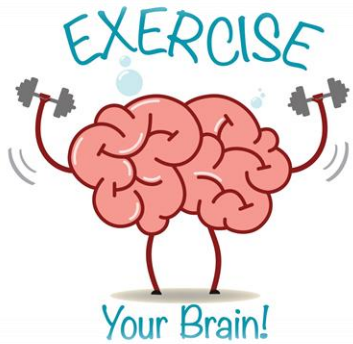
#1 : Create one BadgeComponent as shown above.

#2 : Use profile picture that should be available in your application. No direct url of the image is accepted in this assignment.

#3 : Image, name and username should be passed as props.

Hint : think about static site

Note : day01/hello-world/ [component]



Time Allotted : 10 mins

Start



Name : Pankaj Sharma

UserName : pankaj

#1 : Output should be same but here approach is different.

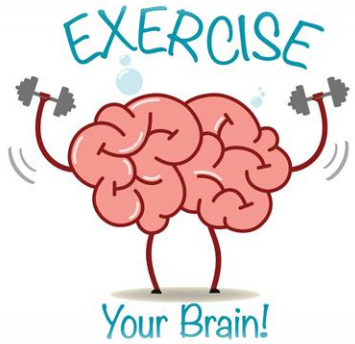
#2 : Here we have three information about user, his profile, name and username. Better approach will be when all the related information will be together. We can create one USER_INFO object that should hold user related information, and then this object should be passed as props in component.

Hint : think about how to create object in javascript

Note : day01/hello-world/ [component]

>> Assignment 04 : props assignment improvisation

42



Time Allotted : 10 mins

Start



Name : Pankaj Sharma

UserName : pankaj

#1 : Output should be same but here approach is different.

#2 : This time you need to convert Image, Name and UserName section in the respective components.

Note : Try to use props main component as well as sub components.

Note : day01/hello-world/ [component]

#Assignment: Create a basic JavaScript code that should add 10 in each number of the array number. [use browser console.]

```
var number = [10,20,30,40,50];
```

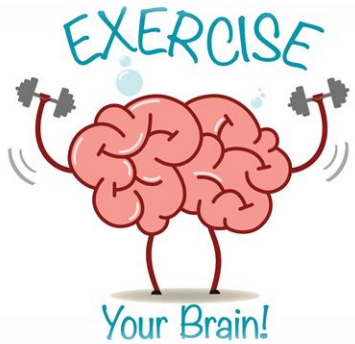
Output must be
20,30,40,50,60

```
var numbers=[10,20,30,40,50];  
var numberplusten=numbers.map(function (num){  
    return num+10;  
});  
console.log(numberplusten);
```

Know more about map here

<https://www.discovermeteor.com/blog/understanding-javascript-map/>

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map



Time Allotted : 10 mins Start

We want to show the list of tasks that are added by user.

```
TaskContainerComponent  
userName = "Pankaj Sharma";  
  
tasks = ['Creating React POC', 'Working on Project', 'Attending Meeting'];  
  
return (  
  <div>  
    Task Created By :  
    <ShowTaskList taskList={tasks}>  
  </div>  
)
```



```
ShowTaskList  
//Read tasklist props and return Unordered list of tasks as UI.
```

Tasks Created By : Pankaj Sharma

- Creating React POC
- Working on Project
- Attending meeting

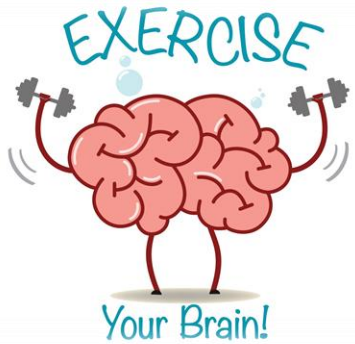
Note : day01/hello-world/ TaskContainerComponent

.filter is same as **.map**, but instead of returning new array after you've modified each item in the array. **.filter** allows you to filter out certain items in an array.

For example we want to display the list of all active users, this list of users who would have been activated by admin.

```
var users = [  
  {name: 'Pankaj Sharma', status: '1'},  
  {name: 'Priyansh Sharma', status: '1'},  
  {name: 'Manvi Sharma', status: '2'}  
];
```

```
users.filter(function(user){  
  return user.status == 1;  
});
```



Time Allotted : 10 mins

Start

Your task is to display the list of only active users. There should be two components. **UserListContainerComponent** that will hold the list of users, and **ShowActiveUserComponent** that will return the list of only active users out of total users.

Note : No Solution provided

- **Introduction to React JS**
 - Virtual DOM
 - Different Approaches for Hello World app
 - Use of Babel JS
 - Bootstrapping React App
- **React Component and Data flow with props**
 - What is React component
 - Rendering and Extracting React Component
 - Use of props and data flow with props
 - Practical exercises on props
 - .map and .filter methods for list
 - Practical exercises on .map and .filter

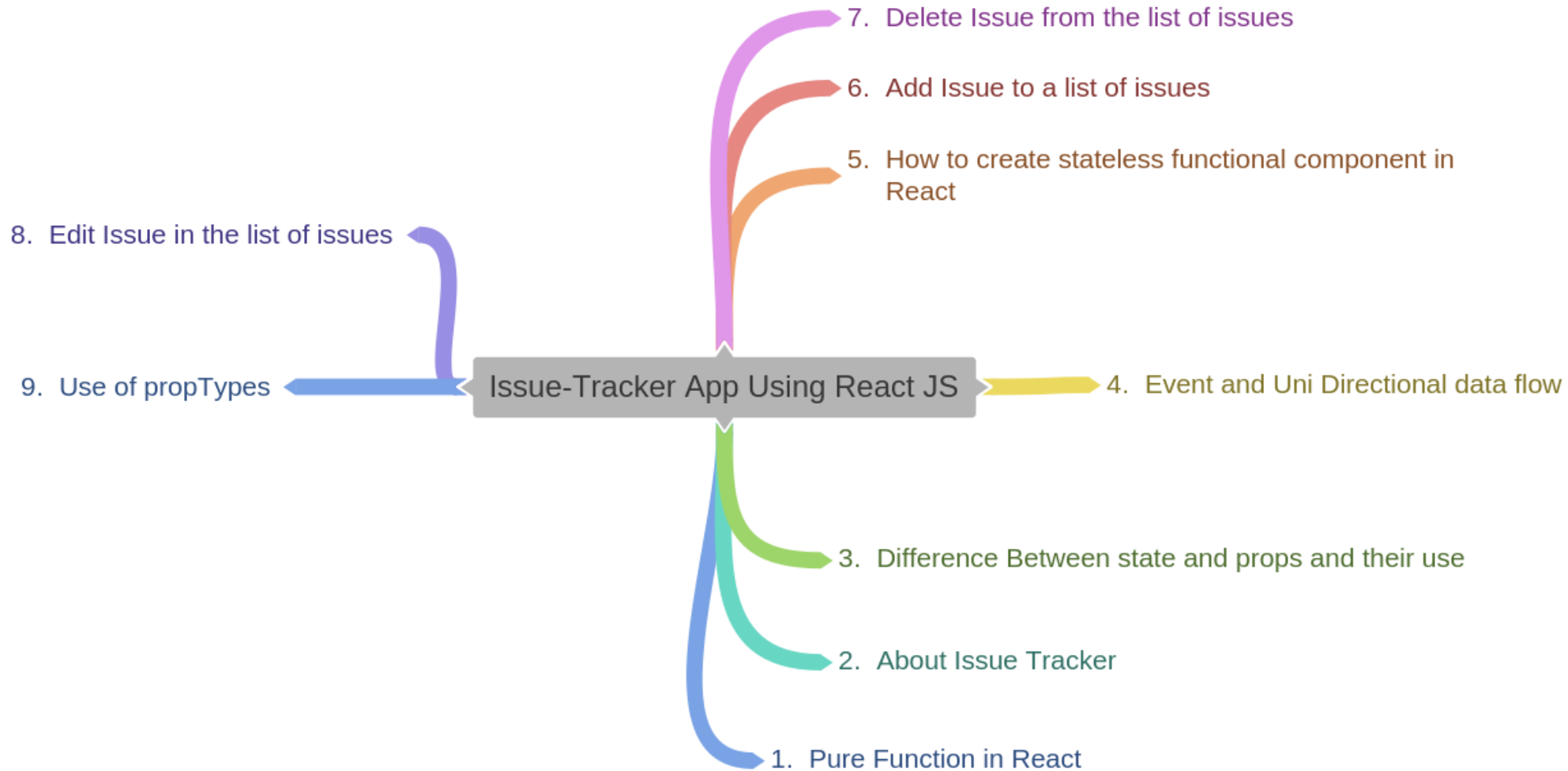




Day -2

Issue Tracker POC using all the concepts
of ReactJS

1. ReactJS Basics or DAY -1 Complete Training





Time Allotted : NA

1. Create issue-tracker app using **create-react-app** command in **day02** folder.
2. Run your application using **npm-start**
3. Open issue-tracker app in visual studio

Pure function ensures the consistency and predictability because of the below characteristics.

- Pure function always returns the same result given the same arguments.
- Pure function's execution doesn't depend on the state of the application.
- Pure function don't modify the variable outside of their scope.

```
> function sum(x,y){  
  return x+y;  
}  
< undefined  
  
> sum(10,20);  
< 30  
  
> var numbers=[1,2,3,4,5];  
< undefined  
  
> numbers.slice(0);  
< ▶ (5) [1, 2, 3, 4, 5]  
  
> numbers.slice(0,1);  
< ▶ [1]  
  
> numbers.slice(0,2);  
< ▶ (2) [1, 2]  
  
> numbers.splice(0,1);  
< ▶ [1]  
  
> numbers  
< ▶ (4) [2, 3, 4, 5]
```

Refrence : <https://tylermcginnis.com/building-user-interfaces-with-pure-functions-and-function-composition-in-react-js/>

We will create an application which will allow us:

- To add issues
- Show the list of issues
- Edit the issue in separate edit view
- Will give us functionality to toggle the status of the issue.

Issue Tracker App

New Task -2	Submit
• Create React Demo App	X Edit Issue
• Create PPT for ReactApp	X Edit Issue
• Create a POC in React and Redux	X Edit Issue
• New Task -1	X Edit Issue

- State is the place where the data comes from.
- Keep the state as simple as possible.
- You should minimize the number of statefull components. For example if you have ten components that need data from ten state, you should create one container component that will keep the state for all of them.
- Let us Create DemoComponent with demoText as state and render this.



Time Allotted : 5 Mins

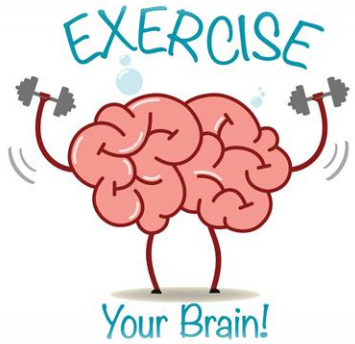
1. Create a HeaderComponent
2. It should have state, that should hold headerText as “**YASH Issue Tracker App**” and logo as “**YITS**” [Yash Issue Tracker System]
3. This component should render headerText and logo
4. Call this component on index.js file

Issue-tracker/src/HeaderComponent.js

```
-----  
import React from 'react';  
class HeaderComponent extends React.Component{  
  constructor(){  
    super();  
    this.state={  
      headerText:"Yash Training Management App",  
      "logo":"YTMS"  
    }  
  }  
  render(){  
    return(  
      <div>  
        <h1>{this.state.headerText}</h1>  
        <h3>{this.state.logo}</h3>  
      </div>  
    );  
  }  
}  
export default HeaderComponent;
```

Issue-tracker01/src/index.js

```
-----  
import React from 'react';  
import ReactDOM from 'react-dom';  
import HeaderComponent from './headercomponent'  
  
ReactDOM.render(<HeaderComponent/>,document  
  .getElementById("root"))
```



Time Allotted : 5 Mins

Req : Show one issue in unordered list

1. Create one **IssueListComponent**
2. It should have one issue in state that should have data as “Create React Demo App.”
3. This component should render that one list item in the unordered list.
4. Call **IssueListComponent** from ReactDOM.render().

```
Issue-tracker/src/IssueListComponent.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
import HeaderComponent from './headercomponent';
class IssueListComponent extends React.Component{

  constructor(){
    super();
    this.state={
      issue:"Create React Demo App"
    }
  }
  render(){
    return(
      <ul>
      <li>{this.state.issue}</li>
      </ul>
    )
  }
}

-----
ReactDOM.render(<IssueListComponent/>,document.getElementById("root"));
```



Time Allotted : 5 Mins

Req : Pass the issue name in child component and child component should return the list item.

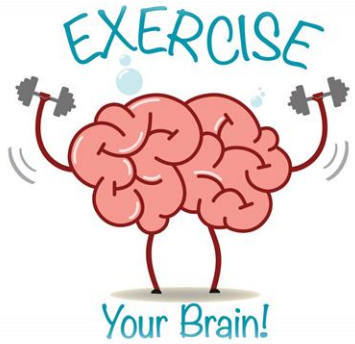
1. Create a separate IssueNameComponent
2. Pass the state of IssueListComponent to IssueNameComponent.
3. IssueNameComponent Should return name of the issue that is received from IssueListComponent in `` tag
4. IssueListComponent will render IssueNameComponent instead of direct list item.

Issue-tracker/src/IssueListComponent.js

```
-----  
import React from 'react';  
import ReactDOM from 'react-dom';  
import HeaderComponent from './headercomponent.jsx';  
class IssueListComponent extends React.Component{  
  constructor(){  
    super();  
    this.state={  
      issue:"Create React Demo App!"  
    }  
  }  
  render(){  
    return(  
      <ul>  
        <IssueNameComponent issue={this.state.issue}/>  
      </ul>  
    )  
  }  
}
```

Continue. . .

```
-----  
class IssueNameComponent extends  
  React.Component{  
  render(){  
    return(  
      <li>{this.props.issue}</li>  
    )  
  }  
}  
  
ReactDOM.render(<IssueListComponent/>,document.  
  getElementById("root"));
```



Time Allotted : 5 Mins

Req : Create Issue list, iterate it in parent component and pass the issue name to child component. Child component will return the name of the issue. On page issue list should be displayed.

1. Create issues array in IssueListComponent and add 4 issues in that.
2. In the ul tag all issues should be iterated, and IssueNameComponent should be called with issue as props.
3. Hint : use of .map function.

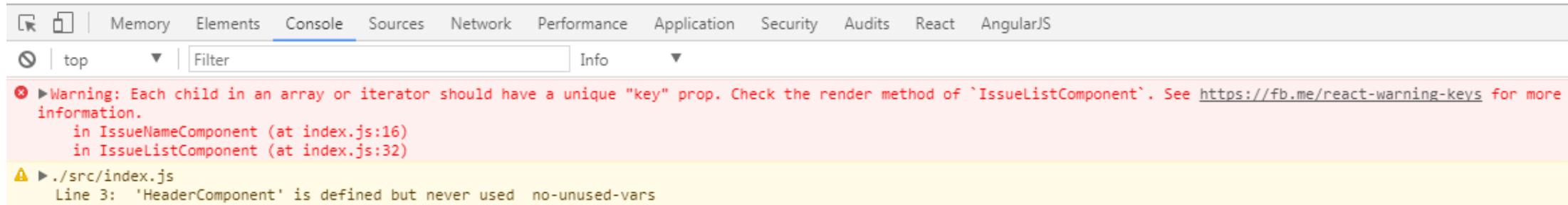
```
Issue-tracker/src/IssueListComponent.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
import HeaderComponent from './headercomponent.jsx';
class IssueListComponent extends React.Component{
  constructor(){
    super();
    this.state={
      issues:["Create React Demo App","Create PPT for
      ReactApp","Create a POC in React and Redux"]
    }
  }
  render(){
    return(
      <ul>
      {
        this.state.issues.map(function(issue){
          return <IssueNameComponent issue={issue}/>
        })
      }
      </ul>
    )
  }
}
```

```
Continue. . .
-----
class IssueNameComponent extends
React.Component{
  render(){
    return(
      <li>{this.props.issue}</li>
    )
  }
}

ReactDOM.render(<IssueListComponent/>,documen
t.getElementById("root"));
```



If you refresh the browser, you must get the changes. But there is one issue. Open the browser developer tool and check the console.



This warning is coming because React DOM will check which component need to be rendered based on some unique checking. This warning message can be removed by adding the key attribute while displaying the array element.

```
Issue-tracker/src/IssueListComponent
-----
{
  this.state.issues.map(function(issue){
    return <IssueNameComponent key={issue} issue={issue}/>
  })
}
```

Key will add the uniqueness in record. In real apps, you will replace it with some backend code.

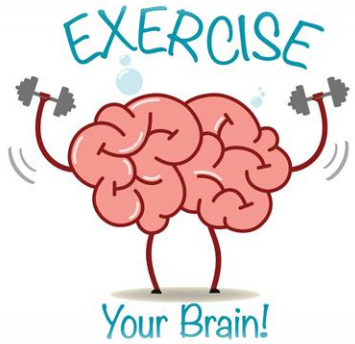
Event Ref : <https://facebook.github.io/react/docs/handling-events.html>

Req : Make small change in your application. Your issues list should be of object type with two properties. 1. name and 2. completed. Accordingly make changes in IssueListComponent and IssueNameComponent

```
Issue-tracker/src/IssueListComponent
-----
import React from 'react';
import ReactDOM from 'react-dom';
class IssueListComponent extends React.Component{
  constructor(){ super();
  this.state={
    issues:[
      {name:"Create React Demo App",completed:false},
      {name:"Create PPT for ReactApp",completed:false},
      {name:"Create a POC in React and
      Redux",completed:false}
    ]}
  render(){ return( <ul>
  {
    this.state.issues.map(function(issue){
      return <IssueNameComponent key={issue.name}
      issue={issue}/>
    })
  } </ul>
  )}}
```

```
Continue. . .
-----
class IssueNameComponent extends
React.Component{
  render(){
    return(
      <li>{this.props.issue.name}</li>
    )
  }
}

ReactDOM.render(<IssueListComponent/>,documen
t.getElementById("root"));
```



Time Allotted : NA

Req : Now we want to click on list item, and its completed state should be changed. We will also be changing its style. Here our purpose is to handle the event.

Note : child component can not directly change the state. To make any changes in state, child component will notify the parent component

Steps:

1: very first add one class 'completed' in tag of *IssueNameComponent* which will be added dynamically as the user click on item.

```
Issue-tracker/src/IssueListComponent
-----
class IssueNameComponent extends React.Component{
  render(){
    return(
      <li className={this.props.issue.completed ? 'completed' : ''}>
        {this.props.issue.name}
      </li>
    )
  }
}
```

Here if issue completed status is true then completed class will be added otherwise nothing will be added.



2: now create `changeStatus()` method in **IssueListComponent**. It should be created above `render()` method.

```
Issue-tracker/src/IssueListComponent
-----
class IssueListComponent extends React.Component{
  . . . .
  changeStatus(){
  }
  . . . .
}
```

- Now in this custom method we want to access `this.state`.
- Directly we can not access value of `this`, because its value is changed in custom method.
- To access the value of `this`, we will have to bind it with `changeStatus()` method in constructor it self.

```
Issue-tracker/src/IssueListComponent
-----
class IssueListComponent extends React.Component{
  constructor(){
    super();
    this.changeStatus=this.changeStatus.bind(this);
    this.state={
      issues:[
        . . . .
      ]
    }
  }
}
```

Now when the event will be triggered on element `changeStatus()` method should be invoked, for that we need to pass clicked element's index to `changeStatus()` method. We will be passing the event handler to child component using props.

First make changes to your `changeStatus()` method as below.

```
Issue-tracker/src/IssueListComponent
-----
class IssueListComponent extends React.Component{
  . . . .
  changeStatus(index){
    console.log(this.state.issues[index]);
  }
  . . . .
}
```

Here `changeStatus()` method is taking one index and out of clicked issues `console.log()` will display the information of clicked issue object with its completed status.

Add `clickHandler` attribute in <IssueNameComponent /> as below.

```
Issue-tracker/src/IssueListComponent
-----
{
  this.state.issues.map(function(issue){
    return <IssueNameComponent key={issue.name} issue={issue} clickHandler={this.changeStatus}/>
  })
}
```

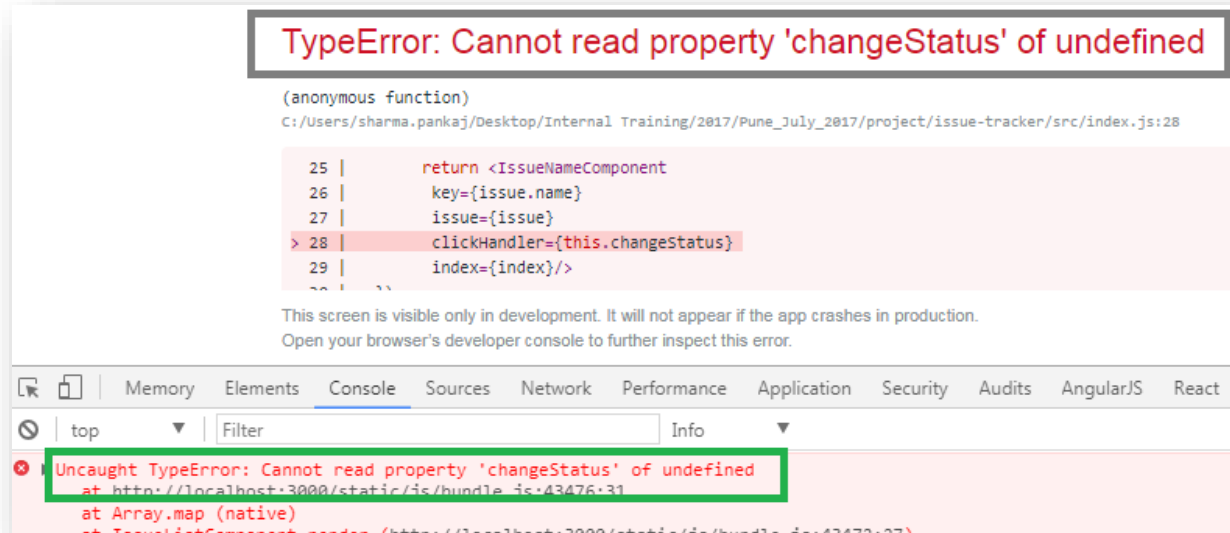
Let's add onClick() event on which will execute one function which will be calling clickHandler property provided by the parent component, here we will have to pass the index of clicked list element.

```
Issue-tracker/src/IssueNameComponent
-----
class IssueNameComponent extends React.Component{
  render(){
    return(
      <li onClick={()=>{
        this.props.clickHandler(this.props.index)
      }} className={this.props.issue.completed ? 'completed' : ''}>
        {this.props.issue.name}
      </li>
    )
  }
}
```

Now as you can see that we have added **this.props.index**, which should be passed as a props in <IssueNameComponent/>

```
Issue-tracker/src/IssueListComponent
-----
this.state.issues.map(function(issue, index){
  return <IssueNameComponent
    key={issue.name} issue={issue} clickHandler={this.changeStatus} index={index}/>
})
```

Now check your browser, you should get one error.

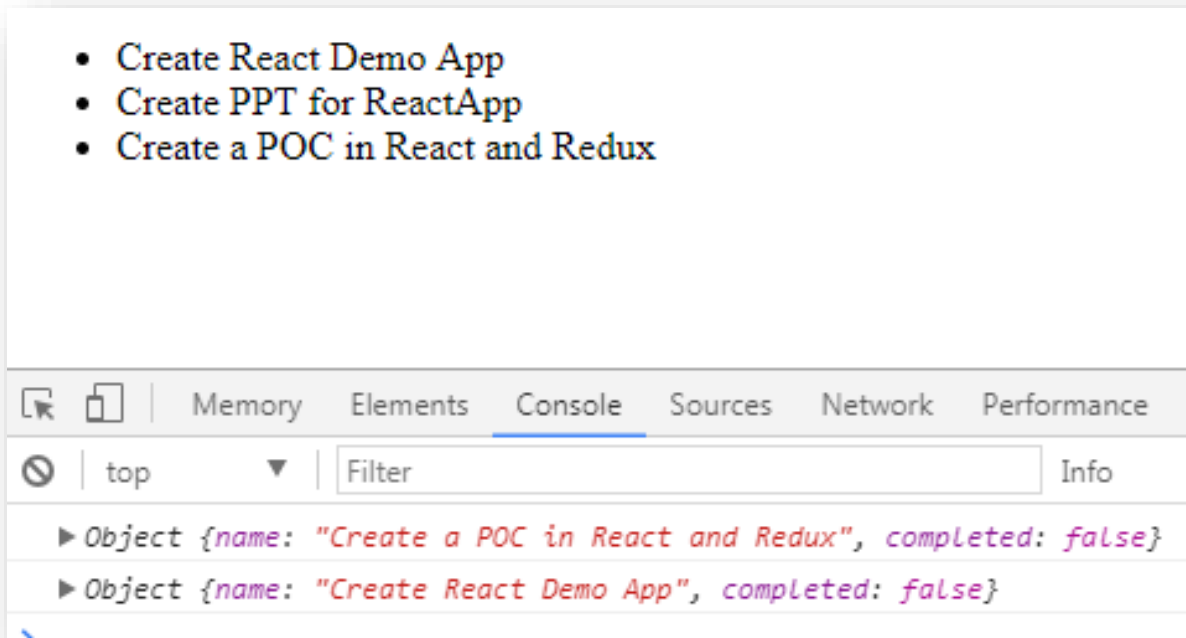


- This is because `this.changeStatus` is not accessible inside `.map(function(issue,index))`.
- To make it accessible you will have to use the `=>` function.
- Do the below changes in your code and check the browser.

```
Issue-tracker/src/IssueListComponent
-----
this.state.issues.map((issue,index)=>{
  return <IssueNameComponent
    key={issue.name} issue={issue} clickHandler={this.changeStatus} index={index}/>
})
```

- Now that problem will be resolved.

- Check on chrome browser.
- Open the browser console.
- Click on any list item.
- In console you should get which list item is clicked.



Uptill now we are able to get the clicked object.

Req : now we want to change the completed status of clicked list. We will have to implement the below logic in `changeState()` method.

- Copy all issues in separate issues variable.
- Get the clicked issue from issues.
- Change the completed status of clicked issue
- Change the complete list of issues by `setState`

```
Issue-tracker/src/IssueListComponent
-----
changeStatus(index){
  var issues=this.state.issues;
  var issue=issues[index];
  issue.completed=!issue.completed;
  this.setState({
    issues:issues
  })
  console.log(this.state.issues[index])
}
```

Open the browser, open browser console and now click on any list item twice, and check the completed status in browser.

We can remove `console.log()` from production code, as this is only for testing.



- Create React Demo App
- Create PPT for ReactApp
- Create a POC in React and Redux

```
▼ Object {name: "Create a POC in React and Redux", completed: true} ⓘ  
  completed: true  
  name: "Create a POC in React and Redux"  
  ▶ __proto__: Object  
▼ Object {name: "Create a POC in React and Redux", completed: false} ⓘ  
  completed: false  
  name: "Create a POC in React and Redux"  
  ▶ __proto__: Object
```

Req: now let us add some CSS to see the completed issues effect.

Add style for completed class in index.css

Import index.css in index.js file.

```
Issue-tracker/src/index.js
```

```
-----  
import './index.css'
```

```
Issue-tracker/src/index.css
```

```
-----  
.completed{  
text-decoration: line-through;  
color:#eee;  
}
```

Now check on browser.

- Create React Demo App
- Create PPT for ReactApp
- Create a POC in React and Redux

Req: uptill now we have two components and we have written both these components in one file. We want to organize the custom components in different location.

Keeping all components in one file is not a good approach.

- Create a **component** folder in src
- Create a file named as IssueNameComponent.js
- Copy IssueNameComponent from index.js file and paste in IssueNameComponent.js file
- Import React from 'react' in IssueNameComponent.js.
- Export IssueNameComponent in IssueNameComponent.js file.
- Import IssueNameComponent in index.js file

```
Issue-tracker/src/component/IssueNameComponent.js
-----
import React from 'react';
class IssueNameComponent extends React.Component{
  render(){
    return(
      <li onClick={()=>{
        this.props.clickHandler(this.props.index)
      }} className={this.props.issue.completed ? 'completed' : ''}>
        {this.props.issue.name}
      </li>
    )
  }
}
export default IssueNameComponent;
```

```
Issue-tracker/src/index.js
-----
import IssueNameComponent from './component/IssueNameComponent';
```

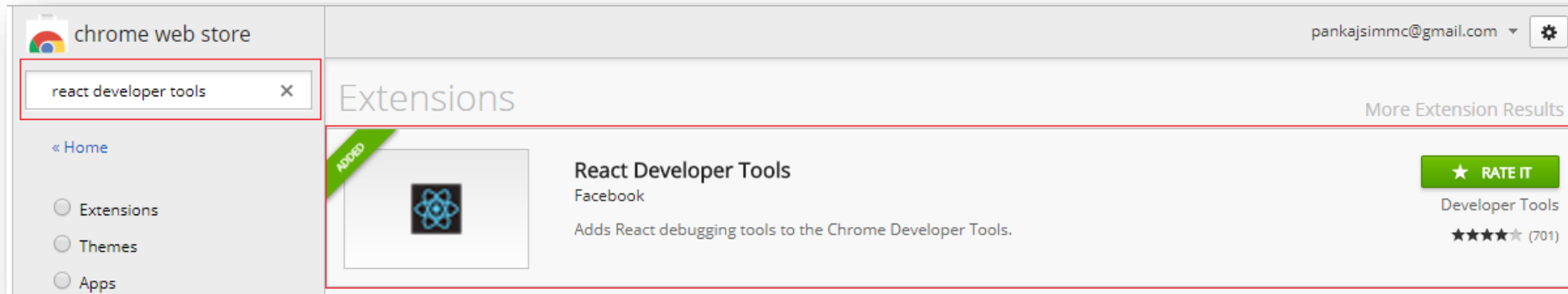
We have also used the bootstrap for better UI.

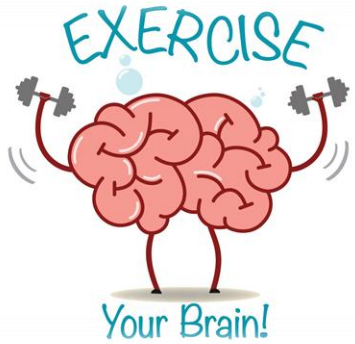
- Copy bootstrap cdn from bootstrap
- Add the link in index.html
- Wrap the root div in container div.

```
Issue-tracker/public/index.html
-----
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mi
n.css" >
<div class="container">
<div id="root"></div>
</div>
```

At this point we need to add one extension to chrome.

- Open chrome web extension from <https://chrome.google.com/webstore/category/extensions>
- Search for React Developer Tools
- Add the extension in chrome





Time Allotted : NA

Req : add IssueForm with text box and a submit button to add new issue.



Req : add IssueForm with text box and a submit button to add new issue.

- Create one IssueForm.js file in component
- Create one stateless IssueForm component with basic `<h1>Issue Form</h1>` return statement and export it.
- Import IssueForm on index.js and call `<IssueForm/>` before the code where Issues are listed.

```
Issue-tracker/src/component/IssueForm.js
-----
import React from 'react';
const IssueForm = () =>{
  return (
    <h2>Issue Form</h2>
  )
}
export default IssueForm;
```

```
Issue-tracker/src/index.js
-----
. . . . .
import IssueForm from './component/IssueForm.js';
import './index.css'
class IssueListComponent extends React.Component{
  . . . . .
  render(){
    return(
      <div>
        <h1>Issue Tracker App</h1>
        <hr/>
        <section>
          <IssueForm/>
          <ul>
            . . .
ReactDOM.render(<IssueListComponent/>,document.getElementBy
ntById("root"));
```




- Open the React Developer Tools
- Click on IssueListComponent, you will find the IssueForm component

Issue Tracker App

Issue Form

- Create React Demo App
- Create PPT for ReactApp
- Create a POC in React and Redux

▼ <IssueListComponent> == \$r

▼ <div>
 <h1>Issue Tracker App</h1>
 <hr />
 ▼ <section>
 ▶ <IssueForm>...</IssueForm> → child component
 ▼
 ▶ <IssueNameComponent key="Create React Demo App" issue={name: "Create React Demo App"} />
 ▶ <IssueNameComponent key="Create PPT for ReactApp" issue={name: "Create PPT for ReactApp"} />
 ▶ <IssueNameComponent key="Create a POC in React and Redux" issue={name: "Create a POC in React and Redux"} />

 </section>
</div>
</IssueListComponent>

parent component

child component

Props
Empty object

State
▶ issues: Array[3]

here in parent component we want a currentIssue state which can be passed to child component, and then child component will be able to update that state.

YASH L&D –India

l&dindia@yash.com

YASHSM
Technologies
More than what you think.



- Add `currentIssue:` as one more state in the state of `IssueListComponent`.
- Create `updateIssue()` method and bind with this ref same as `changeStatus`.
- `updateIssue()` method will receive one `newValue`.
- `newValue` in `updateIssue()` will modify the `currentIssue`.
- `currentIssue` state and `updateIssue()` method will be passed to `IssueForm` component using props.

```
Issue-tracker/src/IssueListComponent
-----
constructor(){
  super();
  this.changeStatus=this.changeStatus.bind(this);
  this.updateIssue=this.updateIssue.bind(this);
  this.state={
    issues:[. . . ],
    currentIssue:''
  }
}

updateIssue(newValue){
  this.setState({
    currentIssue:newValue
  })
}
```

```
Issue-tracker/src/IssueListComponent
-----
render(){
  return(
    . . . .
    <IssueForm
      currentIssue={this.state.currentIssue}
      updateIssue={this.updateIssue}/>
    . . . .
  )
}
```



- Open the React Developer Tool and observe the changes.

Issue Tracker App

Issue Form

- Create React Demo App
- Create PPT for ReactApp
- Create a POC in React and Redux

The screenshot shows the React Developer Tools interface. The left pane displays the component tree for `<IssueListComponent>`. The right pane shows the state of the `IssueListComponent`, which includes `currentIssue` and `issues`. Annotations with arrows point from the state to the corresponding props in the component tree.

```
<div>
  <h1>Issue Tracker App</h1>
  <hr />
  <section>
    <IssueForm currentIssue="" updateIssue=bound updateIssue()>_</IssueForm>
    <ul>
      <IssueNameComponent key="Create React Demo App" issue={name: "Create React Demo App"} />
      <IssueNameComponent key="Create PPT for ReactApp" issue={name: "Create PPT for ReactApp"} />
      <IssueNameComponent key="Create a POC in React and Redux" issue={name: "Create a POC in React and Redux"} />
    </ul>
  </section>
</div>
```

Props: Empty object

State:

- `currentIssue: ""`
- `issues: Array[3]`

in the state of IssueListComponent `currentIssue` added.

`currentIssue` and `updateIssue()` is passed in `<IssueForm />`



- Now we need to create <form> with textbox and button control in IssueForm component.
- Text box will have currentIssue as value, and updateIssue as method binding on onChange event.
- currentIssue and updateIssue() method will be accessible using props
- props will be available on IssueForm component as an argument.

```
Issue-tracker/src/component/IssueForm.js
```

```
-----
```

```
import React from 'react';
const IssueForm = (props) =>{
  return (
    <div>
      <form>
        <input type="text"
          value={props.currentIssue}
          onChange={props.updateIssue}
        />
      </form>
    </div>
  )
}
export default IssueForm;
```



- Go on browser
- Open React Developer Tool
- Type something in text box.

Issue Tracker App

[object Object]

- Create React Demo App
- Create PPT for ReactApp
- Create a POC in React and Redux

The screenshot shows the React DevTools interface. The top bar includes tabs for Memory, Elements, Console, Sources, Network, Performance, Application, and React. The React tab is active, showing the component tree. The selected component is `<IssueListComponent>`. The console shows a warning: `Warning: Can't perform a React state update on an unmounted component. This is a no-op, but it indicates a memory leak in your application. To fix, cancel all subscriptions and unsubscribe from lifecycle events. See https://reactjs.org/docs/unsafe-callbacks.html#unsafe-callback for more details.` The state of the component is displayed on the right, showing `currentIssue` as a `SyntheticEvent` and `issues` as an `Array`.

Instead of getting the value in CurrentIssue, we are getting SyntheticEvent{...}. this is because in the updateIssue(newValue) method, we are assigning newValue to currentIssue. This is not the value assignment, here newValue is an event. we need to assign the value of targeted source.



Add issues to list of issues

86

- Make below changes in your `updateIssue()` method, and then try to type something in text box.

```
Issue-tracker/src/IssueListComponent
-----
updateIssue(newValue){
  this.setState({
    currentIssue:newValue.target.value
  })
}
```

Issue Tracker App

- Create React Demo App
- Create PPT for ReactApp
- Create a POC in React and Redux

once you type something in text box. that much will be reflected. this will be varified by observing changes in the `currentIssue` state while typing.



- Now we will add the added issue in issues list.
- Create addIssue() method in IssueListComponent

```
Issue-tracker/src/IssueListComponent
-----
addIssue(){
}
```

- Bind the addIssue() method with “this” ref so that addIssue() method can access “this” ref.

```
Issue-tracker/src/IssueListComponent
-----
this.addIssue=this.addIssue.bind(this);
```

- Pass the addIssue() method as a prop to <IssueForm/> tag so that further in IssueForm component that can be available.

```
Issue-tracker/src/IssueListComonent
-----
<IssueForm
currentIssue={this.state.currentIssue}
updateIssue={this.updateIssue}
addIssue={this.addIssue}/>
```



- Add onSubmit event to form in IssueForm component and pass addIssue() method using props to onSubmit event.

```
Issue-tracker/src/component/IssueForm.js
-----
const IssueForm = (props) =>{
  return (
    <div>
      <form onSubmit={props.addIssue}>
        <input type="text"
          value={props.currentIssue}
          onChange={props.updateIssue}
        />
      </form>
    </div>
  )
}
```

- Create a submit button.
- Note : onSubmit event will be fired if you press Enter key or click the submit button.

```
Issue-tracker/src/component/IssueForm.js
-----
const IssueForm = (props) =>{
  return (
    <div>
      <form onSubmit={props.addIssue}>
        <input type="text"
          value={props.currentIssue}
          onChange={props.updateIssue}
        />
        <button type="submit">Submit</button>
      </form>
    </div>
  )
}
```




- Now add a `console.log('-----addIssue triggered-----');` statement in `addIssue()` method and click on submit or either press Enter, and see whether `addIssue()` method triggered or not.

```
Issue-tracker/src/IssueListComponent
-----
addIssue(){
  console.log('-----Add Issue Triggered-----');
}
```

Check the console, nothing will happen. This is because `addIssue()` method should have a ref of event triggered. We need one argument in `addIssue()` method.

```
Issue-tracker/src/IssueListComponent
-----
addIssue(event){
  console.log('-----Add Issue Triggered-----');
}
```

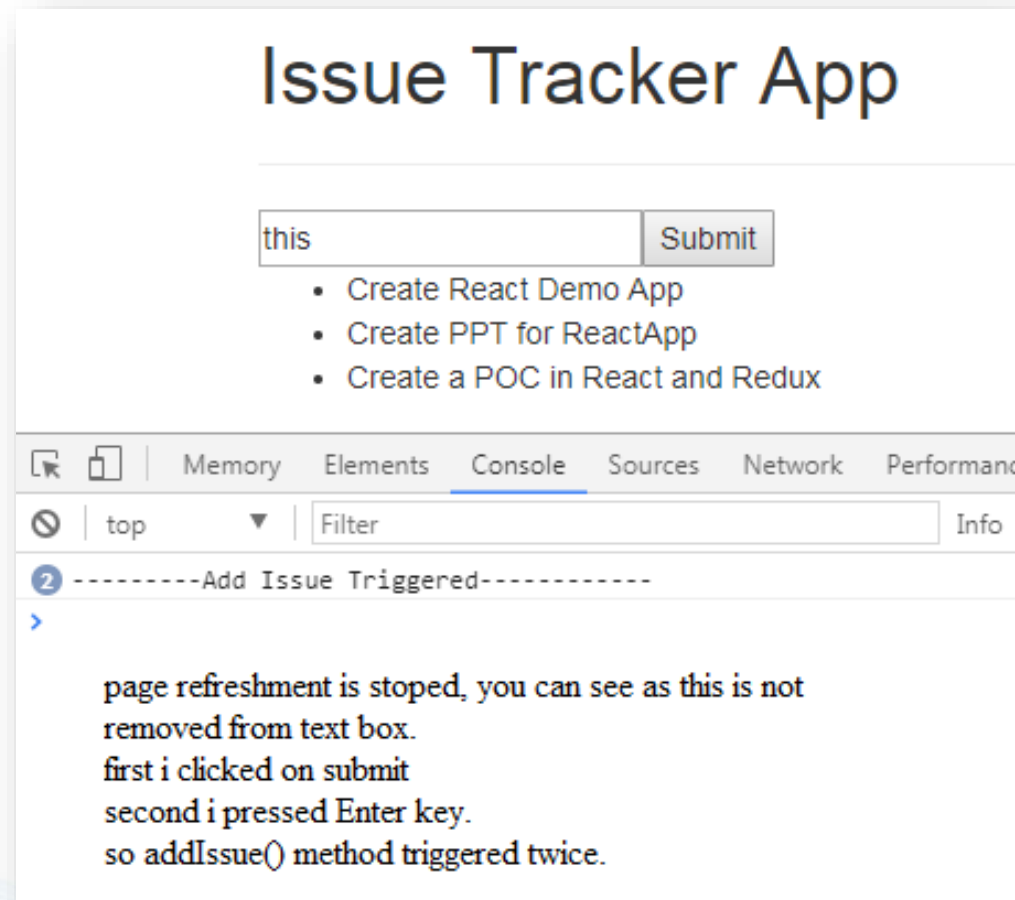
Write something in text box and click on submit. Still nothing `addIssue()` seems not to be triggered. But actually it is triggered, now the issue is that page is refreshed. We need to stop the page refresh.

Make below changes in your `addIssue()` method.

```
Issue-tracker/src/IssueListComponent
-----
addIssue(event){
  event.preventDefault();
  console.log('-----Add Issue Triggered-----');
}
```



- Now check the browser.





Req : We want to push the currentIssue in Issues array and would like to make currentIssue in default state.

```
addIssue(event){  
  event.preventDefault();  
  // console.log('-----Add Issue Triggered-----');  
  let issues=this.state.issues;   
  let currentIssue=this.state.currentIssue;  
  
  issues.push(  
    name:currentIssue,  
    completed:false  
  })  
  
  this.setState(  
    issues:issues,  
    currentIssue:''  
  )  
}
```

→ copying issues array into local issues variable.
→ copying currentIssue in local currentIssue variable.

→ create new object with currentIssue and completed status by default as false and push that object in local issues array

→ assign local issues array into issues state of IssueListComponent. and put the currentIssue in default state.

Now check your application, you should be able to add the new task and text box will be empty. You will be able to add task either by pressing Enter key or by pressing submit Button.



Time Allotted : NA

Req : we want to create a delete button with each issue, and when user click on delete button. the corresponding issue must be deleted.

Req: we want to create a delete button with each issue, and when user click on delete button. the corresponding issue must be deleted.

- Create deleteIssue() method in index.js file and bind with “this” reference.

```
Issue-tracker/src/IssueListComponent
-----
. . .
this.deleteIssue=this.deleteIssue.bind(this);
. . .
deleteIssue(){
}
```

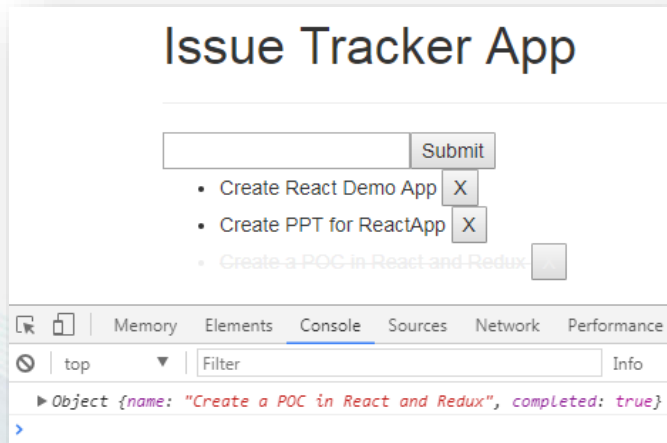
- We need to pass deleteIssue() to <IssueNameComponent> tag.

```
Issue-tracker/src/IssueListComponent
-----
return <IssueNameComponent
key={issue.name}
issue={issue}
clickHandler={this.changeStatus}
deleteIssue={this.deleteIssue}
index={index}/>
})
```

- Create a delete button in IssueNameComponent and apply onClick event on button which will trigger the deleteIssue() method.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
<li onClick={()=>{
  props.clickHandler(props.index)
}} className={props.issue.completed ? 'completed' : ''}>
  {props.issue.name}&nbsp;
  <button onClick={props.deleteIssue}>X</button>
</li>
```

- Now if you go on browser you will get the changes. Now click on Delete button.
- By clicking on delete button list item will be changed.
- Here we need to first work on clicked source.



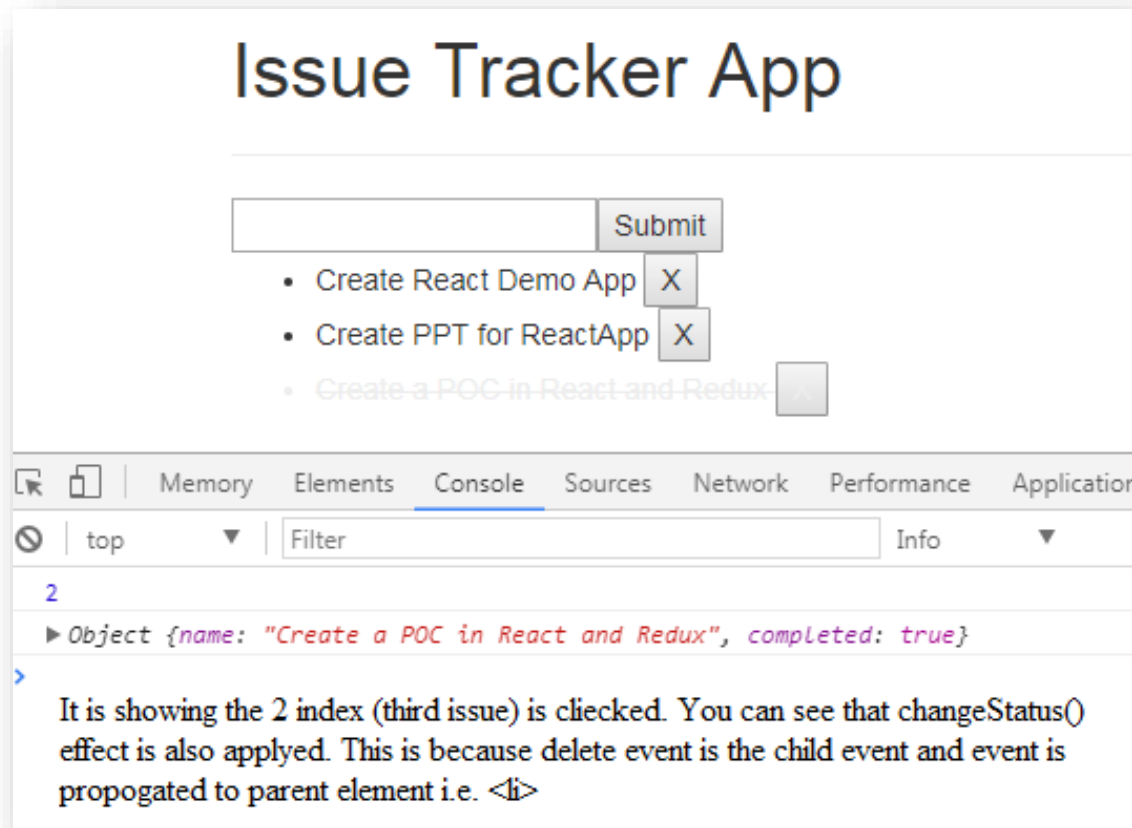
- Which element is clicked for delete request, for that we need to apply the same technique as we have done for which element is clicked for changing the status.

```
Issue-tracker/src/component/IssueNameCoponent
-----
<li onClick={()=>{
  props.clickHandler(props.index)
}} className={props.issue.completed ? 'completed' : ''}>
  {props.issue.name}&nbsp;
  <button
    onClick={()=>{
      props.deleteIssue(props.index)
    }}>X</button>
</li>
```

- Now to varify the clicked item, add below code in deleteIssue() method.

```
Issue-tracker04/src/IssueListComponent
-----
deleteIssue(itemToBeDeleted){
  console.log(itemToBeDeleted+'-----delete button clicked-----');
}
```

- Now verify your changes on browser console.



- Let's write us the logic to delete the issue first and then we will resolve this issue.

- Now verify your changes on browser console.

```
23 deleteIssue(indexTobeDeleted){
24   console.log(indexTobeDeleted);
25   let issues=this.state.issues;
26   issues.splice(indexTobeDeleted,1);
27   this.setState({
28     issues:issues
29   })
30 }
```

take the copy of issues in local issues array

splice method is used to remove the element from array.

this will set the local issues array in the state of IssueListComponent issue array

index location

number of items to be deleted.

- Now go on browser, let console open and click on delete button. you might get below error.

TypeError: Cannot read property 'completed' of undefined

IssueListComponent.changeStatus
C:/Users/sharma.pankaj/Desktop/Internal Training/2017/Pune_July_2017/project/issue-tracker04/src/index.js:61

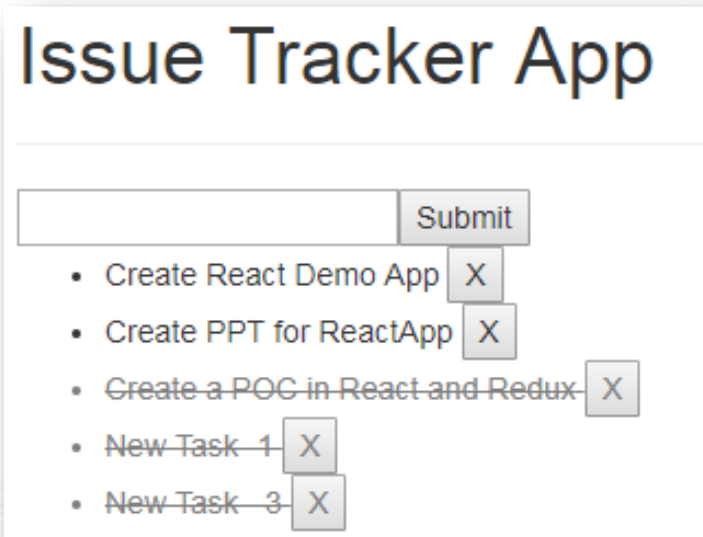
```
58 | changeStatus(index){
59 |   var issues=this.state.issues;
60 |   var issue=issues[index];
61 |   issue.completed=!issue.completed;
```

- This is actually the issue of event propagation.

- Now let's resolve event propagation issue.
- Make below changes in the button section of the IssueNameComponent

```
Issue-tracker/src/component/IssueNameComponent.js
-----
<button
onClick={(event)=>{
event.stopPropagation(),
props.deleteIssue(props.index)
}}>X</button>
```

- Now check the functionality on browser. It should work.





Time Allotted : NA

Req : We want to add the Edit button, while clicking on this button, data will be available for editing in text box and update button will be there to update the data.

Req: We want to add the Edit button, while clicking on this button, data will be available for editing in text box and update button will be there to update the data.

- Uptill now we have stateless component, but for this requirement we need class based component, because for editing we need one state in IssueNameComponent. So first let us convert the component in class based component.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
import React from 'react';
class IssueNameComponent extends React.Component{
  render(){
    return(
      <li onClick={()=>{
        this.props.clickHandler(this.props.index)
      }} className={this.props.issue.completed ? 'completed' : ''}>
        {this.props.issue.name}&nbsp;
        <button
          onClick={(event)=>{
            event.stopPropagation(),
            this.props.deleteIssue(this.props.index)
          }}>X</button>
        </li>
      )
    )
  }
}
export default IssueNameComponent;
```

Check on browser, every thing will work properly.



Now let us add constructor in IssueNameComponent and create isEditing state with value as false.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
constructor(props){
  super(props);
  this.state = {
    isEditing:false
  }
}
```

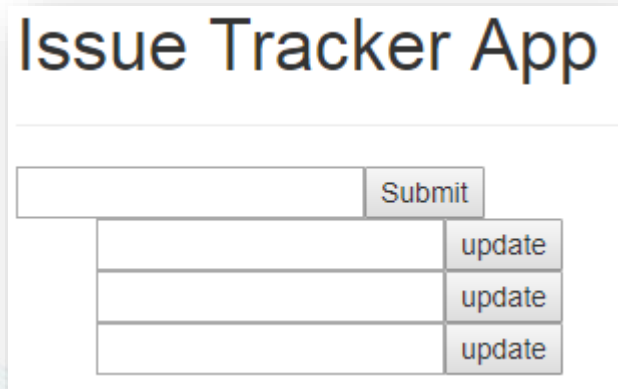
Create <section></section> tag and comment out complete .

```
Issue-tracker/src/component/IssueNameComponent.js
-----
return( <section>
  {/*
  <li onClick={()=>{
    this.props.clickHandler(this.props.index)
  }} className={this.props.issue.completed ? 'completed' : ''}>
    {this.props.issue.name}&nbsp;
    <button
      onClick={(event)=>{
        event.stopPropagation(),
        this.props.deleteIssue(this.props.index)
      }}>X</button>
    </li>
  */} </section> )
```

Create update form as shared below.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
return(
  <section>
    <form>
      <input type="text"/>
      <button>update</button>
    </form>
    { /* . . . */ }
  </section>
)
```

Save changes and check on browser, you must see output as below.



Issue Tracker App

<input type="text"/>	Submit
<input type="text"/>	update
<input type="text"/>	update
<input type="text"/>	update



Now we will show the default Value in update text boxes.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
return(
  <section>
  <form>
  <input type="text" defaultValue={this.props.issue.name}/>
  <button>update</button>
  </form>
  { /* . . . */ }
  </section>
)
```

Save changes and check on browser, you must see output as below.

Issue Tracker App	
<input type="text" value="New Task -2"/>	<input type="button" value="Submit"/>
Create React Demo App	<input type="button" value="update"/>
Create PPT for ReactApp	<input type="button" value="update"/>
Create a POC in React and	<input type="button" value="update"/>
New Task -1	<input type="button" value="update"/>

Now our purpose is to either show the updated form or the list item based on the `isEditing` value.

```
render(){
  //const isEditing = this.state.isEditing;
  const {isEditing} = this.state; /* same as above statement,
                                   can be written in ES6,when local and state variable is same*/
  return(
    <section>
      {
        isEditing ? <form>
          <input type="text" defaultValue={this.props.issue.name}/>
          <button>update</button>
        </form>
        : <li onClick={()=>{
            this.props.clickHandler(this.props.index)
          }} className={this.props.issue.completed ? 'completed' : ''}>
          {this.props.issue.name}&nbsp;
          <button
            onClick={(event)=>{
              event.stopPropagation(),
              this.props.deleteIssue(this.props.index)
            }}>X</button>
        </li>
      }
    </section>
  )
}
```

code will execute when `isEditing` is false.

code will execute when `isEditing` is true

Check on browser, change the `isEditing` value from false to true. We want to create this toggle effect.

Before creating the toggle effect, let us reorganize our code. We have two templates one will be rendered based on true condition and other will be based on false condition. But our code is very difficult to understand now.

We will create separate methods for both the templates, and we will render those methods with our conditions.

- Bind two methods in constructor(). 1. renderForm() and renderIssues()

```
Issue-tracker/src/component/IssueNameComponent.js
-----
constructor(props){
  . . . .
  this.renderForm=this.renderForm.bind(this);
  this.renderIssues=this.renderIssues.bind(this);
}
```

- Create renderForm() and renderIssues() methods and cut and paste the templates used with ? And : operators.



```
Issue-tracker/src/component/IssueNameComponent.js
-----
renderForm(){
  return(
    <form>
    <input type="text" defaultValue={this.props.issue.name}/>
    <button type="submit">update</button>
    </form>
  )
}
```

```
Issue-tracker/src/component/IssueNameComponent.js
-----
renderIssues(){
  return(
    <li onClick={()=>{
      this.props.clickHandler(this.props.index)
    }} className={this.props.issue.completed ? 'completed' : ''}>
    {this.props.issue.name}&nbsp;
    <button
      onClick={(event)=>{
        event.stopPropagation()
        this.props.deleteIssue(this.props.index)
      }}>X</button>
    </li>
  )
}
```

Now make appropriate changes in your IssueNameComponent.js to display renderForm() or renderIssue() method.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
class IssueNameComponent extends React.Component{
. . . . .
render(){
  const {isEditing}=this.state;
  return(
    <section>
    {
      isEditing ? this.renderForm() : this.renderIssues()
    }
    </section>
  )
}
export default IssueNameComponent;
```



Now we will create one button that says Edit, so that, we can change the list view to update view.

- Create toggleState() method and bind the “this” ref with toggleState() method.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
constructor(props){
  . . .
  this.toggleState = this.toggleState.bind(this);
  . . .
}
```

```
Issue-tracker/src/component/IssueNameComponent.js
-----
toggleState(){
  const {isEditing} = this.state;
  this.setState({
    isEditing:!isEditing
  })
}
```

We need one more button labeled as Edit same as delete button. Make below changes in Edit button of inside the `renderIssue()` method as shown in the screen shot below.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
renderIssues(){
  return(
    <li onClick={()=>{
      this.props.clickHandler(this.props.index)
    }} className={this.props.issue.completed ? 'completed' : ''}>
      {this.props.issue.name}&nbsp;
      <button
        onClick={(event)=>{
          event.stopPropagation()
          this.props.deleteIssue(this.props.index)
        }}>X</button>
      <button
        onClick={(event)=>{
          event.stopPropagation()
          this.toggleState()
        }}>Edit Issue</button>
    </li>
  )
}
```

Issue Tracker App

<input type="text"/>	Submit
• Create React Demo App	X Edit Issue
Create PPT for ReactApp	update
Create a POC in React and	update



Uptil now we have created update form with prefilled data. Now we want to update the data in update form and toggle the form with changed list view.

- Add onSubmit event on form when user press Edit button or Enter key form should be submitted.
- onSubmit event will trigger on updateIssue() method.
- Create one updateIssue() method and bind the “this” ref with it.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
class IssueNameComponent extends React.Component{
  . . . .
  this.updateIssue = this.updateIssue.bind(this);
  . . . .
  updateIssue(){
  }
  . . . .
  renderForm(){
  return(
    <form onSubmit={this.updateIssue}>
    <input type="text" defaultValue={this.props.issue.name}/>
    <button type="submit">update</button>
    </form>
  )
  }
}
```

Now we have onSubmit event on <form> which will trigger the updateIssue() method. Once the updateIssue() method is triggered, we are now interested to take the value of <input... /> control. This can be achieved by ref attribute in React.

To know more : <https://facebook.github.io/react/docs/refs-and-the-dom.html>

```
class IssueNameComponent extends React.Component{
  -----
  43  renderForm(){
  44    return(
  45      <form onSubmit={this.updateIssue}>
  46        <input type="text"
  47          defaultValue={this.props.issue.name}
  48          ref={(value) => { this.input = value; }}
  49        />
  50        <button type="submit">update</button>
  51      </form>
  52    )
  53  }
  -----
}
```

input DOM element will be received as a value

Value will be available to IssueNameComponent using this ref.

Note : JS concept of adding prop to object at run time.



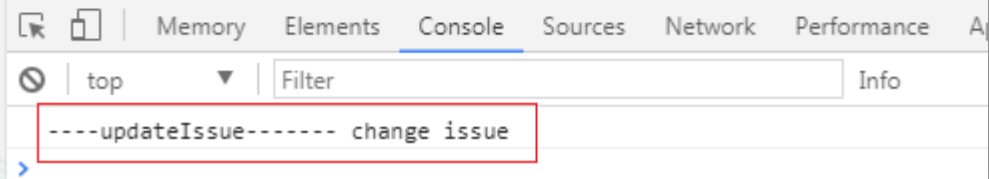
Now we want to check whether `updateIssue()` is triggering. So just `console.log("----updateIssue-----",this.input.value)`, and verify from browser console.

```
updateIssue(event){  
  event.preventDefault(); _____ this will stop the page refresh.  
  console.log('----updateIssue-----',this.input.value)  
}
```

Issue Tracker App

change issue

- Create PPT for ReactApp
- Create a POC in React and Redux



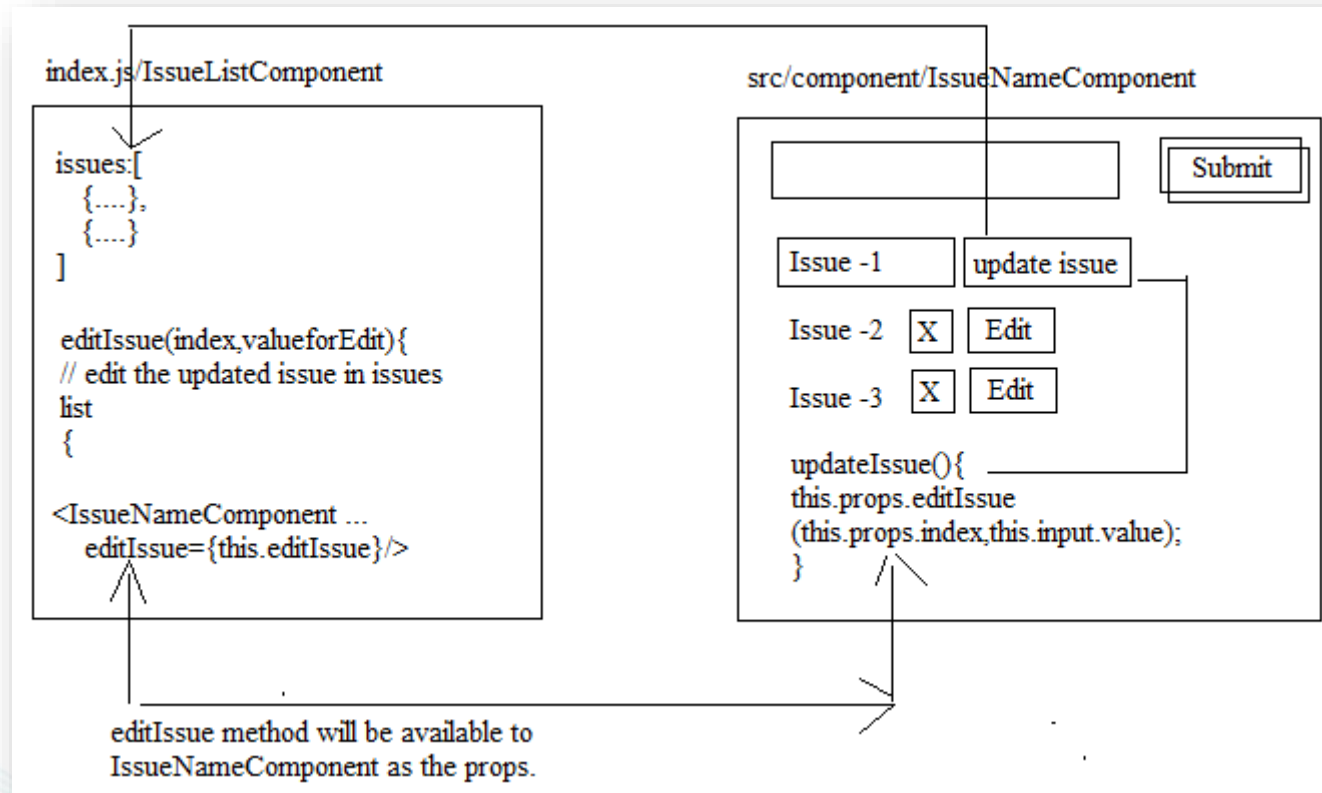


Edit issue in the list of issues

113

Now updateIssue() method is triggered when we click on update issue button. now the requirement is that we want to pass index and value to be updated in IssueListComponent, so that changes can be reflected to issues array in IssueListComponent.

Refer below diagram to understand the flow.





Now let us first add the `editIssue()` method, bind it with this ref and pass it to `IssueNameComponent` as props.

```
Issue-tracker/src/IssueListComponent
-----
class IssueListComponent extends React.Component{
  constructor(){
    . . . .
    this.editIssue=this.editIssue.bind(this);
    . . . .
  }
  editIssue(){

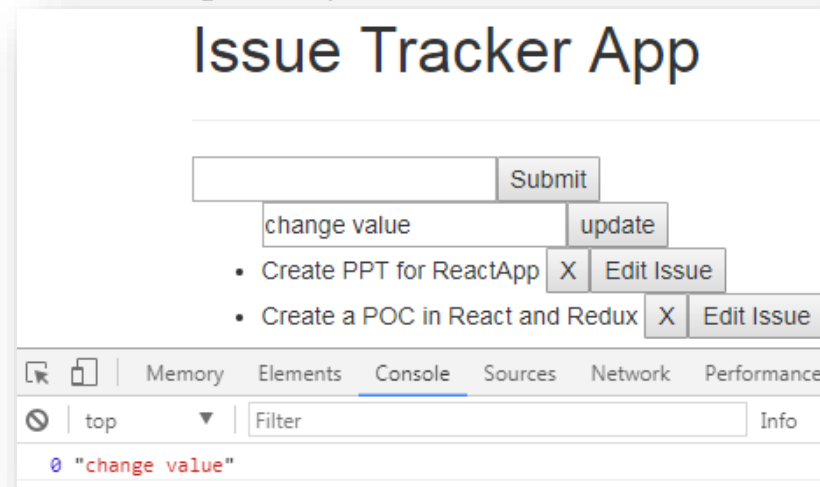
  }
  . . . .
  return <IssueNameComponent
    key={issue.name}
    issue={issue}
    clickHandler={this.changeStatus}
    deleteIssue={this.deleteIssue}
    editIssue={this.editIssue}
    index={index}/>
```

Now we can use `editIssue()` method in `updateIssue()` and need to pass the index and value to `editIssue()` method.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
updateIssue(event){
  event.preventDefault();
  //console.log('----updateIssue-----
  ',this.input.value)
  this.props.editIssue(this.props.index,
  this.input.value);
}
```

Now index and value will be available in `editIssue()` of `IssueListComponent.js`.

```
Issue-tracker/IssueListComponent
-----
editIssue(index,newValue){
  console.log(index,newValue);
}
```





Now we want to update the existing name with the newName. We need to update the editIssue() method of IssueListComponent.

```
24 editIssue(index, newValue){
25   console.log(index, newValue);
26   let issues=this.state.issues; ----- copy issues array into local issues array
27   let currentIssue=issues[index]; ----- take the current value based on index.
28   currentIssue['name']=newValue; ----- replace the name with the new value.
29   this.setState({
30     issues ----- update the state.
31   }) ----- this can be written as issues:issues
                but in ES6 if we have key and value pair same, we can
                write this JSX.
```

Make changes to toggle the update view to list view in IssueNameComponent's updateIssue() method.

```
15 updateIssue(event){
16   event.preventDefault();
17   //console.log('----updateIssue-----',this.input.value)
18   this.props.editIssue(this.props.index, this.input.value);
19   this.toggleState(); -----
20 } ----- to toggle the state from update view to list view.
```

Now check your application, all CRUD operations will work.

PropTypes in ReactJS provides the type safety on props that are used in components.

Ref : <https://facebook.github.io/react/docs/typechecking-with-proptypes.html>

Code

```
import PropTypes from 'prop-types';

class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}
```

```
Greeting.propTypes = {
  name: PropTypes.string
};
```

PropTypes exports a range of validators that can be used to make sure the data you receive is valid. In this example, we're using `PropTypes.string`. When an invalid value is provided for a prop, a warning will be shown in the JavaScript console. For performance reasons, `propTypes` is only checked in development mode.

We can add type-checking in our issue-tracker application.

- Open the index.js file
- We have `<IssueForm .../>` and `<IssueNameComponent .../>` and here we are passing some props.

```
<IssueForm
  currentIssue={this.state.currentIssue}
  updateIssue={this.updateIssue}
  addIssue={this.addIssue}/>
<ul>
  {
    this.state.issues.map((issue,index)=>{
      return <IssueNameComponent
        key={issue.name}
        issue={issue}
        clickHandler={this.changeStatus}
        deleteIssue={this.deleteIssue}
        editIssue={this.editIssue}
        index={index}/>
    })
  }
</ul>
</section>
```

Diagram illustrating the props passed to the components:

- `currentIssue`, `updateIssue`, and `addIssue` are passed to `<IssueForm>` and are labeled as **Function**.
- `key`, `issue`, `clickHandler`, `deleteIssue`, and `editIssue` are passed to `<IssueNameComponent>`.
 - `key` is labeled as **React Specific**.
 - `issue` is labeled as **Object**.
 - `clickHandler`, `deleteIssue`, and `editIssue` are labeled as **Function**.
 - `index` is labeled as **Number**.

We can add type-checking in our issue-tracker application.

- Copy all the props except key from `<IssueNameComponent ..>` tag
- Open the `IssueNameComponent.js` file and before the ***export default IssueNameComponent*** statement add below code.

```
Issue-tracker/src/component/IssueNameComponent.js
-----
. . . .
IssueNameComponent.propTypes = {
  issue:React.PropTypes.object,
  clickHandler:React.PropTypes.func,
  deleteIssue:React.PropTypes.func,
  editIssue:React.PropTypes.func,
  index:React.PropTypes.number
}
export default IssueNameComponent;
```

Now if any prop other than the mentioned types comes in `IssueNameComponent`, then React will give the warning on console.
Let's experiment

- Go back to `index.js` and change the type of any prop in `<IssueNameComponent..>` other than the mentioned type.
For example index is number : write as `index={" "+index}`, and check your application console.

In console if you see below warning.

```
⚠ Warning: Accessing PropTypes via the main React package is deprecated, and will be removed in React v16.0. Use the latest available v15.* prop-types package from npm instead. For info on usage, compatibility, migration and more, see https://fb.me/prop-types-docs
```

Now to resolve this issue. Follow below steps.

- Install the prop-types node package in your application
 - `npm install --save prop-types`
- Import the prop-types in your desired component.
 - `import PropTypes from 'prop-types'; // ES6`
 - `var PropTypes = require('prop-types'); // ES5 with npm`

```
Issue-tracker/src/component/IssueNameComponent.js
```

```
-----  
import PropTypes from 'prop-types';  
. . .
```

```
✖ Warning: Failed prop type: Invalid prop `index` of type `string` supplied to `IssueNameComponent`,  
expected `number`.  
    in IssueNameComponent (at index.js:92)  
    in IssueListComponent (at index.js:110)
```

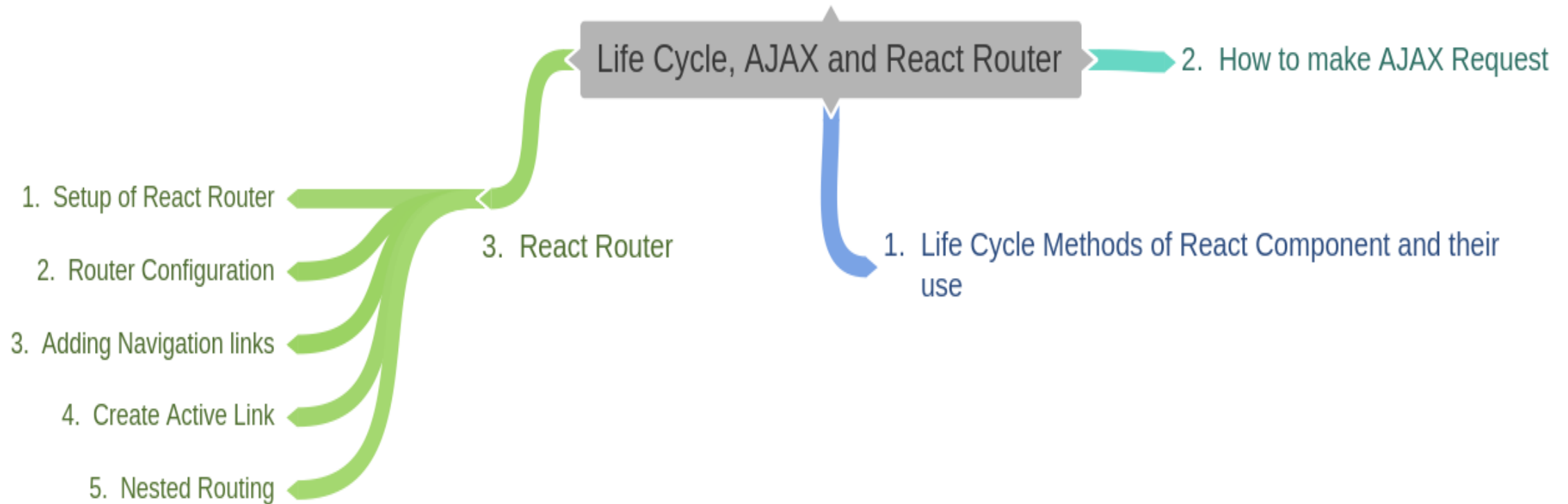
prop-types ref : <https://github.com/facebook/prop-types#prop-types>





Day -3

React Life Cycle Methods, AJAX and React Router



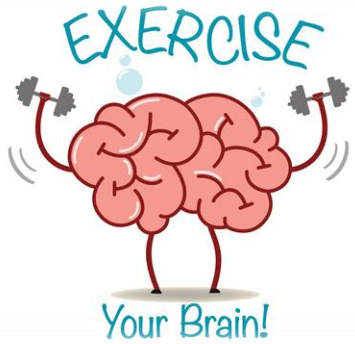
Each component has several life cycle methods.

- These life cycle methods are important to run a particular code at particular time in the process.
- Methods prefixed with “will” are called right before something happens.
- Methods prefixed with “did” are called right after something happens.

All Lifecycle methods can be split in four phases : **initialization, mounting, updating** and **unmounting**.

Ref :

- <https://medium.com/react-ecosystem/react-components-lifecycle-ce09239010df>
- <https://facebook.github.io/react/docs/react-component.html>



Time Allotted : NA

Create a day3 project in DAY-3 folder

Up till now we have already used two lifecycle methods in our applications.

- constructor()
- render()

These two methods comes under the category of Mounting. We have two more methods in this category

- componentWillMount()
- componentDidMount()

To understand the behaviour let us write the application, to check which method will trigger at what time.

Create a component and name it as LifeCycleDemoComponent.

```
day3/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
class LifeCycleDemoComponent extends React.Component{
  render(){
    return (
      <div>
        <h1>Demo of LifeCycle Methods</h1>
      </div>
    )
  }
}
ReactDOM.render(<LifeCycleDemoComponent />, document.getElementById('root'));
```

Now create constructor(), componentWillMount(), componentDidMount() methods, and for testing write console.log() in all the four methods, and check on browser console the sequence of execution.

```
Life-cycle-demo/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom';

class LifecycleDemoComponent extends React.Component{

  constructor(){
    super();
    this.state={
      methods:['constructor()', 'componentWillMount()', 'render()', 'componentDidMount()']
    }
  }
  console.log('-----constructor called-----');
}

componentWillMount(){
  console.log('-----componentWillMount called-----');
}

componentDidMount(){
  console.log('-----componentDidMount called-----');
}
```

```
day3/src/LifeCycleDmoComponent
-----
render(){
  console.log('-----render called-----');
  return (
    <div>
      <h1>Demo of LifeCycle Methods</h1>
      <hr/>
      <h2>Open the browser console to check the output</h2>
      <h3>
        Mounting
      </h3>
      <p>These methods are called when an instance of a component is being created and instered into the DOM.</p>
      <ul>
        {this.state.methods.map((method)=>{return <MethodNameComponent key ={method} method={method}/>})}
      </ul>
    </div>
  )
}
```



```
day3/src/index.js
-----
class MethodNameComponent extends React.Component{
  render(){
    return(
      <li>{this.props.method}</li>
    )
  }
}
```

ReactDOM.render(<LifeCycleDemoComponent />, document.getElementById('root'));

Now here we will learn how to make AJAX Request. We will be using the JQuery support here to make AJAX Request. There are other better alternatives as well : Axios and Fetch API

To add the JQuery in your project run below command on command prompt in day3 project terminal window.

`npm i jquery --s`

Our plan is to take JSON data from some other server and render the information from available JSON.

<https://jsonplaceholder.typicode.com/>

We will be using the todos json from here.

<https://jsonplaceholder.typicode.com/todos>

Create basic application structure

```
day3/src/App.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component{
  render(){
    return(
      <div>
        <h1>AJAX-Demo</h1>
        <hr/>
      </div>
    )
  }
}
ReactDOM.render(<App />, document.getElementById('root'));
```

Now we will be getting some todos data from live server, to hold this data in our application we will require one todos array, it should be instantiated in state, that we mention in constructor. Follow below code.

```
day3/src/App.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component{
  constructor(props){
    console.log('-----constructor-----');
    super(props);
    this.state={
      todos:[]
    }
    . . . .
  }
  ReactDOM.render(<App />, document.getElementById('root'));
```

Now improvise the render() method to show title from todos in list.

```
day3/src/App.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
. . . .
render(){
  console.log('-----render-----');
  const {todos} =this.state;
  return(
    <div>
    <h1>AJAX-Demo</h1>
    <hr/>
    <ul>
    {
      todos.map((todo)=>{
        return <li>{todo.title}</li>
      })
    }
    </ul>
    </div>
  )
}
ReactDOM.render(<App />, document.getElementById('root'));
```

What is the purpose of using => function here?

Now import \$ from jquery on App.js

```
day3/src/App.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
import $ from 'jquery'
. . . .
ReactDOM.render(<App />, document.getElementById('root'));
```

Now add `componentDidMount()` method, and get data from server and assign it in `todos` array.

```
day3/src/App.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
import $ from 'jquery'
. . . . // constructor
componentDidMount(){
  console.log('-----componentDidMount-----');
  $.ajax({
    url: 'https://jsonplaceholder.typicode.com/todos',
    success:(data)=>{
      this.setState({
        todos:data
      })
    }
  })
}
. . . . // render
ReactDOM.render(<App />, document.getElementById('root'));
```

Refs : How to use axios in React

<https://alligator.io/react/axios-react/>

Watch This

- Open the day3 project in visual studio code.
- Now run below command to install *react-router-dom* in your application
- *npm install react-router-dom -S*
- To verify this we can open the package.json and check whether it is installed and added in package.json

Create a stateless component and name it as ***RouterComponent***. Import {BrowserRouter as Router, Route} from 'react-router-dom'

```
day3/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom';
import {BrowserRouter as Router, Route} from 'react-router-dom';

const RouterComponent = () =>(
  {/* will be adding the code for Router configuration*/}
)

ReactDOM.render(<RouterComponent/>, document.getElementById("root"));
```

Add Router using `<Router></Router>` in RouterComponent

```
React-router-demo/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom'; import {
BrowserRouter as Router,Route} from 'react-router-dom';

const RouterComponent = () =>(
  <Router>
  <Route path="/" component={HomeComponent}/>
  </Router>
)

ReactDOM.render(<RouterComponent/>,document.getElementById("root"));
```

Path : url path

Component : based on path which Component will render

Create HomeComponent and return the *HOME* String to represent home component rendering.

```
day3/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom'; import {
BrowserRouter as Router, Route } from 'react-router-dom';
const HomeComponent = () =>(
<h1>Home Page</h1>
)
const RouterComponent = () =>(
<Router>
<Route path="/" component={HomeComponent}/>
</Router>
)

ReactDOM.render(<RouterComponent/>, document.getElementById("root"));
```

Now save the changes and check on Browser, your Home component must be rendering.

Now create another Router for about component. And create AboutComponent that should return About String.

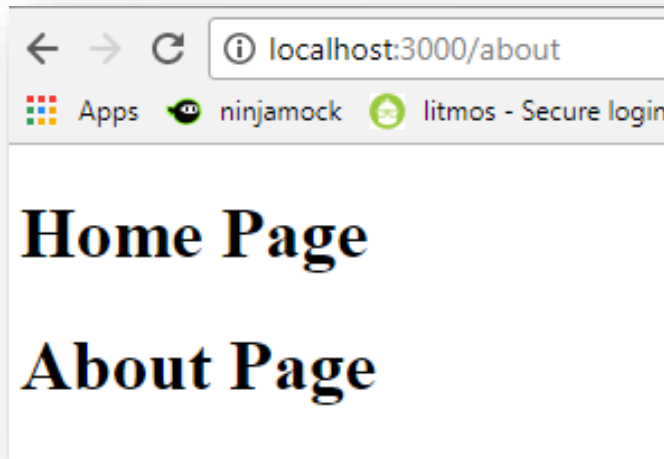
```
day3/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom'; import {
BrowserRouter as Router, Route } from 'react-router-dom';
const HomeComponent = () =>(
<h1>Home Page</h1>
)
const AboutComponent = () =>(
<h1>About Page</h1>
)
const RouterComponent = () =>(
<Router>
<Route path="/" component={HomeComponent}/>
<Route path="/about" component={AboutComponent}/>
</Router>
)
ReactDOM.render(<RouterComponent/>, document.getElementById("root"));
```

Now check your application, it will throw some error.

It is because your <Router> returning two values, you can return only single value, so place both the Route tags in <div></div>

Now if you check it on browser, Home Page will be displayed.

Add following in URL : *localhost:3000/about* and hit Enter. You will find below output.



It is rendering both the components, because it is following the path of Home i.e. / and also the path of about i.e. /about

To resolve this you need to tell that Home will be rendering exactly for /
Add below code and recheck.

```
day3/src/index.js
-----
. . . .
<Route exact path="/" component={HomeComponent}/>
```

Again and again hitting the browser to activate a particular view is not good practice, it is better to have Navigation Menu, where user can click and activate desired component.

Let us create Links component that will expose the Links. Add Link attribute in import

```
day3/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom'; import {
BrowserRouter as Router,Route,Link} from 'react-router-dom';
. . HomeComponent . .
. . AboutComponent. .
const Links = () =>(
<ul>
<li><Link to="/">Home</Link></li>
<li><Link to="/about">About</Link></li>
</ul>
)
const RouterComponent = () =>(
<Router><div>
<Links/>
<Route exact path="/" component={HomeComponent}/>
<Route path="/about" component={AboutComponent}/>
</Router></div>
)
ReactDOM.render(<RouterComponent/>,document.getElementById("root"));
```

Now if you check,
your links should
function.

Link tag is with limited features, if you want to show the active link, you will have to use another option. i.e. NavLink. Include NavLink in import statement and make appropriate changes.

```
day3/src/index.js
-----
import React from 'react';
import ReactDOM from 'react-dom'; import {
BrowserRouter as Router, Route, Link, NavLink} from 'react-router-dom';
. . HomeComponent . .
. . AboutComponent. .
const Links = () =>(
<ul>
<li><NavLink activeClassName="selected" to="/">Home</NavLink></li>
<li><NavLink activeClassName="selected" to="/about">About</NavLink></li>
</ul>
)
. . RouterComponent . .
ReactDOM.render(<RouterComponent/>,document.getElementById("root"));
```

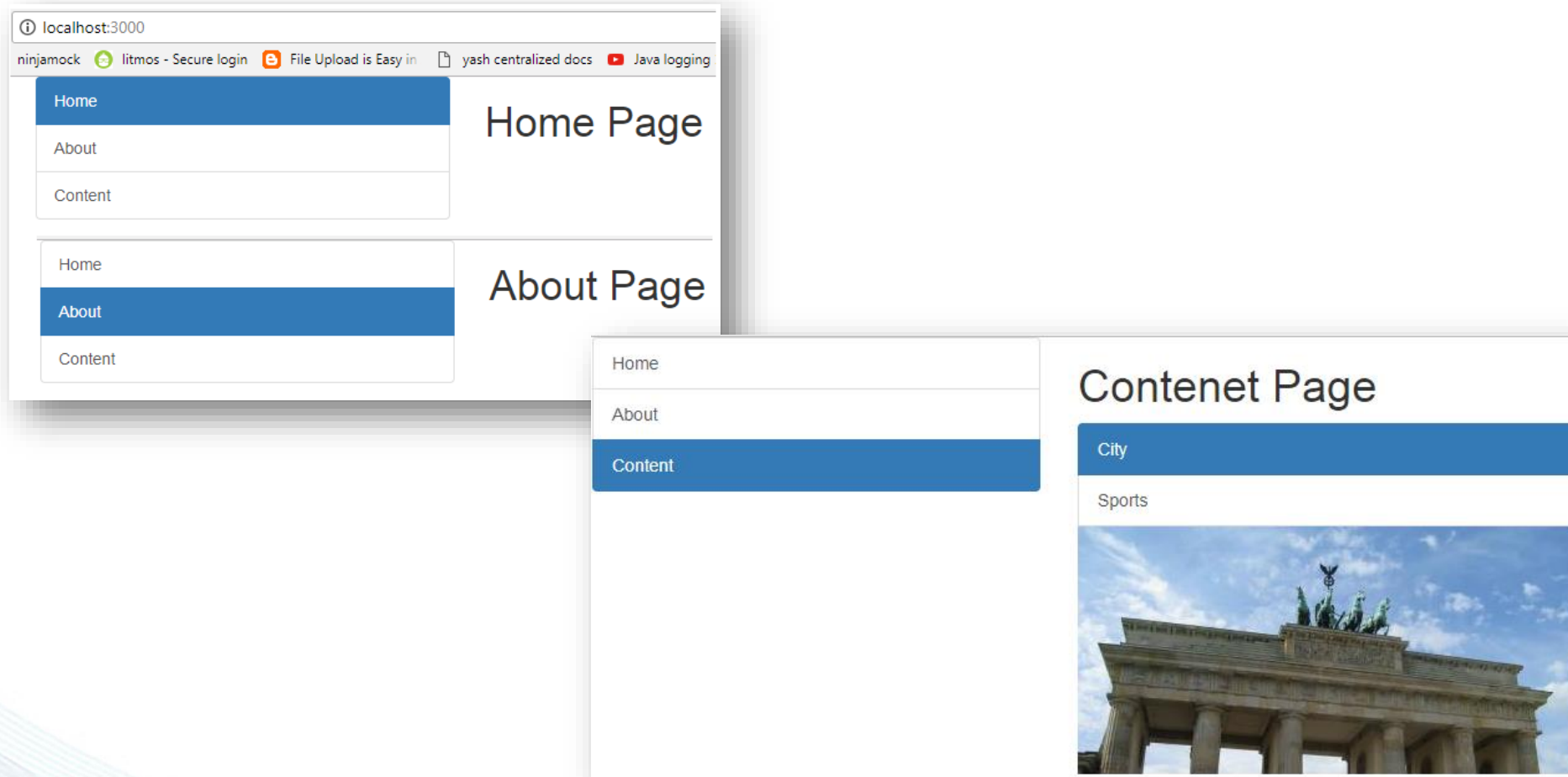
```
.selected{
color:red;
}
```

Add it in index.css

Your link will be activated but the problem will be that when you will click on About. It will not bring the Home link in default state. This is the same problem we faced with path in Route. We need to add exact here as well. Make below changes and check.

```
day3/src/index.js
-----
<li><NavLink exact activeClassName="selected" to="/">Home</NavLink></li>
```


Here I will show you something about Routing inside another Routing. Below screen shots will clear it.





To add some styling I have used here the bootstrap.

Go to below location and copy the link of bootstrap from CDN and paste that in the index.html's head section.

<http://getbootstrap.com/getting-started/#download>

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <!--
      manifest.json provides metadata used when your web app is added to the
      homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div class="container">
      <div id="root"></div>
    </div>
```

Now make appropriate changes in RouterComponent in index.js.

```
const RouterComponent = () =>(  
  <Router>  
    <div className="row"> // will be implementing from bootstrap  
      <section className="col-sm-4">  
        <Links/>  
      </section>  
      <section className="col-sm-8">  
        <Route exact path="/" component={HomeComponent}/>  
        <Route path="/about" component={AboutComponent}/>  
        <Route path="/content" component={ContentComponent}/> // one more Route configuration  
      </section>  
    </div>  
  </Router>  
)
```

Update the Links component to add bootstrap classes.

```
const Links = () =>(  
  <div className="list-group">  
    <NavLink className="list-group-item" exact activeClassName="active" to="/">Home</NavLink>  
    <NavLink className="list-group-item" activeClassName="active" to="/about">About</NavLink>  
    <NavLink className="list-group-item" activeClassName="active" to="/content">Content</NavLink>  
  </div>  
)
```



Create ContentComponent

```
const ContentComponent = () =>(  
  <div>  
    <h1>Content Page</h1>  
  </div>  
)
```

Now we want to add the functionality for the Content section.

```
const ContentComponent = () =>(  
  <div>  
    <h1>Content Page</h1>  
    <div>  
      <NavLink className="list-group-item" exact activeClassName="active" to="/content/city">City</NavLink>  
      <NavLink className="list-group-item" exact activeClassName="active" to="/content/sports">Sports</NavLink>  
    </div>  
  </div>  
)
```

Check your application

Now we want to add the Route for city and Sports, so while clicking the city or sports it should open the appropriate section.

```
const ContentComponent = () =>(  
  <div>  
    <h1>Content Page</h1>  
    <div>  
      <NavLink className="list-group-item" exact activeClassName="active" to="/content/city">City</NavLink>  
      <NavLink className="list-group-item" exact activeClassName="active" to="/content/sports">Sports</NavLink>  
      <Route path="/content/:contentName" component={ContentDetails}/>  
    </div>  
  </div>  
)
```

Create ContentDetails component.

```
const ContentDetails = (props) =>(  
  <div>{props.match.params.contentName} </div>  
)
```

Now we would like to show the some city and some sports image.

```
const ContentDetails = (props) =>(  
  <div>{props.match.params.contentName  
    ?  
    <div>  
      <img src={'http://lorempixel.com/400/200/' + props.match.params.contentName + '/1/'} /></div>  
      : null }  
    </div>  
  )  
)
```

Now check your application

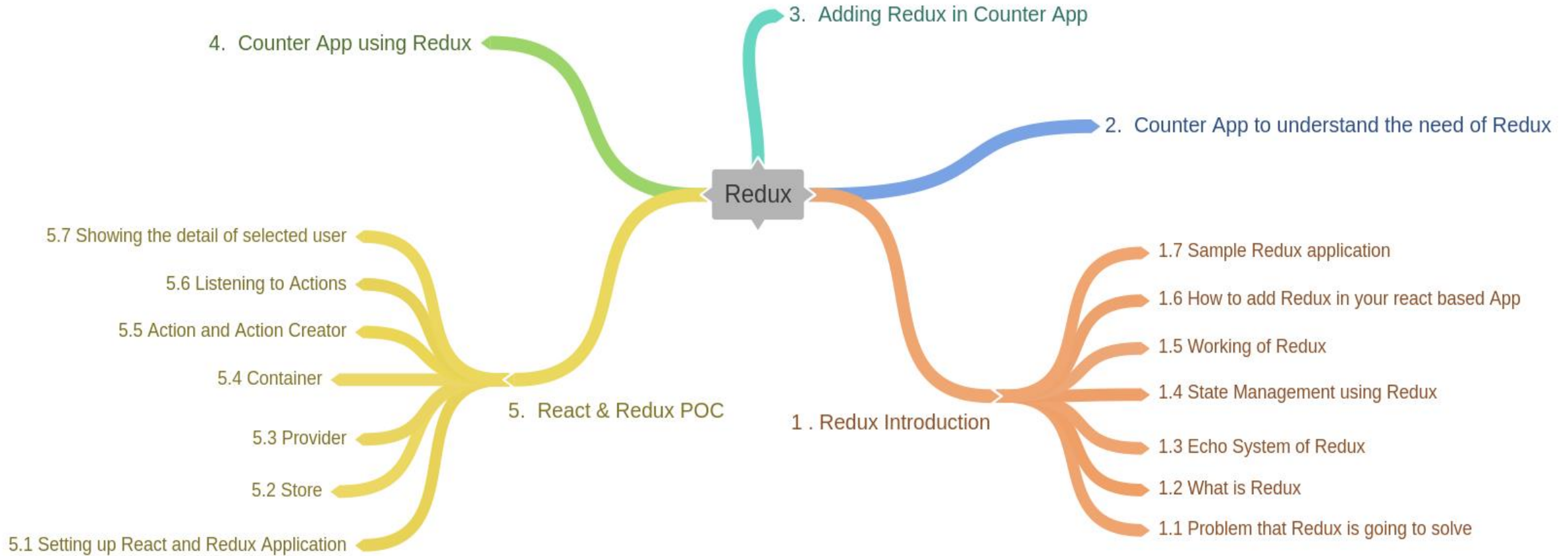
React Router Reference : <https://reacttraining.com/react-router/>

google for npmjs.com > search react-router4



Day -4

Redux

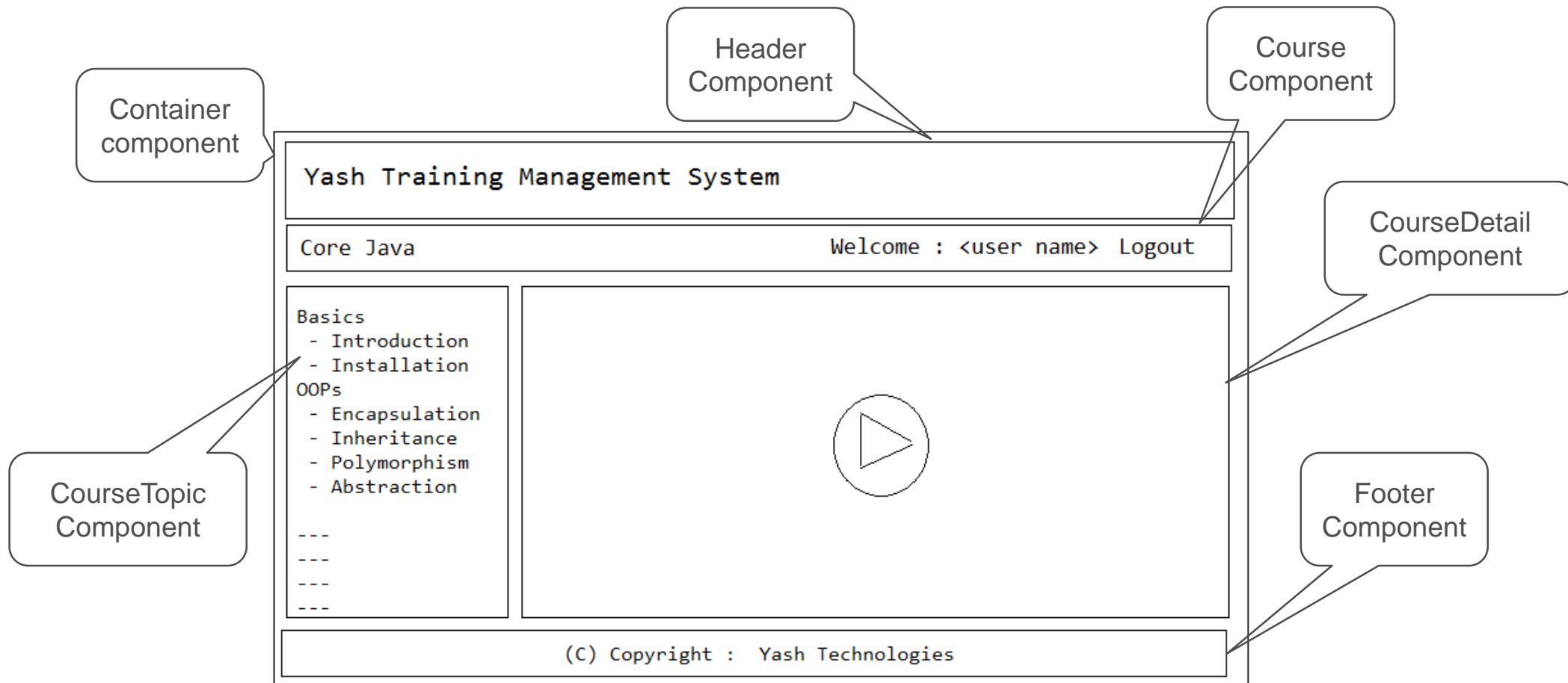




Redux

Introduction of Redux

Consider an application as shown below



Points to notice

- We will have different components for representing each UI part.
- Each component will have some data i.e.state of the component
- State from one component will be passed to other component as props
- Some components may require the Action to be performed.
- Based on the action performed, state of a particular component will be updated

Some of the pain points when working with plain React applications

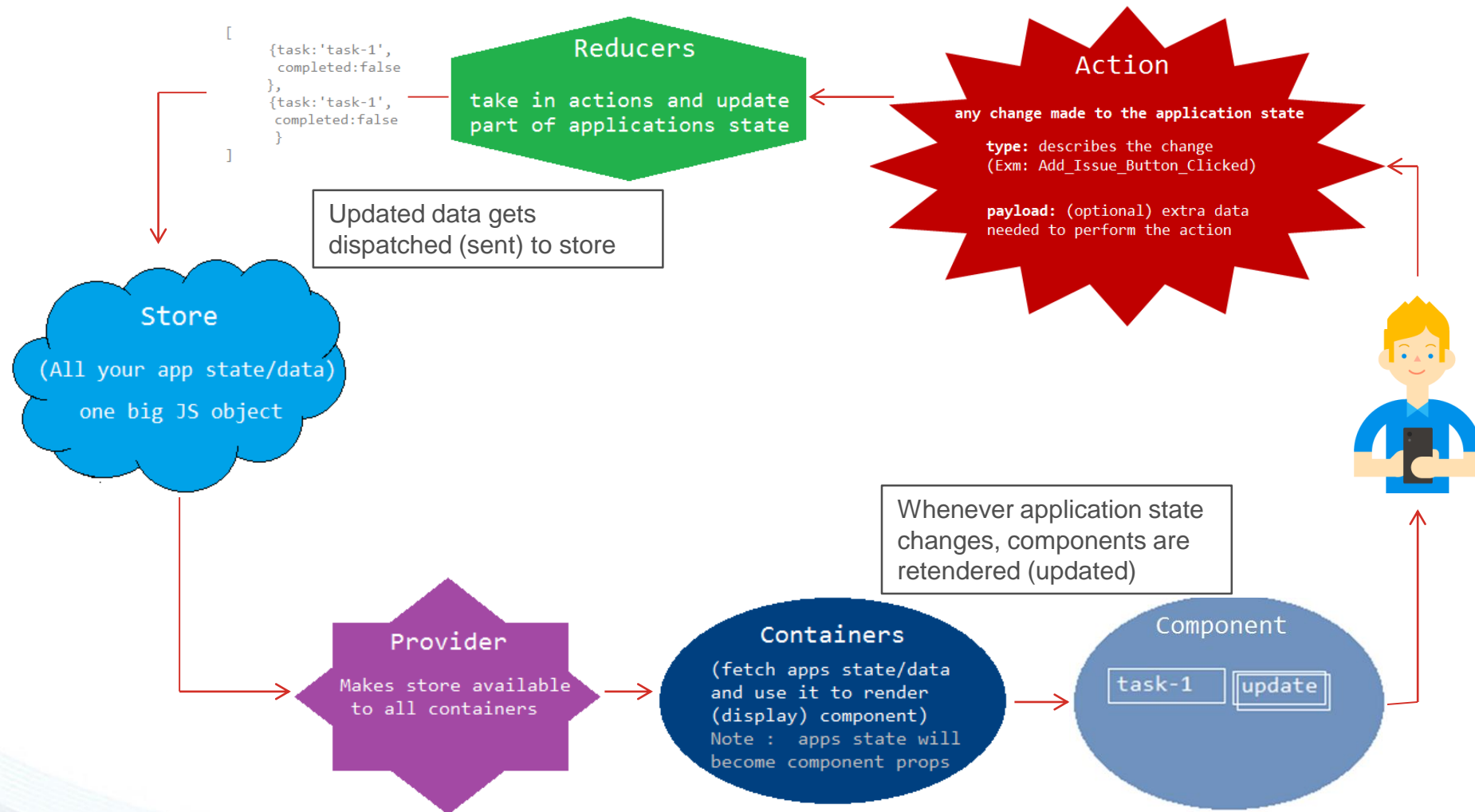
- Managing the state of each component
- Managing the actions
- Action notifications
- Providing the state as props to other component
- Managing the state changes from child to parent component

Redux is the solution for all the above pain points.

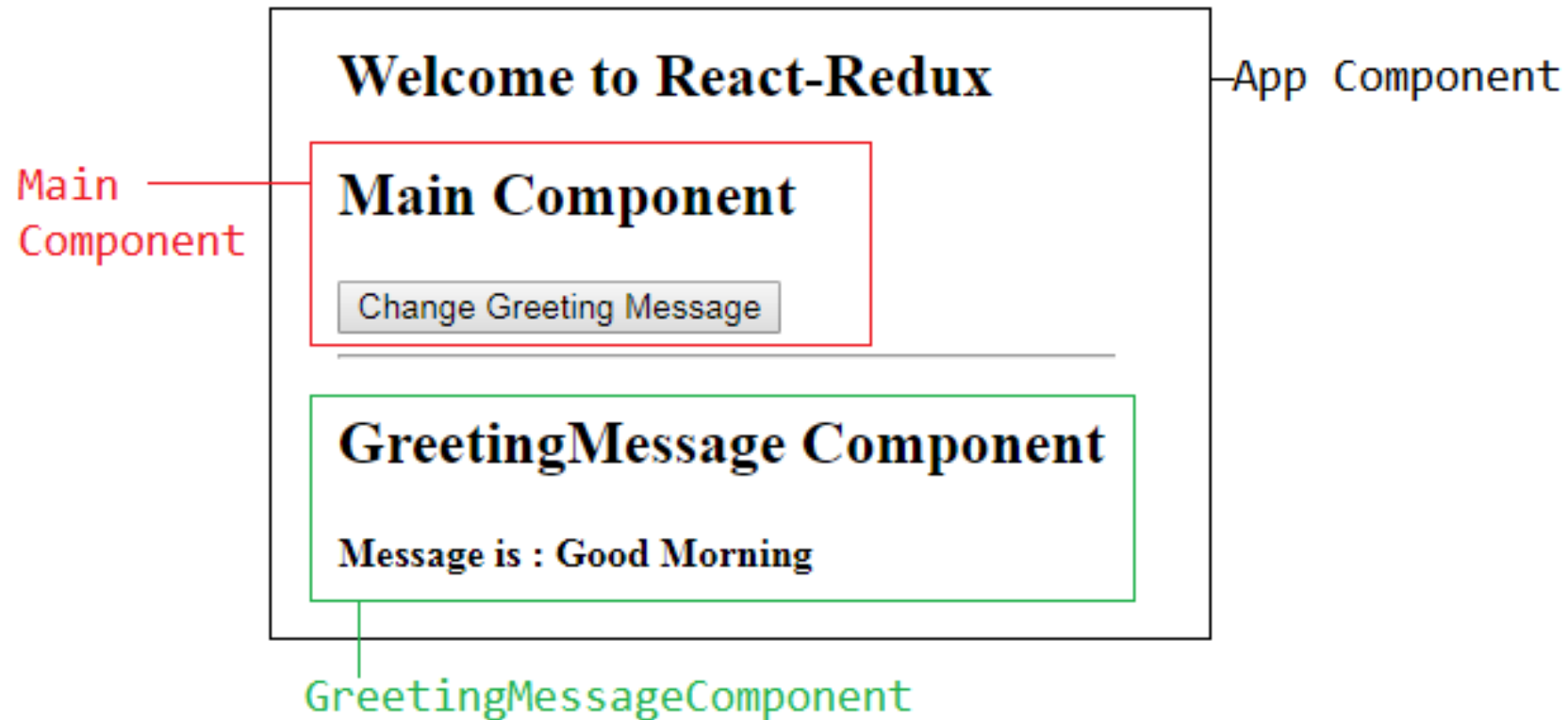
- Redux is a state container for javascript app, all the states for each components will be available in one big javascript object.
- Redux is a framework for managing the state of your application, triggering the actions and providing the state to specific component.

Echo System of Redux

- Store
- Provider
- Container
- Component
- Action
- Reducer

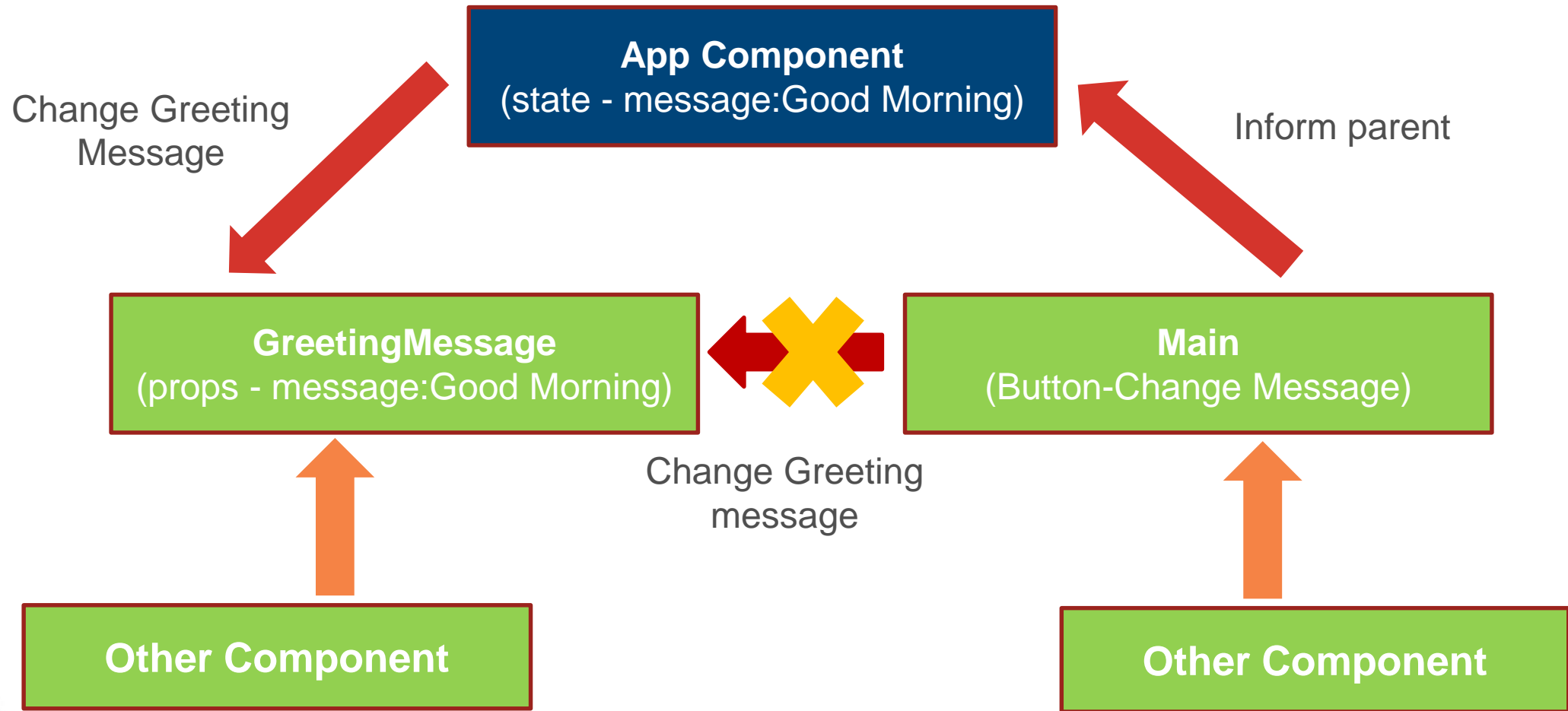


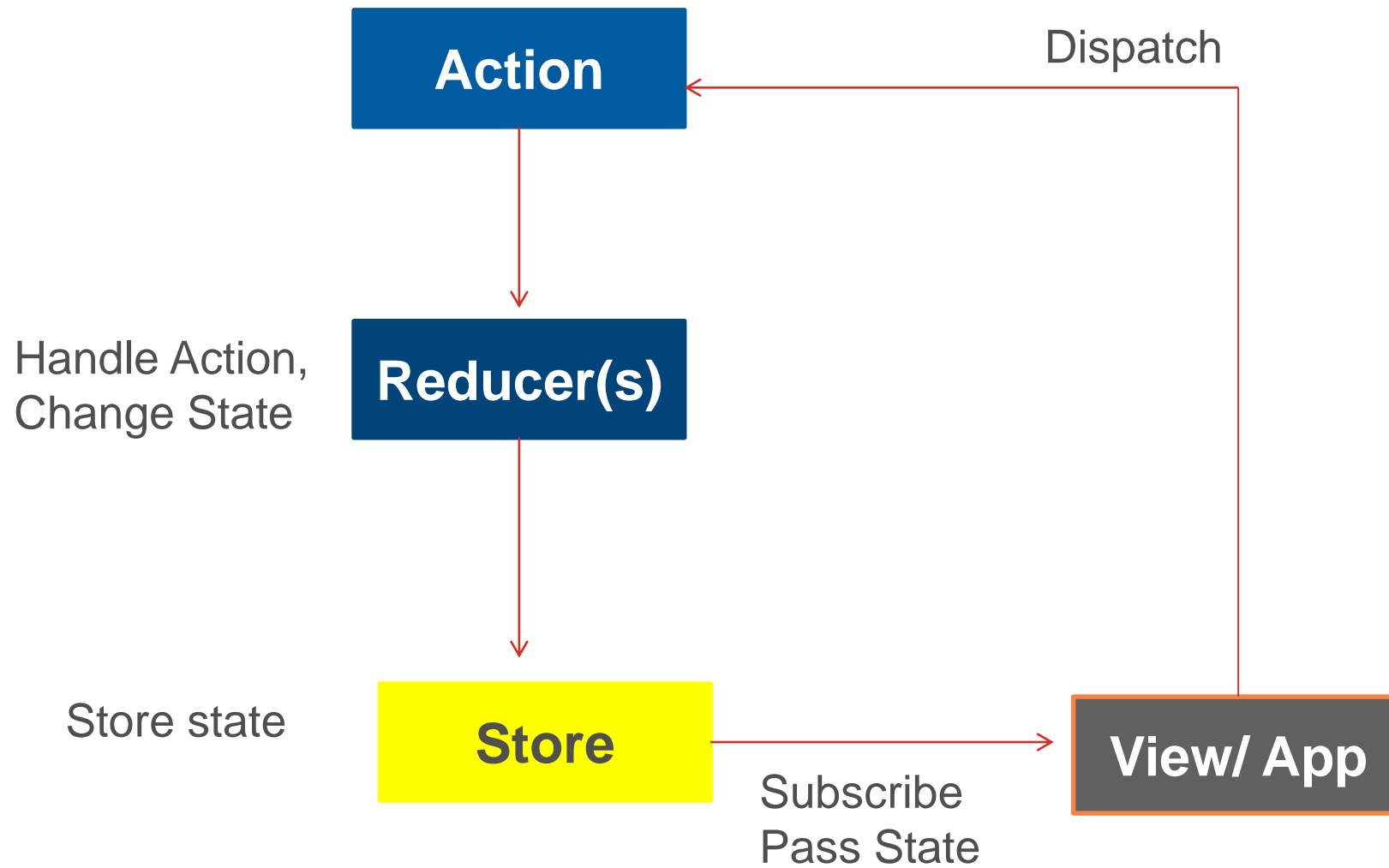
Create the below application in React.

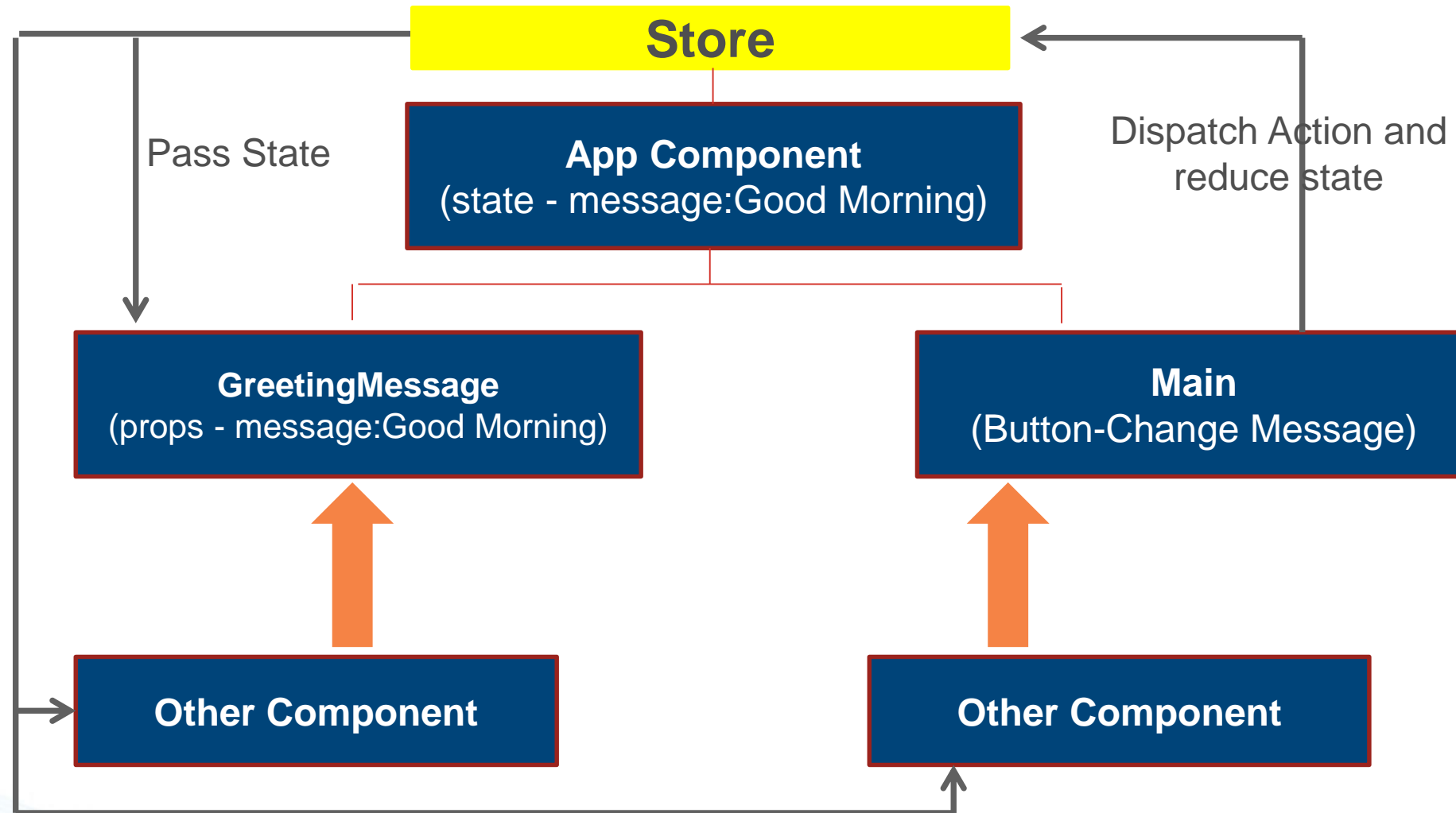


Steps:

- 1: Create new application with **create-react-app react-redux-demo** command.
- 2: Run npm start command and let open the browser.
- 2: Remove **App.js** , **App.css**, **App.test.js** files from **src** folder
- 3: Remove whole code from **index.js** file
5. Create the application from scratch









Redux

Counter App to understand the need of Redux



Ref Link : <https://github.com/code-gram/React-Pune-2018/tree/master/counter-app%20without%20redux/src>

Redux Keeps the **State** of your app in single **Store**. Then you can extract part of that state and plug. This lets you keep data in one global place (**Store**) and feed it directly to any component in the app.

// TODO: Refer Notes to do it by your self



Redux

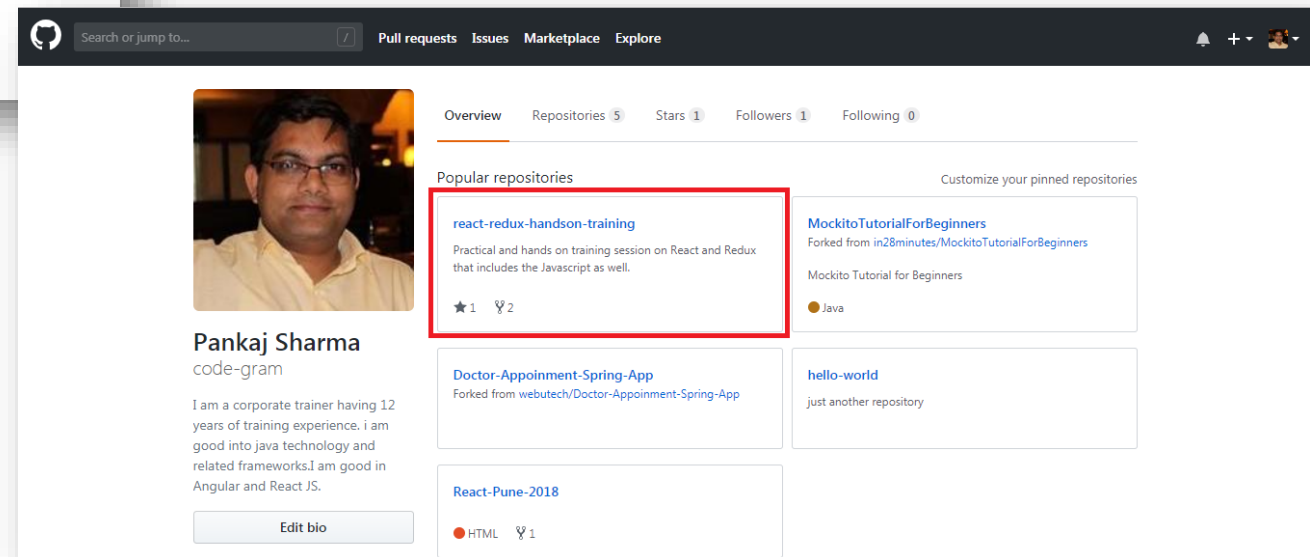
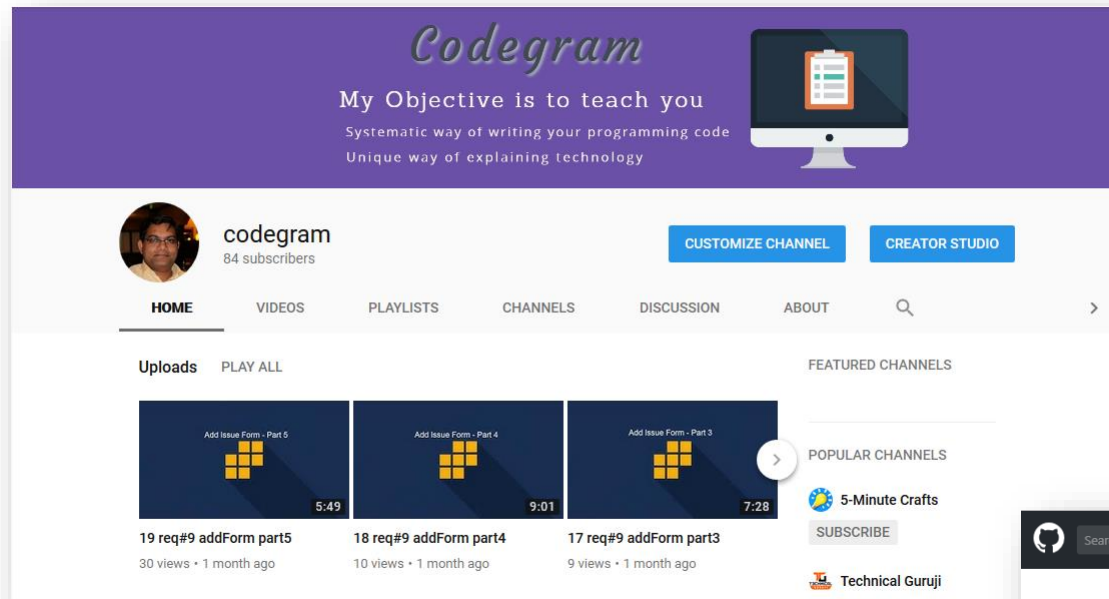
React and Redux POC

Ref :

<https://www.youtube.com/watch?v=DiLVAXIVYR0&list=PL6gx4Cwl9DGBbSLZjvleMwldX8jGgXV6a>

// TODO: Refer Notes on Git Hub

<https://channel9.msdn.com/Series/Visual-Studio-Code-for-Mac-Developers/How-to-intergate-Visual-Studio-Code-and-GitHub>





Thank You

Pankaj Sharma

Sr. Technical Trainer

