

Step by Step Guide To Building React Redux Apps

When I started to learn it, I couldn't find blogs that show "Which part of React Redux to build first?" or how to generally approach building any React-Redux apps. So I went through several example and blogs and came out with general steps as to how to approach building most React Redux Apps.

Please Note: I am using "Mocks" to keep it at a high level and not get into the weeds. I am using the classic [Todo list app](#) as the basis for building **ANY** app. If your app has multiple screens, simply repeat the process for each screen.

Why Redux?

React—A JS library that helps us to divide up our app into multiple components but doesn't clearly specify how to keep track of the data(aka State) and how to deal with all the events(aka Actions) properly.

Redux—A complimentary library to React that provides a way to easily keep the data(State) and the events(Actions).

Essentially Redux allows us to build React app as you are but delegate all the State and Actions to Redux

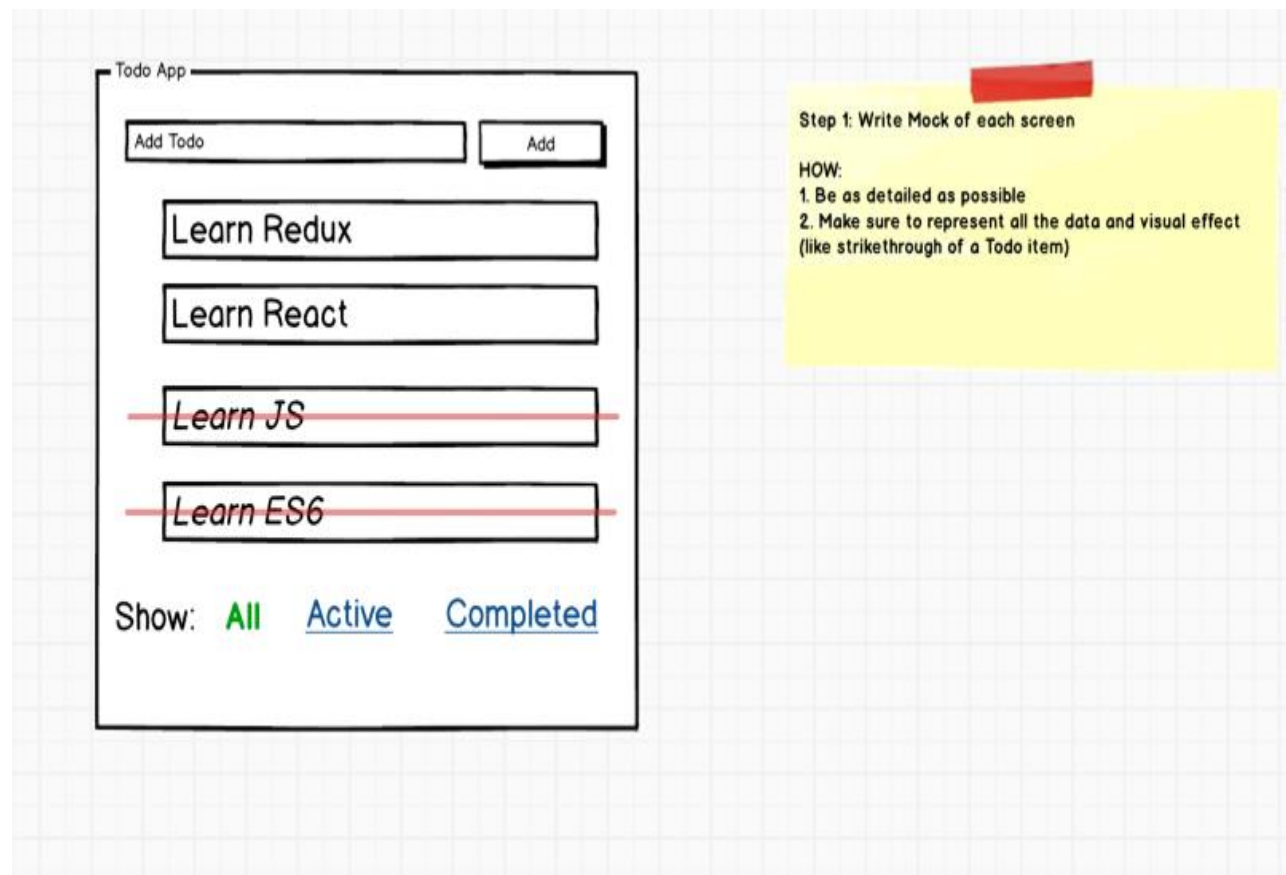
BTW, There are 8 steps for a simple Todo App. The theory is that, earlier frameworks made building Todo apps simple but real apps hard. But React Redux make building Todo apps hard but real productions apps simple.

Let's get started:

STEP 1—Write A Detailed Mock of the Screen

Mock should include all the data and visual effects (like strikethrough the TodoItem, or “All” filter as a text instead of a link)

Please Note: You can click on the pictures to Zoom



STEP 2—Divide The App Into Components

Try to divide the app into chunks of components based on their overall “**purpose**” of each component.

We have 3 components “AddTodo”, “TodoList” and “Filter” component.



Step 2: Divide the app into components

How: Divide them based on their overall purpose. This is usually coarse grained or large block w/in the app.

Our Todo app has 3 main "purposes":

1. Add a new Todo item (AddTodo component)
2. Show List of Todo items (TodoList cmp)
3. Allows filtering Todos to show (Filter cmp)

Redux Terms: "Actions" And "States"

Every component does two things:

1. Render DOM based on some data. This data is called as a "**state**".
2. Listen to the user and other events and send them to JS functions. These are called "**Actions**".

STEP 3—List State and Actions For Each Component

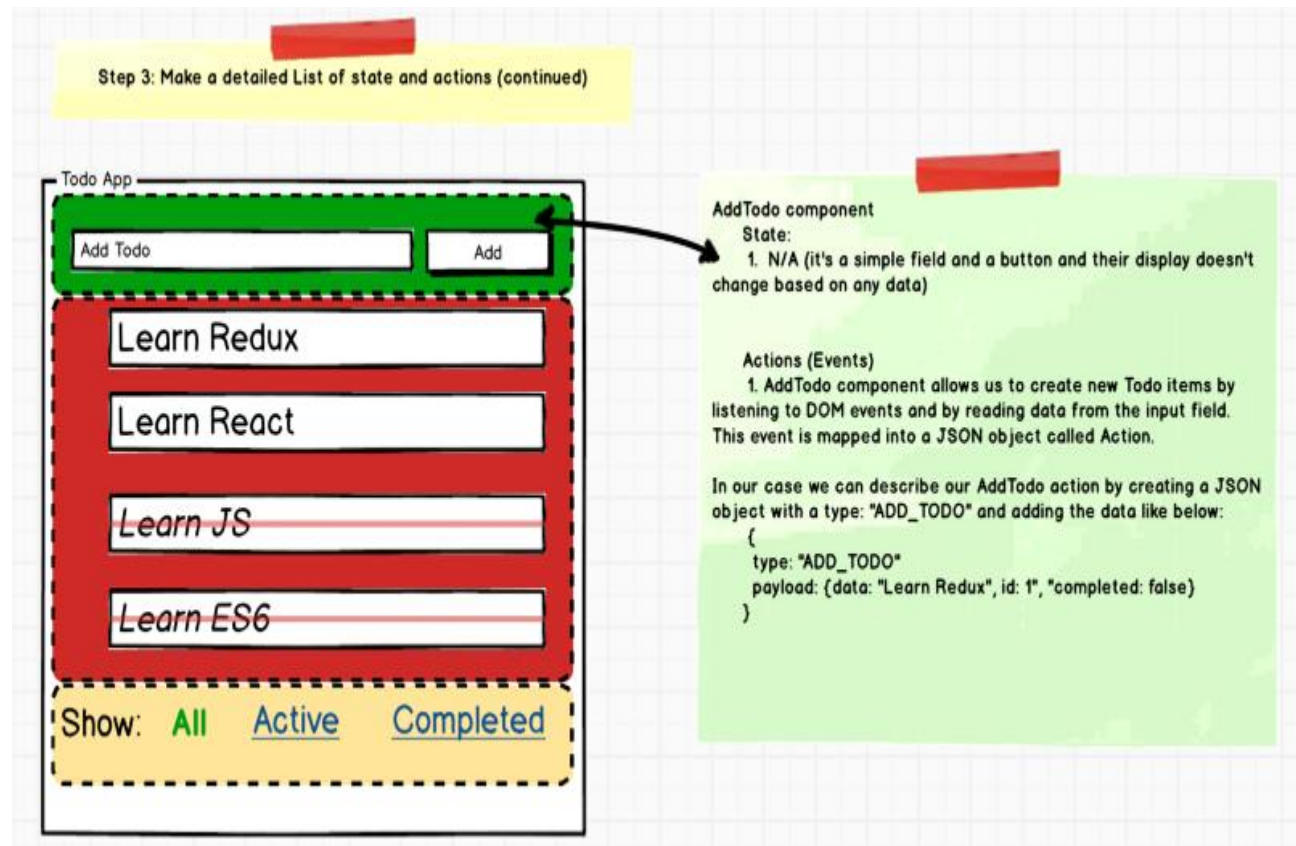
Make sure to take a careful look at each component from STEP 2, and list of States and Actions for each one of them.

We have 3 components "AddTodo", "TodoList" and "Filter" component. Let's list Actions and States for each one of them.

3.1 AddTodo Component—State And Actions

In this component, we have no state since the component look and feel doesn't change based on any data but it needs to let other components know when the user creates a new Todo. Let's call this action "**ADD_TODO**".

Please Note: You can click on the pictures to Zoom



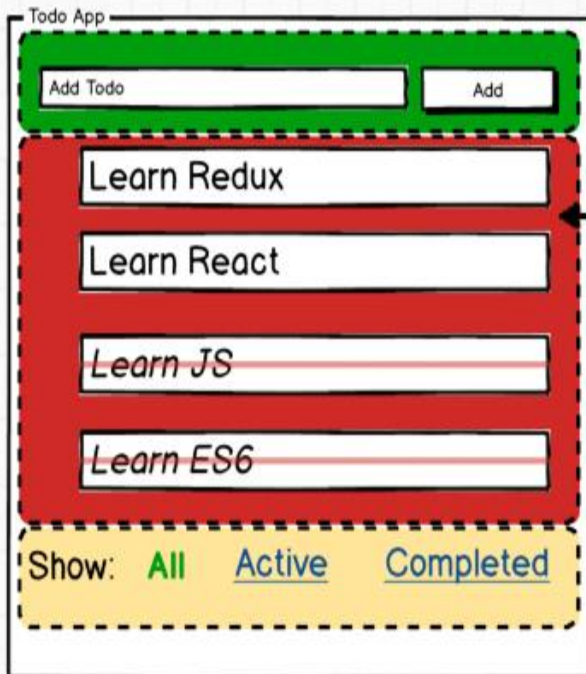
3.2 TodoList Component—State And Actions

TodoList component needs an array of Todo items to render itself, so it needs a state, let's call it **Todos** (Array). It also needs to know which "Filter" is turned on to appropriately display (or hide) Todo items, it needs another state, let's call it "**VisibilityFilter**" (boolean).

Further, it allows us to toggle Todo item's status to completed and not completed. We need to let other components know about this toggle as well. Let's call this action "**TOGGLE_TODO**".

Step 3: Make a detailed List of data and events

Continued...



TodoList component

State:

1. Todos Array
2. Visibility Filter - This tells what kind of Todo items to show and hide. Note: This comes from "Filter" component.

Actions (Events)

TodoList has only one action, it allows users to toggle a Todo item's "completed" status. When the user clicks on a Todo item, it needs to tell other components and DB etc to toggle the item's status.

Again we need to describe the ToggleTodo action JSON object by giving it a name and payload. In this case it may look like below:

```
{
  type: "TOGGLE_TODO"
  payload: {id: <todoItem's id>}
}
```

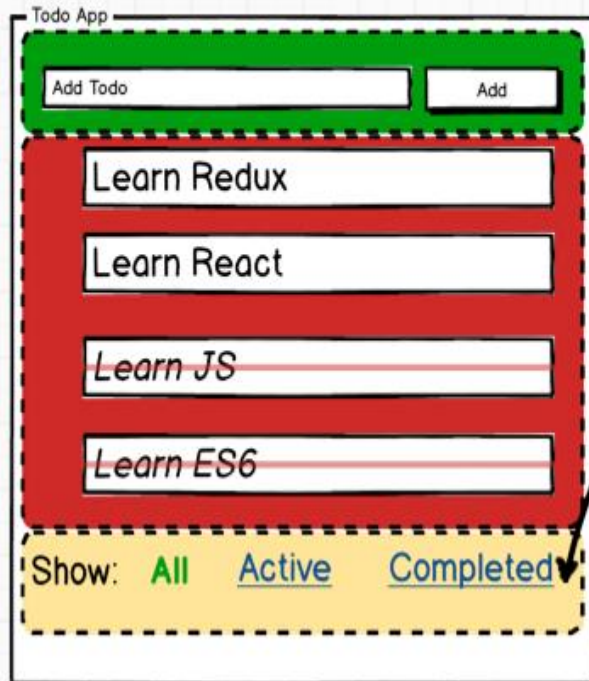
3.3 Filter Component — State And Actions

Filter component renders itself as a Link or as a simple text depending on if it's active or not. Let's call this state as "**CurrentFilter**".

Filter component also needs to let other components know when a user clicks on it. Let's call this actions, "**SET_VIBILITY_FILTER**"

Step 3: Make a detailed List of data(aka state) and events(aka actions)

Continued...



Filter component

State(data):

1. CurrentFilter - A string that tells which filter to display as active (simple text) v/s which ones to show as clickable link. For example "Active" and "Completed" links in our mock.

Actions (Events)

Filter component also has only one action. It simply listens to user clicks on filter links and tells other components which link was clicked.

Again we need to describe the action, i.e. give it a name and payload. In this case it may look like below:

```
{
  type: "SET_VISIBILITY_FILTER"
  payload: {filter: "Completed"}
}
```

Redux Term: "Action Creators"

Action Creators are simple functions whose job is to receive data from the DOM event, format it as a formal JSON "Action" object and return that object (aka "Action"). This helps us to formalize how the data/payload look.

Further, it allows any other component in the future to also send(aka "dispatch") these actions to others.

STEP 4—Create Action Creators For Each Action

We have total 3 actions: ADD_TODO, TOGGLE_TODO and SET_VISIBILITY_FILTER. Let's create action creators for each one of them.

//1. Takes the text from AddTodo field and returns proper "Action" JSON to send to other components.

```
export const addTodo = (text) => {  
  return {  
    type: 'ADD_TODO',  
    id: nextTodoId++,  
    text, //<--ES6. same as text:text, in ES5  
    completed: false //<-- initially this is set to false  
  }  
}
```

//2. Takes filter string and returns proper "Action" JSON object to send to other components.

```
export const setVisibilityFilter = (filter) => {  
  return {  
    type: 'SET_VISIBILITY_FILTER',  
    filter  
  }  
}
```

//3. Takes Todo item's id and returns proper "Action" JSON object to send to other components.

```
export const toggleTodo = (id) => {  
  return {  
    type: 'TOGGLE_TODO',  
    id  
  }  
}
```

Redux Term: "Reducers"

Reducers are functions that take "state" from Redux and "action" JSON object and returns a new "state" to be stored back in Redux.

1. Reducer functions are called by the "Container" containers when there is a user action.
2. If the reducer changes the state, Redux passes the new state to each component and React re-renders each component

For example the below function takes Redux' state (an array of previous todos), and returns a ****new**** array of todos (new state) w/ the new Todo added if action's type is "ADD_TODO".

```
const todo = (state = [], action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return
        [...state, {id: action.id, text: action.text,
completed:false}];
  }
}
```

STEP 5—Write Reducers For Each Action

Note: Some code has been stripped for brevity. Also I'm showing SET_VISIBILITY_FILTER along w/ ADD_TODO and TOGGLE_TODO for simplicity.

```
const todo = (state, action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [...state, {id: action.id, text: action.text,
completed:false}]

    case 'TOGGLE_TODO':
      return state.map(todo =>
        if (todo.id !== action.id) {
          return todo
        }
        return Object.assign({},
          todo, {completed: !todo.completed})
      )

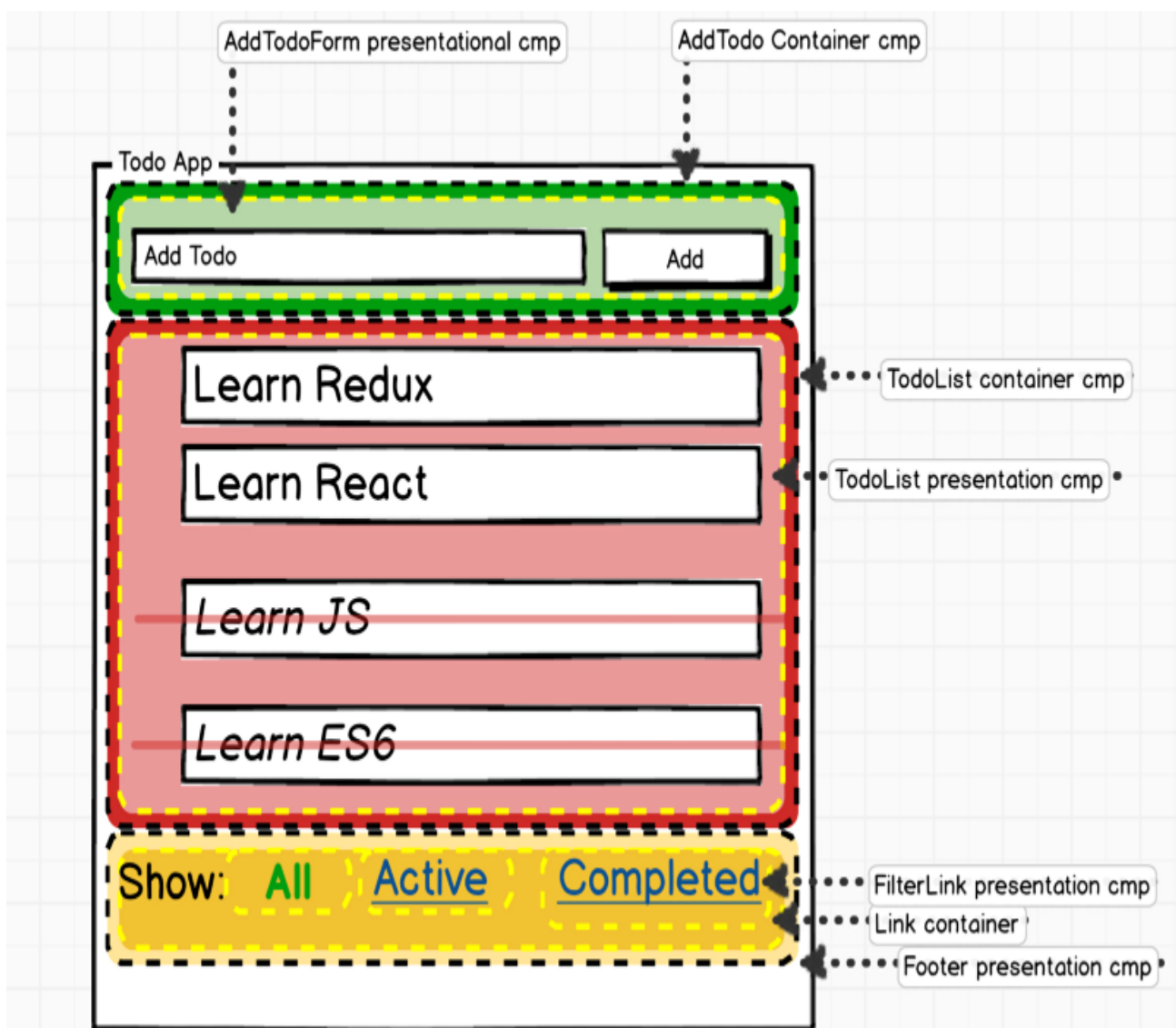
    case 'SET_VISIBILITY_FILTER': {
      return action.filter
    }

    default:
      return state
  }
}
```


Redux Term: “Presentational” and “Container” Components

Keeping React and Redux logic inside each component can make it messy, so Redux recommends creating a dummy presentation only component called “Presentational” component and a parent wrapper component called “Container” component that deals w/ Redux, dispatch “Actions” and more.

The parent Container then passes the data to the presentational component, handle events, deal with React on behalf of Presentational component.



Legend: Yellow dotted lines = "Presentational" components. Black dotted lines = "Container" components.

STEP 6—Implement Every Presentational Component

It's now time for us to implement all 3 Presentational component.

6.1—Implement AddTodoForm Presentational Component

Please Note: Click on the pictures to Zoom and read

Step 6: Create Presentation components for each component (continued)

AddTodoForm Component:

This component renders a field and button. It receives onSubmit callback function (from parent Container component) and when the user submits new Todo, it sends calls that onSubmit function w/ the Todo text.



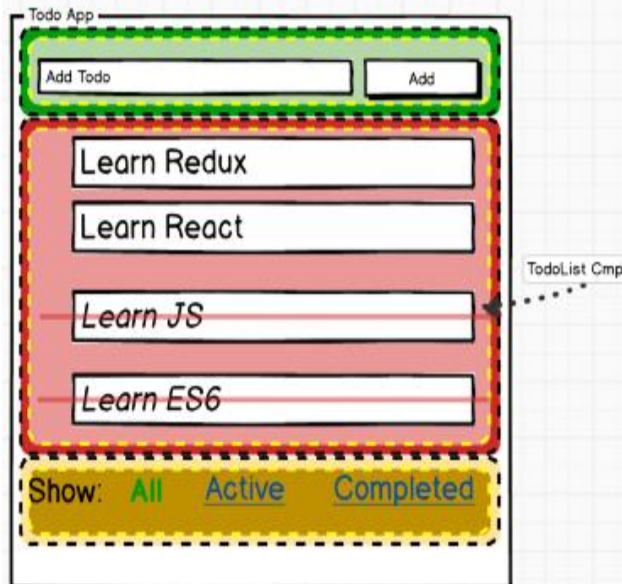
```
let AddTodoForm = ({ onSubmit }) => {  
  let input  
  
  return (  
    <div>  
      <form onSubmit={e => { onSubmit(input.value) }}>  
        <input ref={node => { input = node }} />  
        <button type="submit"> Add Todo</button>  
      </form>  
    </div>  
  )  
}  
  
export default AddTodoForm
```

6.2—Implement TodoList Presentational Component

Step 6: Create Presentation components for each component
(continued)

TodoList Component:

This component renders a list of Todo items. It receives "todos" array and "onTodoClick" callback function (from parent Container component). when the user clicks on Todo item, it sends calls that onTodoClick function w/ the Todo item id to toggle it's status.



```
const TodoList = ({ todos, onTodoClick }) => {  
  if(todos.length === 0)  
    return <div>Add Todos</div>  
  
  return <ul>  
    {todos.map(todo =>  
      <Todo key={todo.id} {...todo} onClick={() => onTodoClick(todo.id)} />  
    )}  
  </ul>  
}  
  
export default TodoList
```

6.3 — Implement Link Presentational Component

Step 6: Create Presentation components for each component
(continued)

Link Component:

This component renders an individual link. There are 3 of them. It receives "active" boolean and renders either a text or a link. It receives "children" property to display the name of the link. It also receives "onClick" callback function that is called when the link is clicked.



```
const Link = ({ active, children, onClick }) => {  
  if (active) {  
    return <span>{children}</span>  
  }  
  
  return <a href="#" onClick={e => {onClick()}}>{children}</a>  
}  
  
export default Link
```

Note: In the actual code, **Link** presentational component is wrapped in "**FilterLink**" container component. And then 3 "FilterLink" components are then displayed inside "Footer" presentational component.

STEP 7—Create Container Component For Some/All Presentational Component

It's finally time to wire up Redux for each component!

7.1 Create Container Component—AddTodo

Step 7: Create Container components for each component
(continued)

AddTodo Container Component.

It implements the "onSubmit" callback function and pass it as a property to the child Presentation component. When the user submits a new Todo, this "onSubmit" will be called.

It acts like a bridge between Redux and AddTodoForm.



AddTodo Container

```
//Implement and send onSubmit callback as a property to
Presentation component
const mapDispatchToProps = (dispatch) => {
  return {
    onSubmit: (text) => {
      dispatch(addTodo(text))
    }
  }
}
```

```
//Connect Redux to the Container and wrap "AddTodoForm"
presentation component
let AddTodo = connect(null, mapDispatchToProps)(AddTodoForm)

export default AddTodo
```

7.2 Create Container Component—TodoList Container

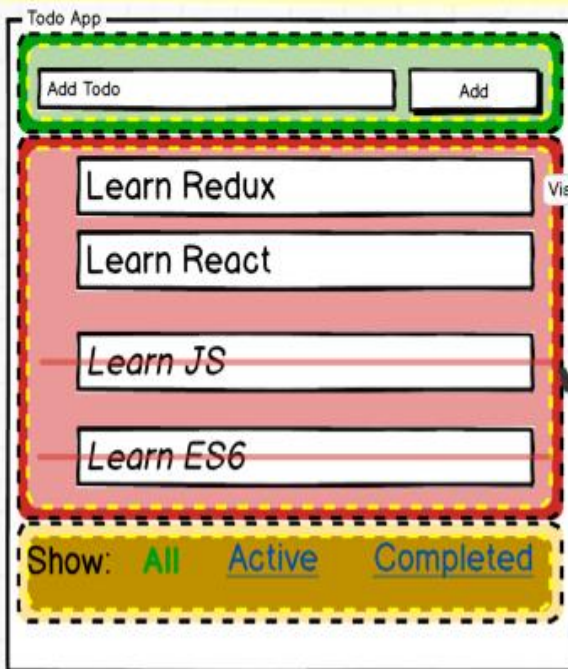
Step 7: Create Container components for each component (continued)

VisibleTodoList Container Component.

It is the bridge between Redux and "TodoList" Presentation component.

It receives filter type and list of Todos from Redux, generates a new array of Todos based on the filter, then passes the filtered(visible) todos to TodoList presentation component.

It implements the "onTodoClick" callback function to toggle Todo item.



```
...  
  
//Get todos and visibilityFilter value from global Redux store,  
//filter the valid todos to display via getVisibleTodos and then pass it  
//to Presentational component(TodoList) as props.  
const mapStateToProps = (state) => {  
  return {  
    todos: getVisibleTodos(state.todos, state.visibilityFilter)  
  }  
}  
  
const mapDispatchToProps = (dispatch) => {  
  return {  
    onTodoClick: (id) => {  
      dispatch(toggleTodo(id))  
    }  
  }  
}  
  
//Connect Redux and TodoList  
it VisibleTodoList = connect(  
  mapStateToProps,  
  mapDispatchToProps  
) (TodoList)  
  
export default VisibleTodoList
```

7.3 Create Container Component — Filter Container

Step 7: Create Container components for each component
(continued)

FilterLink Container Component.

There are 3 FilterLink Components for each filter.

It is the bridge between Redux and "Link" Presentation component..

It receives "filter" props from it's own tag (e.g. SHOW_ALL)

```
<FilterLink filter="SHOW_ALL">  
  All  
</FilterLink>
```



```
...  
...  
// If Redux' GLOBAL filter is same as it's own(tag-level prop) filter,  
// send a boolean prop called "active" as true to "Link" component  
const mapStateToProps = (state, ownProps) => {  
  return {  
    active: ownProps.filter === state.visibilityFilter  
  }  
}  
// Implement onClick  
const mapDispatchToProps = (dispatch, ownProps) => {  
  return {  
    onClick: () => {  
      dispatch(setVisibilityFilter(ownProps.filter))  
    }  
  }  
}  
  
const FilterLink = connect(  
  mapStateToProps,  
  mapDispatchToProps  
) (Link)  
  
export default FilterLink
```

*Note: In the actual code, **Link** presentational component is wrapped in "**FilterLink**" container component. And then 3 "FilterLink" components are then arranged and displayed inside "**Footer**" presentational component.*

STEP 8—Finally Bring Them All Together

```
import React from 'react' // ← Main React library  
import { render } from 'react-dom' // ← Main react  
library  
import { Provider } from 'react-redux' //← Bridge React  
and Redux  
import { createStore } from 'redux' // ← Main Redux  
library
```

```

import todoApp from './reducers' // ← List of Reducers we
created
//Import all components we created earlier
import AddTodo from '../containers/AddTodo'
import VisibleTodoList from
'../containers/VisibleTodoList'
import Footer from './Footer' // ← This is a
presentational component that contains 3 FilterLink
Container comp
//Create Redux Store by passing it the reducers we
created earlier.
let store = createStore(reducers)
render(
  <Provider store={store}> ← The Provider component from
react-redux injects the store to all the child components
  <div>
    <AddTodo />
    <VisibleTodoList />
    <Footer />
  </div>
  </Provider>,
  document.getElementById('root') //←-- Render to a div w/
id "root"
)

```