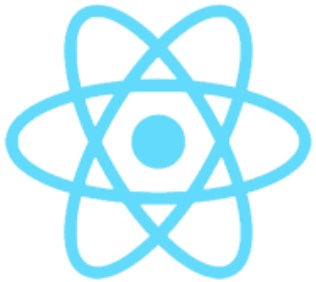
# IssueTracker-app – PART 03



React



Presented by : Pankaj Sharma

# Topics to be covered

- Pure function in React
- this keyword and it’s utility
- About issue-tracker
- Difference between state and props and their use
- Events and Uni Directional Data flow
- How to create and use Stateless functional component in React
- Add issue to a list of issues.
- Delete issue from the list of issues
- Edit issue in the list of issues
- Use of propType

# Edit issue in the list of issues - IssueTracker



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-tracker05/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} 
<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.

# Edit issue in the list of issues - IssueTracker



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

```
Issue-tracker05/src/component/IssueNameComponent.js
-----------------------------------
constructor(props){
super(props);
this.state = {
isEditing:false
}
}
```

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

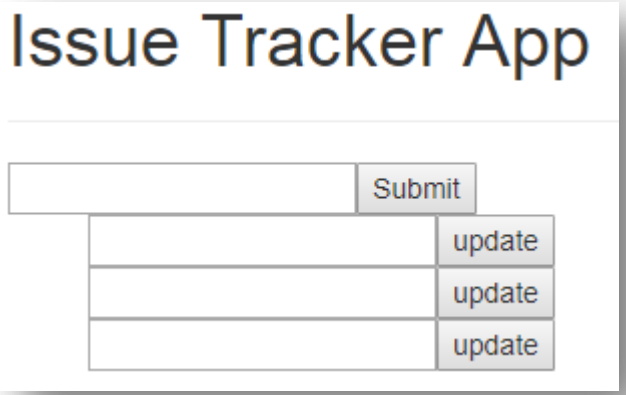
```
Issue-tracker05/src/component/IssueNameComponent.js
-----------------------------------
return( <section>
{/*
<li onClick={()=>{
this.props.clickHandler(this.props.index)
}} className={this.props.issue.completed ? 'completed' : ''}>
{this.props.issue.name} 
<button
onClick={(event)=>{
event.stopPropagation(),
this.props.deleteIssue(this.props.index)
}}>X</button>
</li>
*/} </section> )
```

# Edit issue in the list of issues - IssueTracker

Create update form as shared below.

```
Issue-tracker05/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.



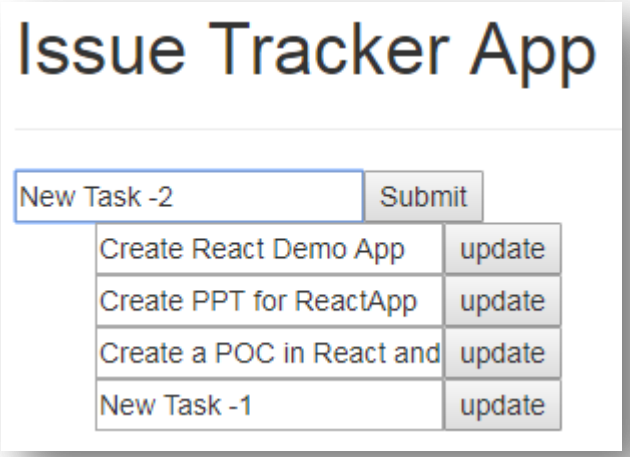
# Edit issue in the list of issues - IssueTracker



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

```
Issue-tracker05/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.



# Edit issue in the list of issues - IssueTracker

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} 
                            <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.

# Edit issue in the list of issues - IssueTracker



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-tracker05/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.

# Edit issue in the list of issues - IssueTracker



```
Issue-tracker05/src/component/IssueNameComponent.js
-----------------------------------
renderForm(){
return(
<form>
<input type="text" defaultValue={this.props.issue.name}/>
<button type="submit">update</button>
</form>
)
}
```

```
Issue-tracker05/src/component/IssueNameComponent.js
-----------------------------------
renderIssues(){
return(
<li onClick={()=>{
this.props.clickHandler(this.props.index)
}} className={this.props.issue.completed ? 'completed' : ''}>
{this.props.issue.name} 
<button
onClick={(event)=>{
event.stopPropagation()
this.props.deleteIssue(this.props.index)
}}>X</button>
</li>
)
}
```

# Edit issue in the list of issues - IssueTracker

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.

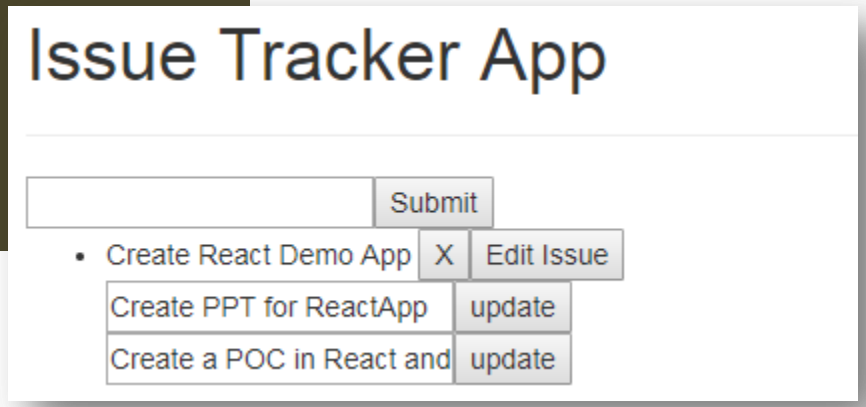
```
Issue-tracker05/src/component/IssueNameComponent.js
-----------------------------------
constructor(props){
. . .
this.toggleState = this.toggleState.bind(this);
. . .
}
```

```
Issue-tracker05/src/component/IssueNameComponent.js
-----------------------------------
toggleState(){
const {isEditing} = this.state;
this.setState({
isEditing:!isEditing
})
}
```

# Edit issue in the list of issues - IssueTracker



```
Issue-tracker05/src/component/IssueNameComponent.js
-----------------------------------
renderIssues(){
return(
<li onClick={()=>{
this.props.clickHandler(this.props.index)
}} className={this.props.issue.completed ? 'completed' : ''}>
{this.props.issue.name} 
<button
onClick={(event)=>{
event.stopPropagation()
this.props.deleteIssue(this.props.index)
}}>X</button>
<button
onClick={(event)=>{
event.stopPropagation()
this.toggleState()
}}>Edit Issue</button>
</li>
)
}
```



# Edit issue in the list of issues _(02) - IssueTracker

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-tracker06/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>
)
}
```

# Edit issue in the list of issues _(02) - IssueTracker

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{
- - - - - - - - -
 renderForm(){
     return(
         <form onSubmit={this.updateIssue}>
             <input type="text"
                    defaultValue={this.props.issue.name}
                    ref={(value) => { this.input = value; }}
                    />
             <button type="submit">update</button>
         </form>
     )
 }
 - - - - - - - - -
}
```

Value will be available to IssueNameComponent using this ref.

input DOM element will be recieved as a value

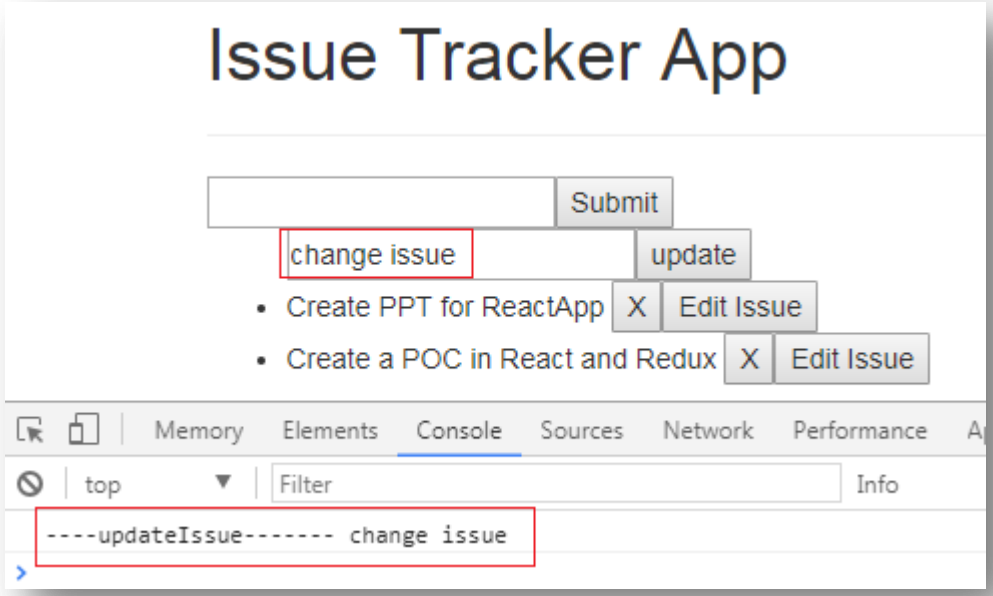
Note : JS concept of adding prop to object at run time.

# Edit issue in the list of issues _(02) - IssueTracker

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

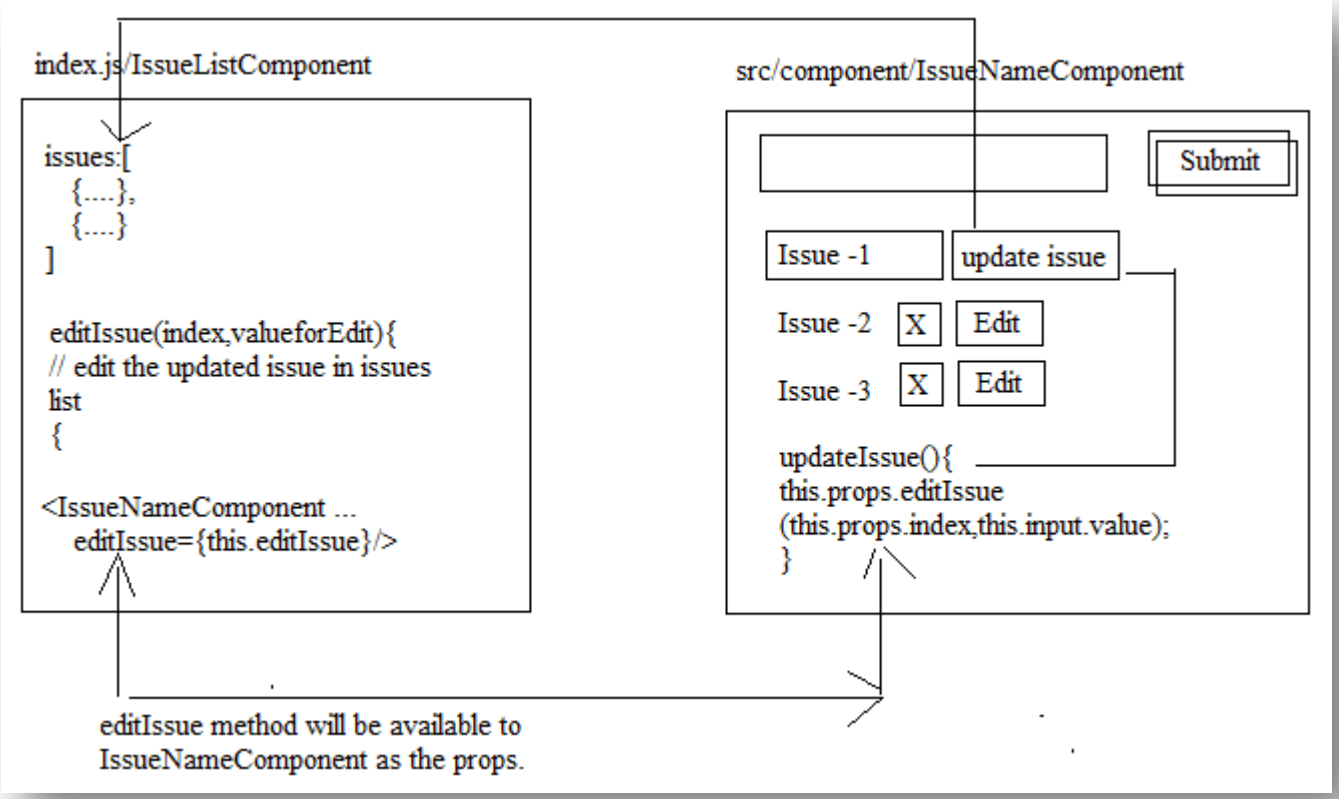
```
updateIssue(event){
    event.preventDefault();
    console.log('----updateIssue-------',this.input.value)
}
```

this will stop the page refresh.



# Edit issue in the list of issues _(02) - IssueTracker

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.



# Edit issue in the list of issues _(02) - IssueTracker

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

```
Issue-tracker06/src/index.js
-----------------------------------
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}/>
```

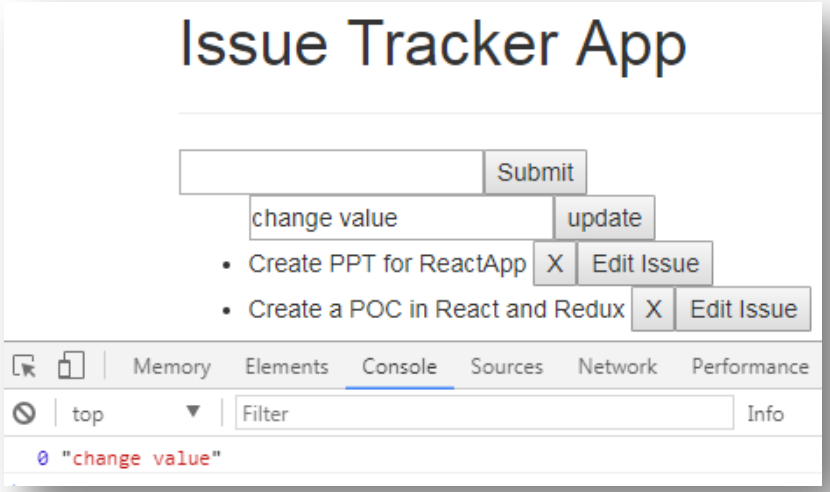
# Edit issue in the list of issues _(02) - IssueTracker

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

```
Issue-tracker06/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-tracker06/index.js
-----------------------------------
editIssue(index,newValue){
console.log(index,newValue);
}
```



# Edit issue in the list of issues _(02) - IssueTracker

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

```
editIssue(index,newValue){
console.log(index,newValue);
let issues=this.state.issues;
let currentIssue=issues[index];
currentIssue['name']=newValue;
this.setState({
  issues
})
```

copy issues array into local issues array

take the current value based on index.

replace the name with the new value.

update the state.
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.

```
updateIssue(event){
    event.preventDefault();
    //console.log('----updateIssue-------',this.input.value)
    this.props.editIssue(this.props.index, this.input.value);
    this.toggleState();
}
```

to toggle the state from update view to list view.

Now check your application, all CRUD operations will work.

# Use of PropTypes- IssueTracker

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.

# Use of PropTypes- IssueTracker

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>
```

Function

React Specific

Object

Function

Number

# Use of PropTypes- IssueTracker

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-tracker07_PropTypes/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.

Lets 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.

# Use of PropTypes- IssueTracker

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    lowPriorityWarning.js:40
```

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-tracker07_PropTypes/src/component/IssueNameComponent.js
----------------------------------
import PropTypes from 'prop-types';
. . .
```

Now restart the application check the browser console.

```
▶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

## Thank You!

**Email:** info@yash.com
**Web:** www.yash.com



Presented by : Pankaj Sharma