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

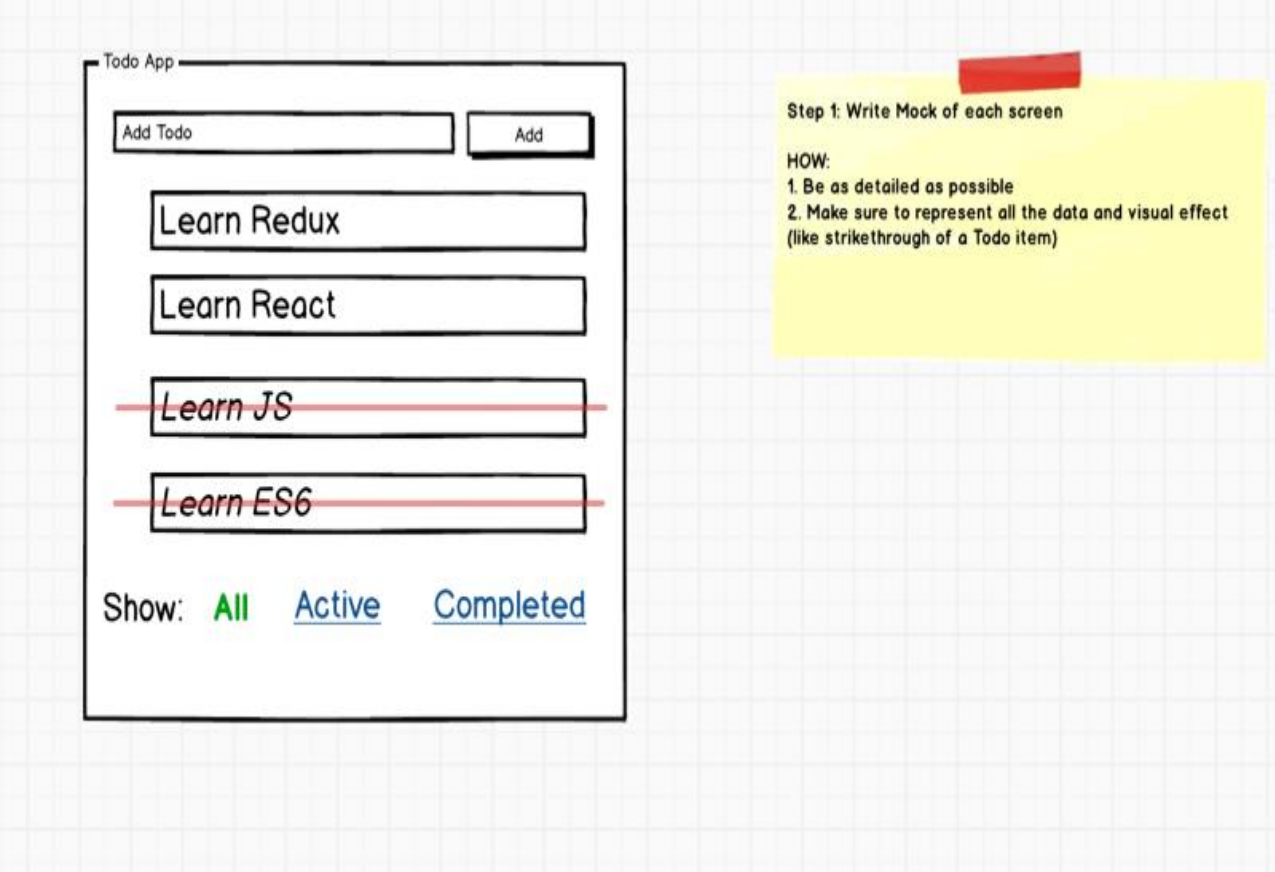
BTW, There are8 steps for a simple Todo App. The theory is that, earlier frameworks made building Todo apps simple but real apps hard. But React Redux make building Todo apps hard but real productions apps simple.

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)

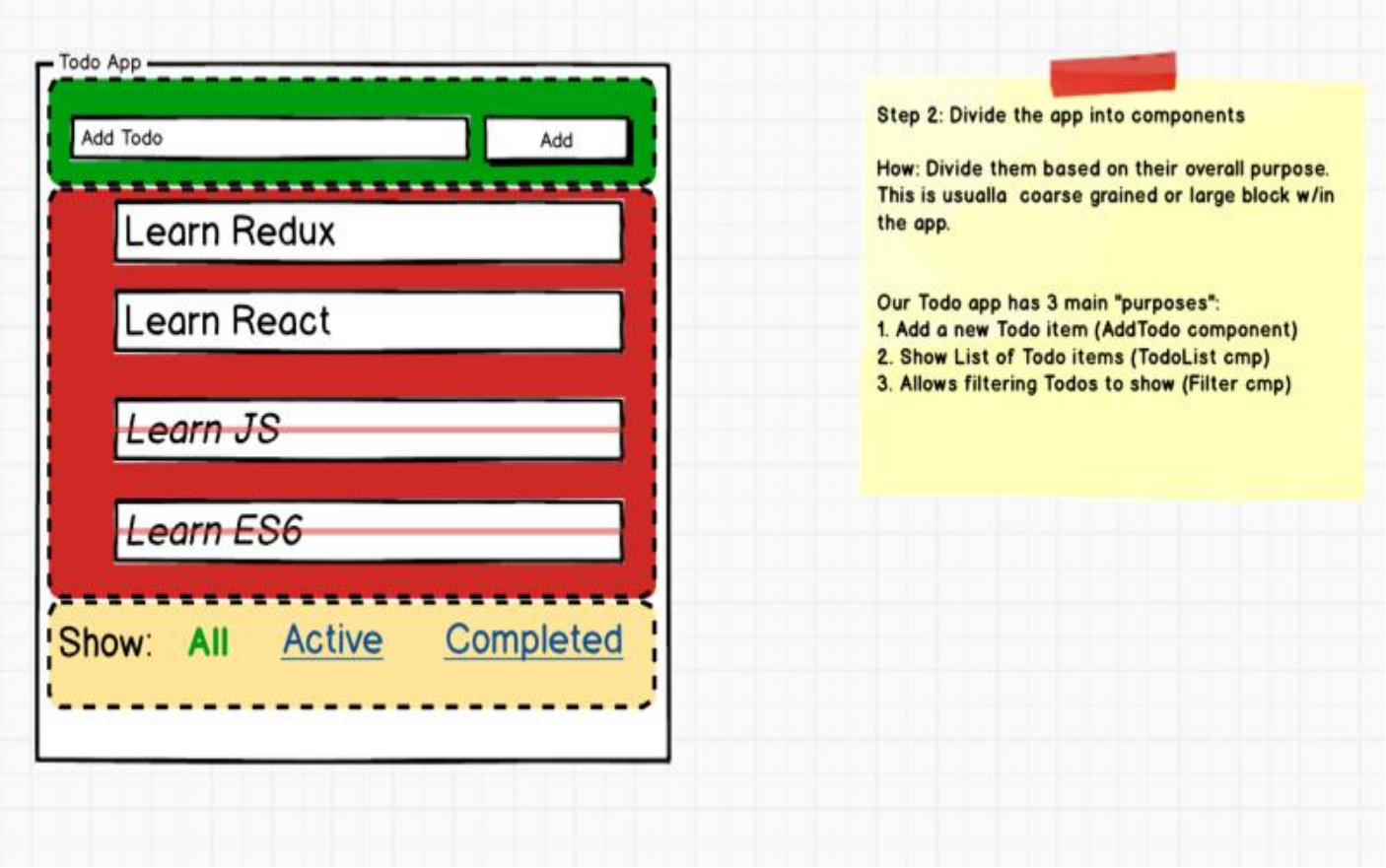
Please Note: You can click on the pictures to Zoom



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.



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

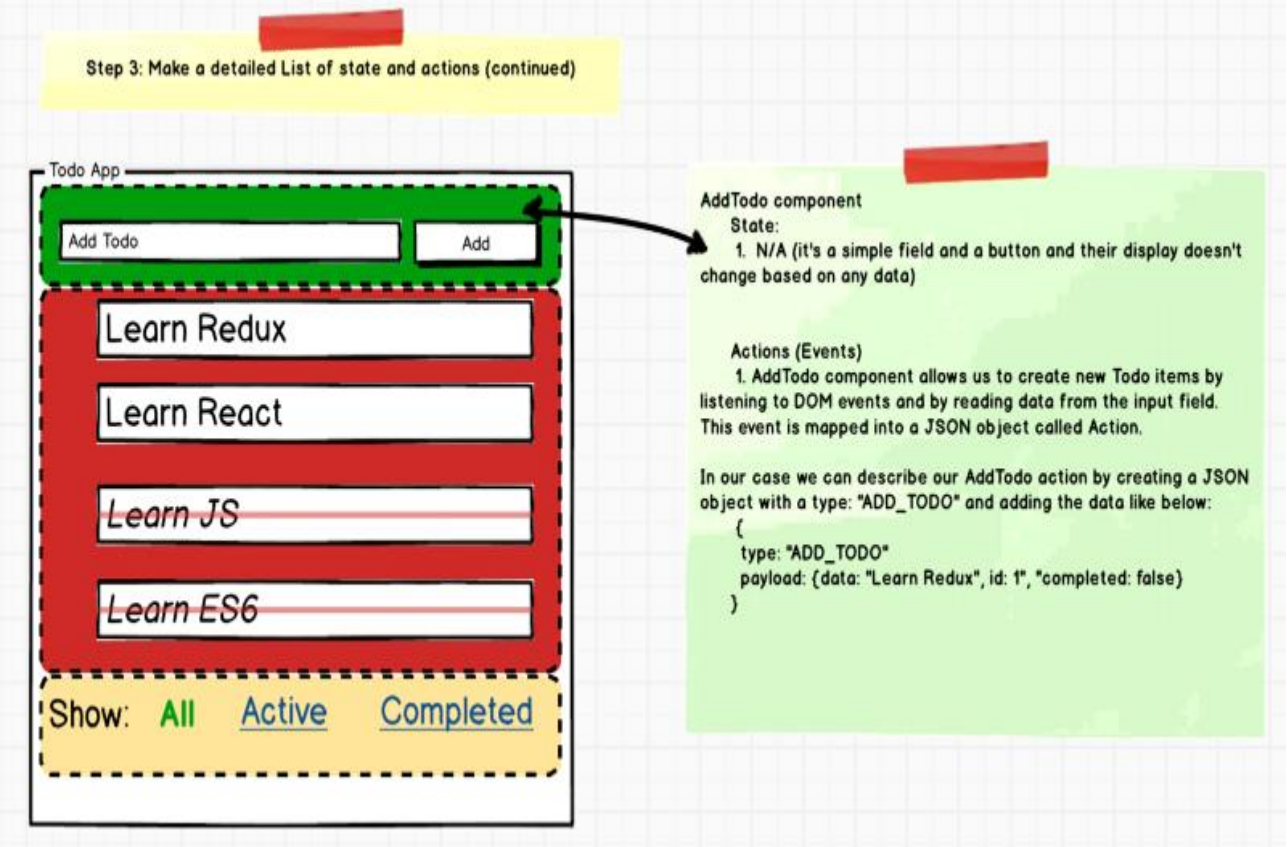
Make sure to take a careful look at each component from STEP 2, and list of States and Actions for each one of them.

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**".

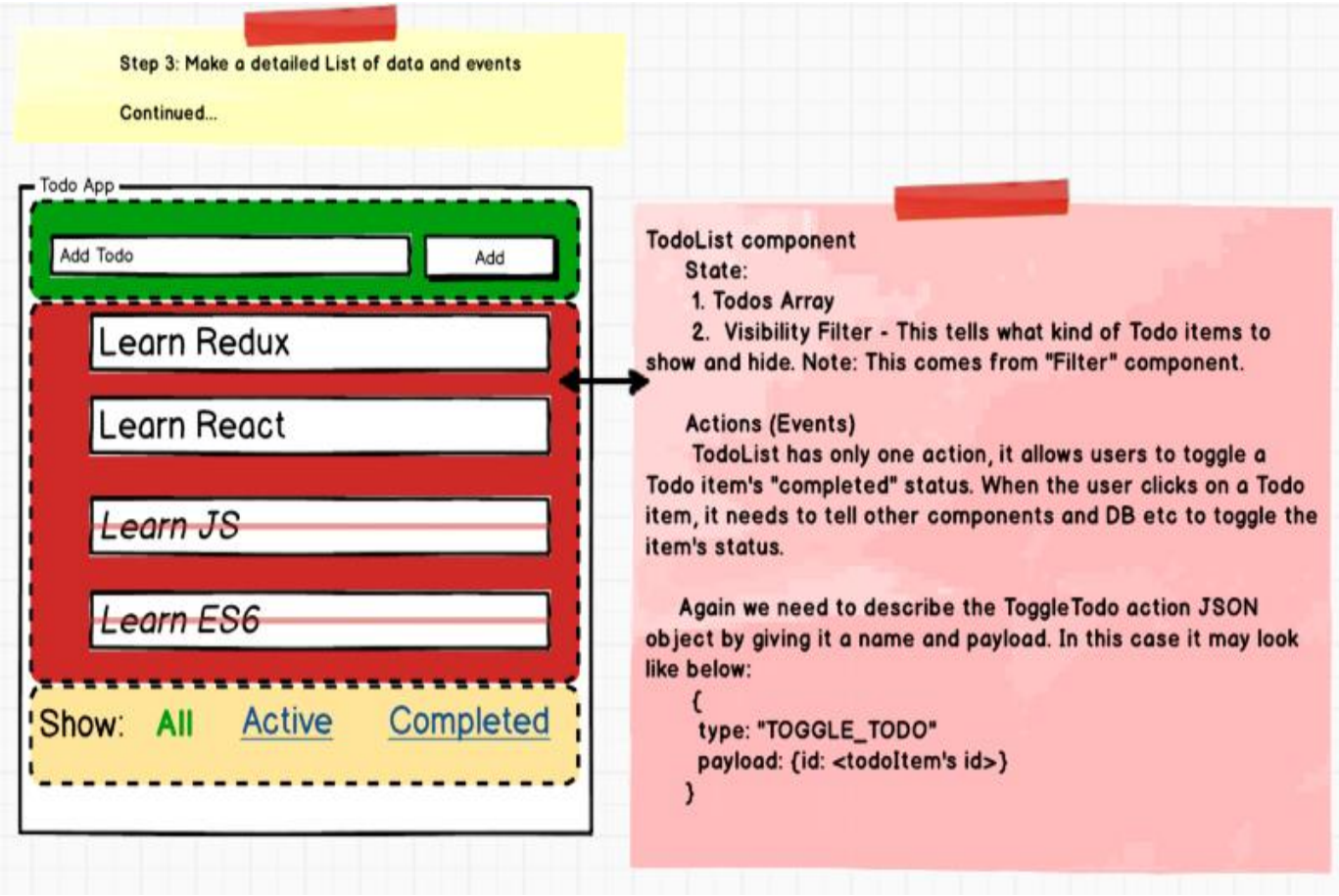
Please Note: You can click on the pictures to Zoom



3.2 TodoList Component—State And Actions

TodoList component needs an array of Todo items to render itself, so it need 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).

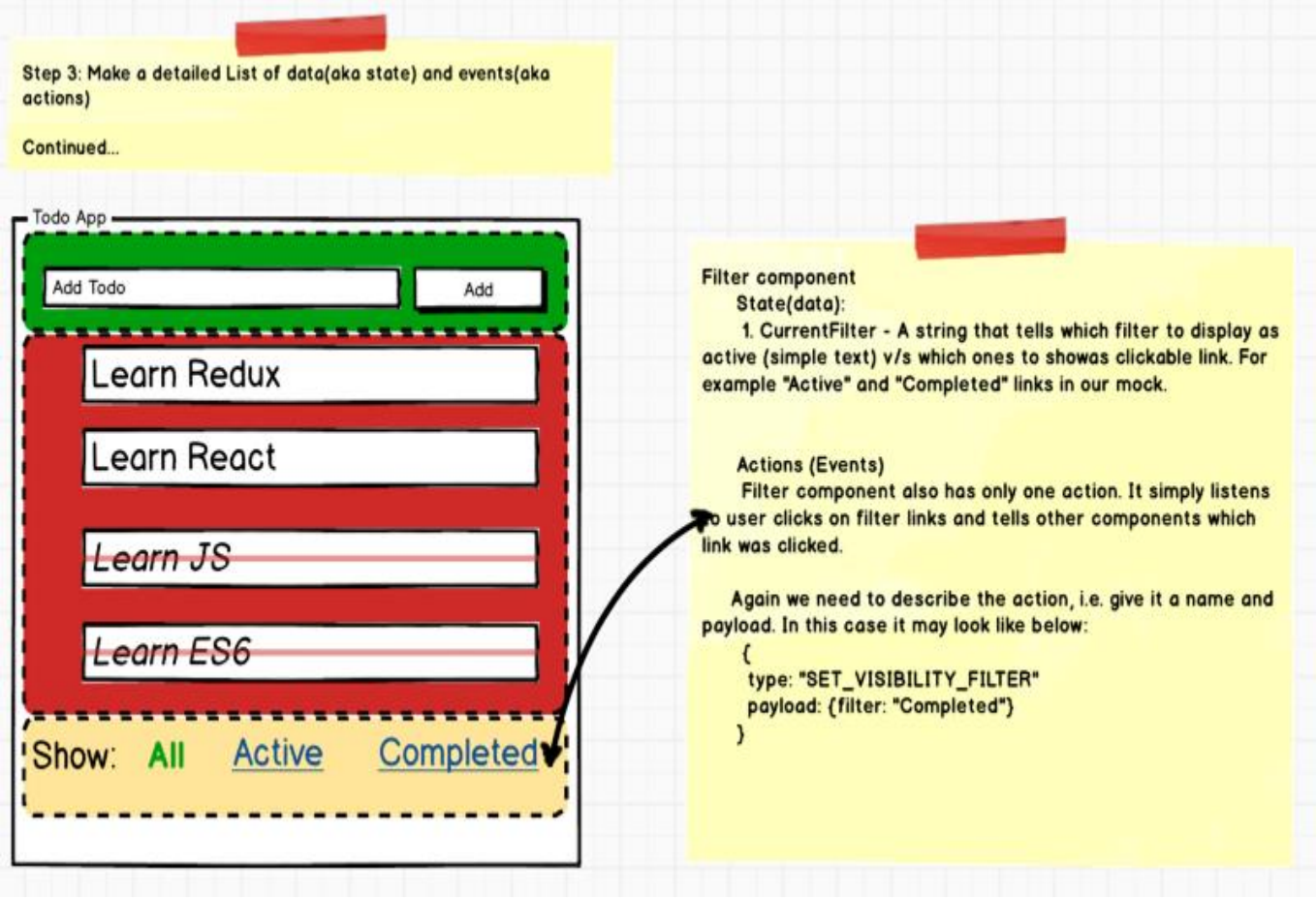
Further, it allows us to toggle Todo item's status to completed and not completed. We need to let other components know about this toggle as well. Let's call this action "**TOGGLE_TODO**"



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**"



Redux Term: "Action Creators"

Action Creators are simple functions who 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

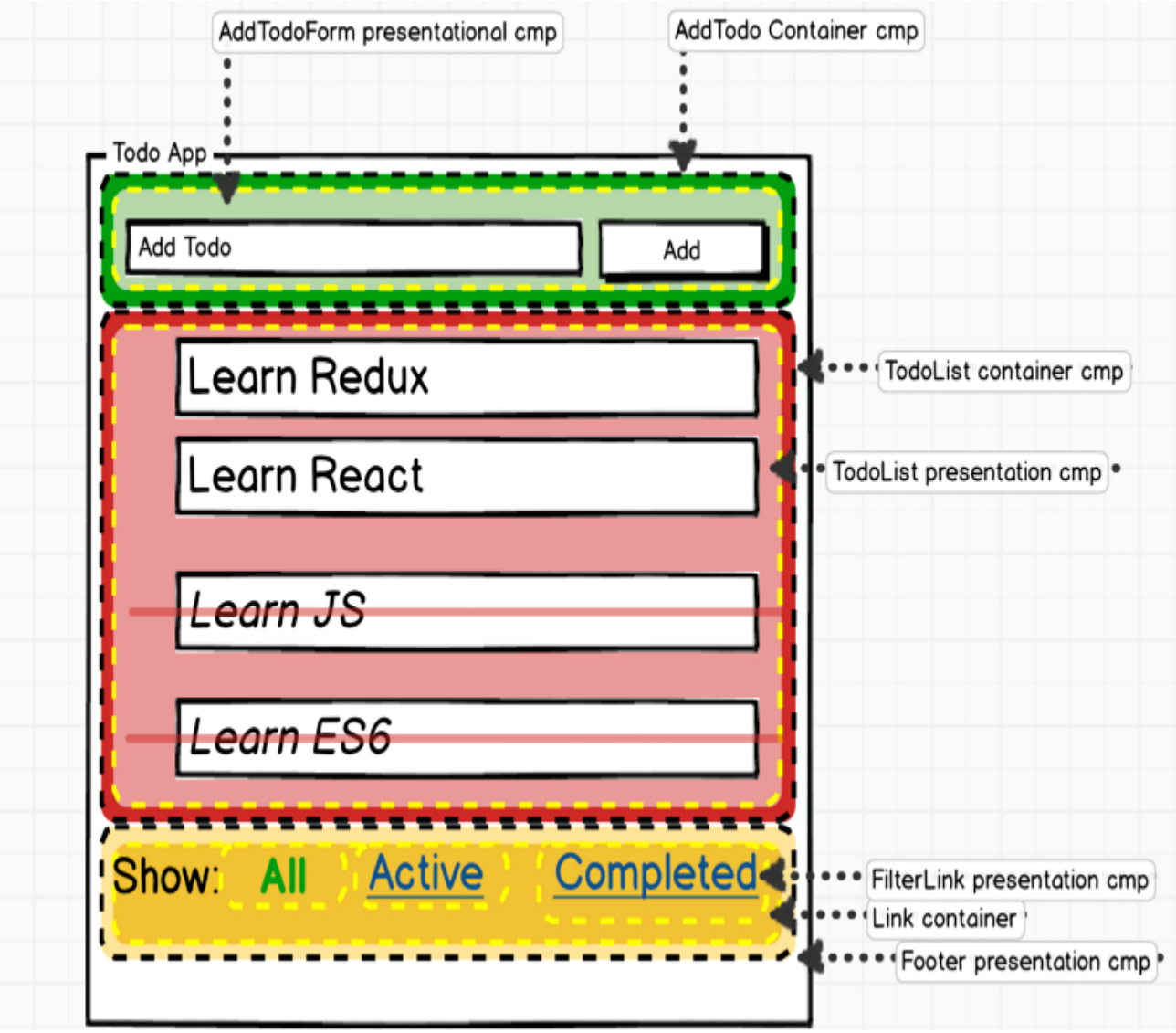
Note: Some code has been stripped for brevity. Also I'm showing SET_VISIBILITY_FILTER along w/ ADD_TODO and TOGGLE_TODO for simplicity.

```
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.



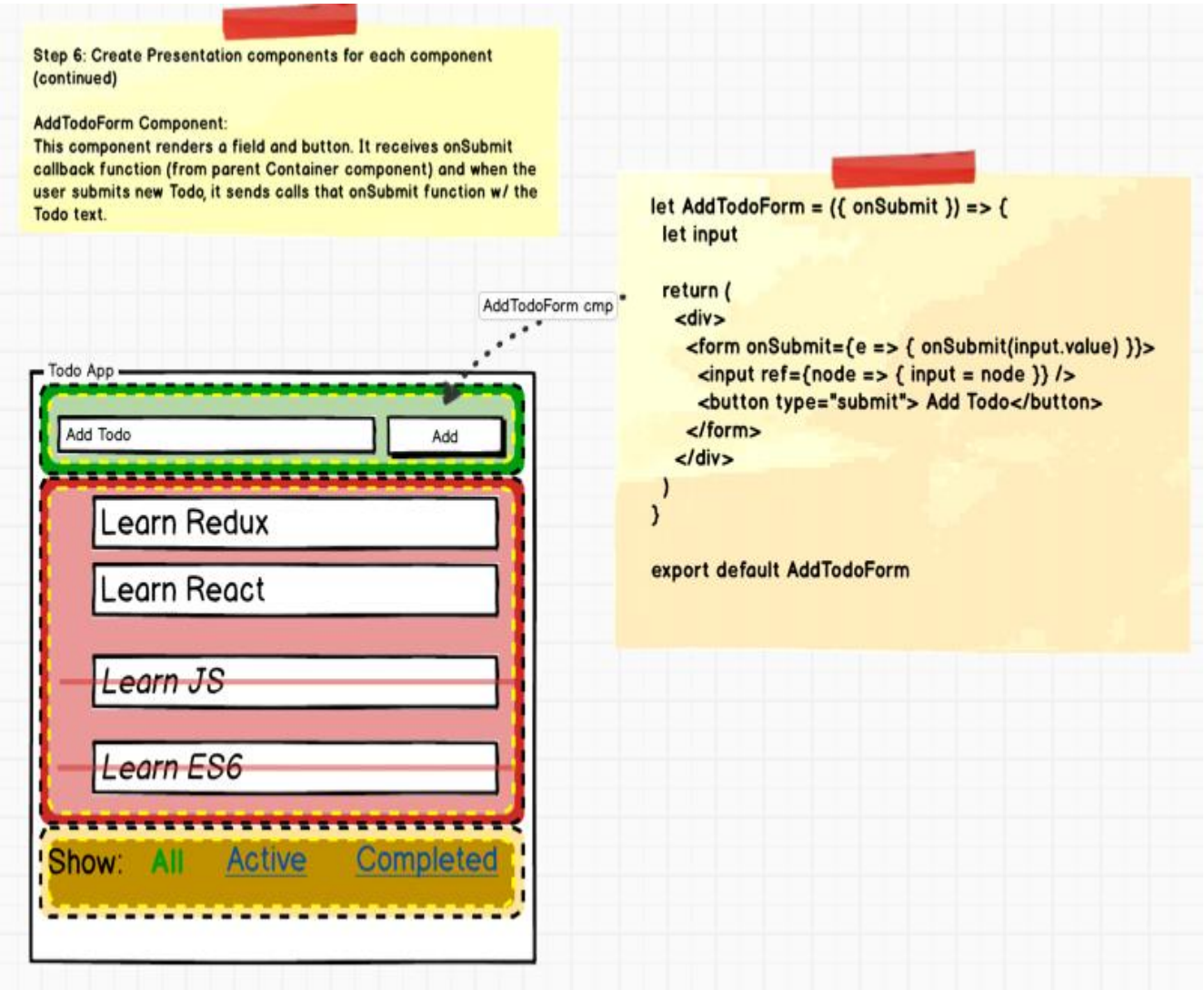
Legend: Yellow dotted lines = "Presentational" components. Black dotted lines = "Container" components.

STEP 6 — Implement Every Presentational Component

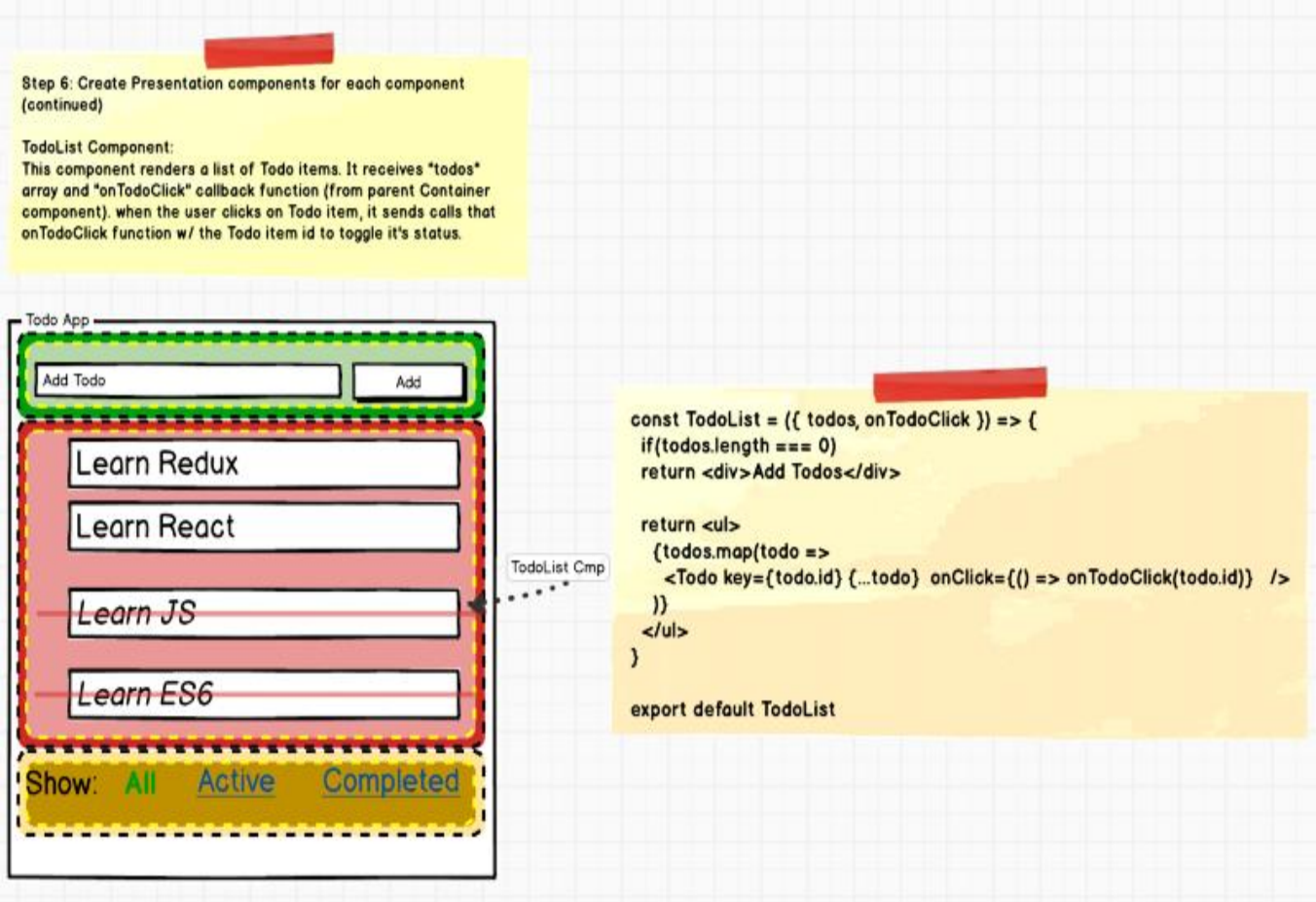
It's now time for us to implement all 3 Presentational component.

6.1 — Implement AddTodoForm Presentational Component

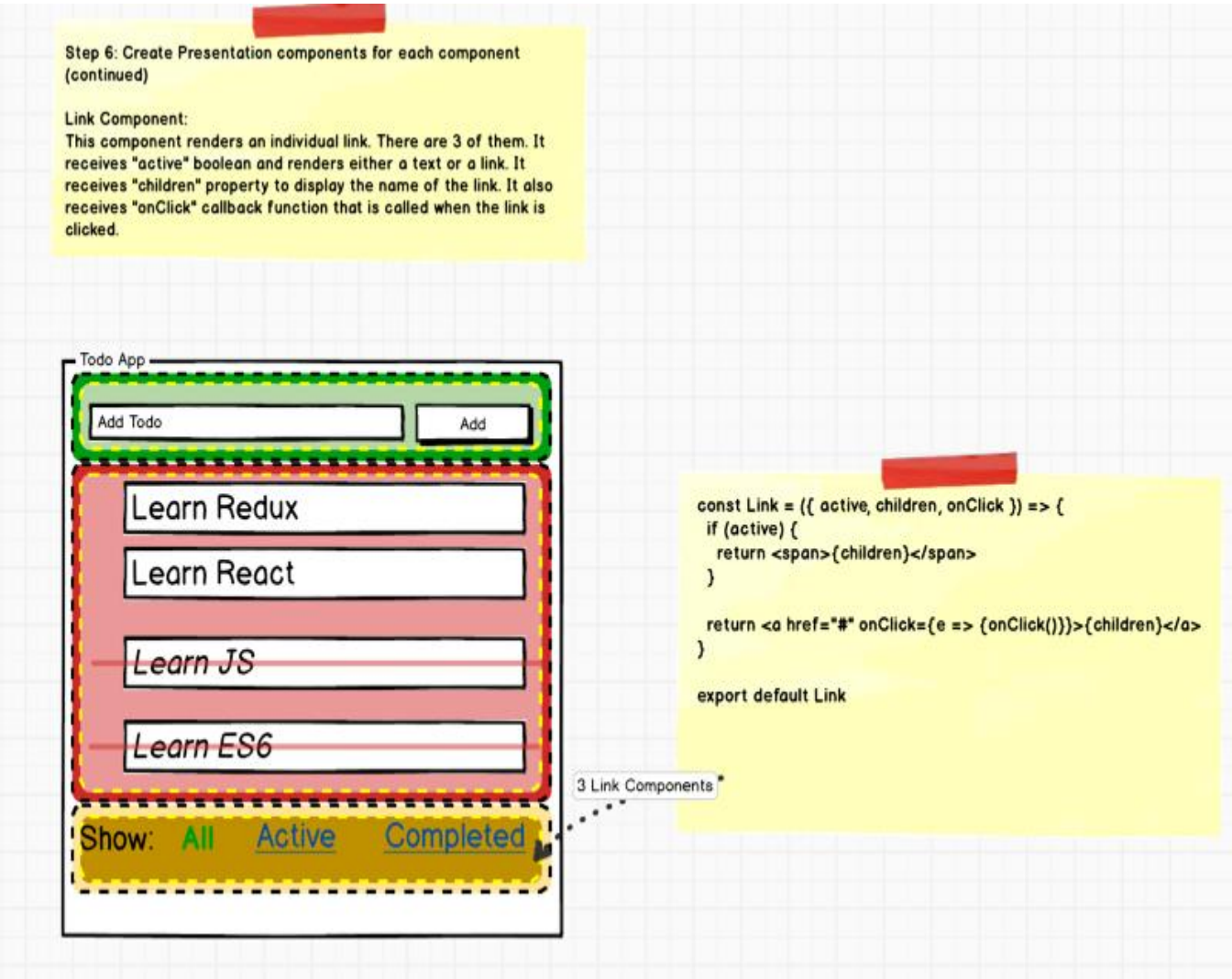
Please Note: Click on the pictures to Zoom and read



6.2 — Implement TodoList Presentational Component



6.3 — Implement Link Presentational Component

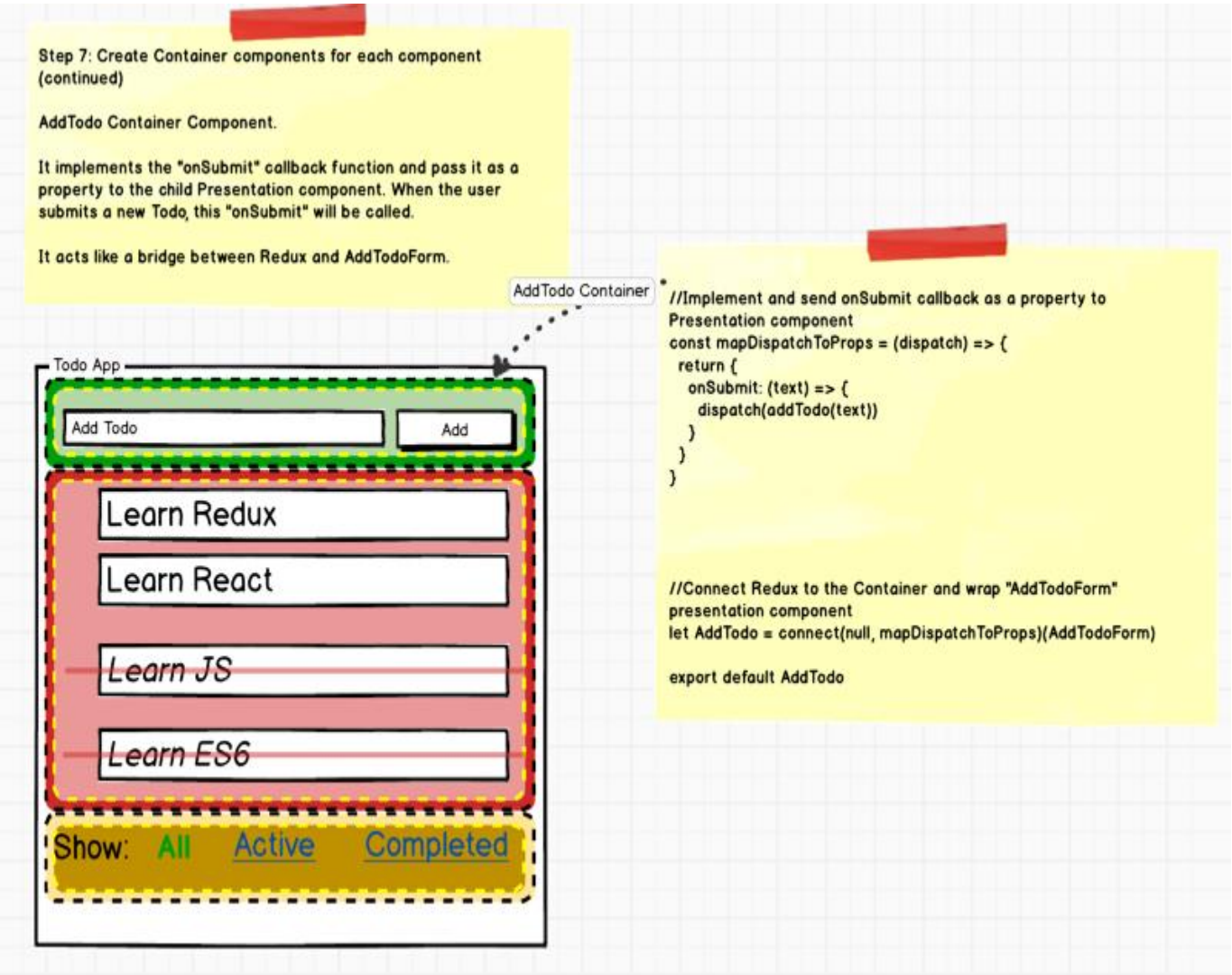


Note: In the actual code, **Link** presentational component is wrapped in "**FilterLink**" container component. And then 3 "FilterLink" components are then displayed inside "Footer" presentational component.

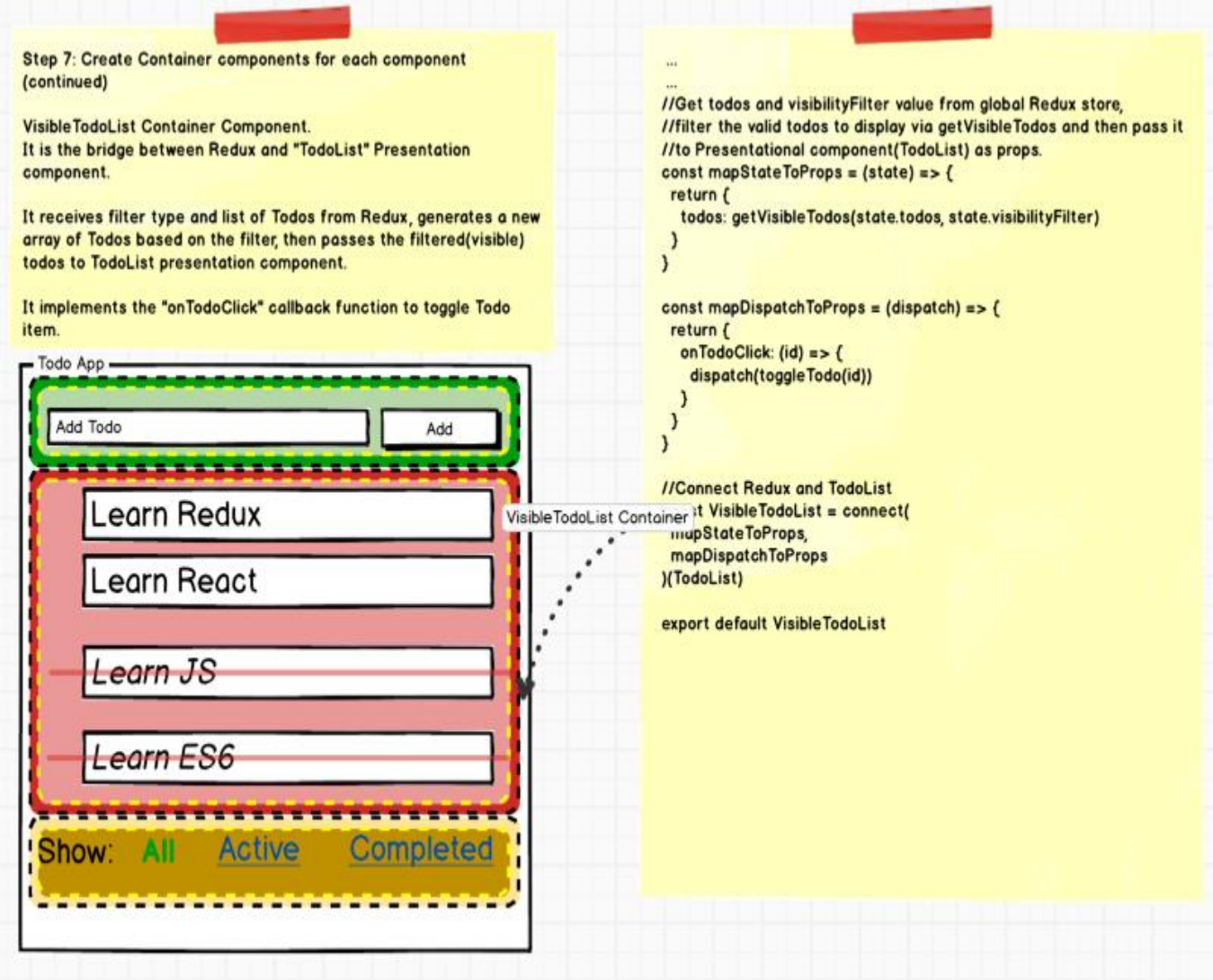
STEP 7 — Create Container Component For Some/All Presentational Component

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

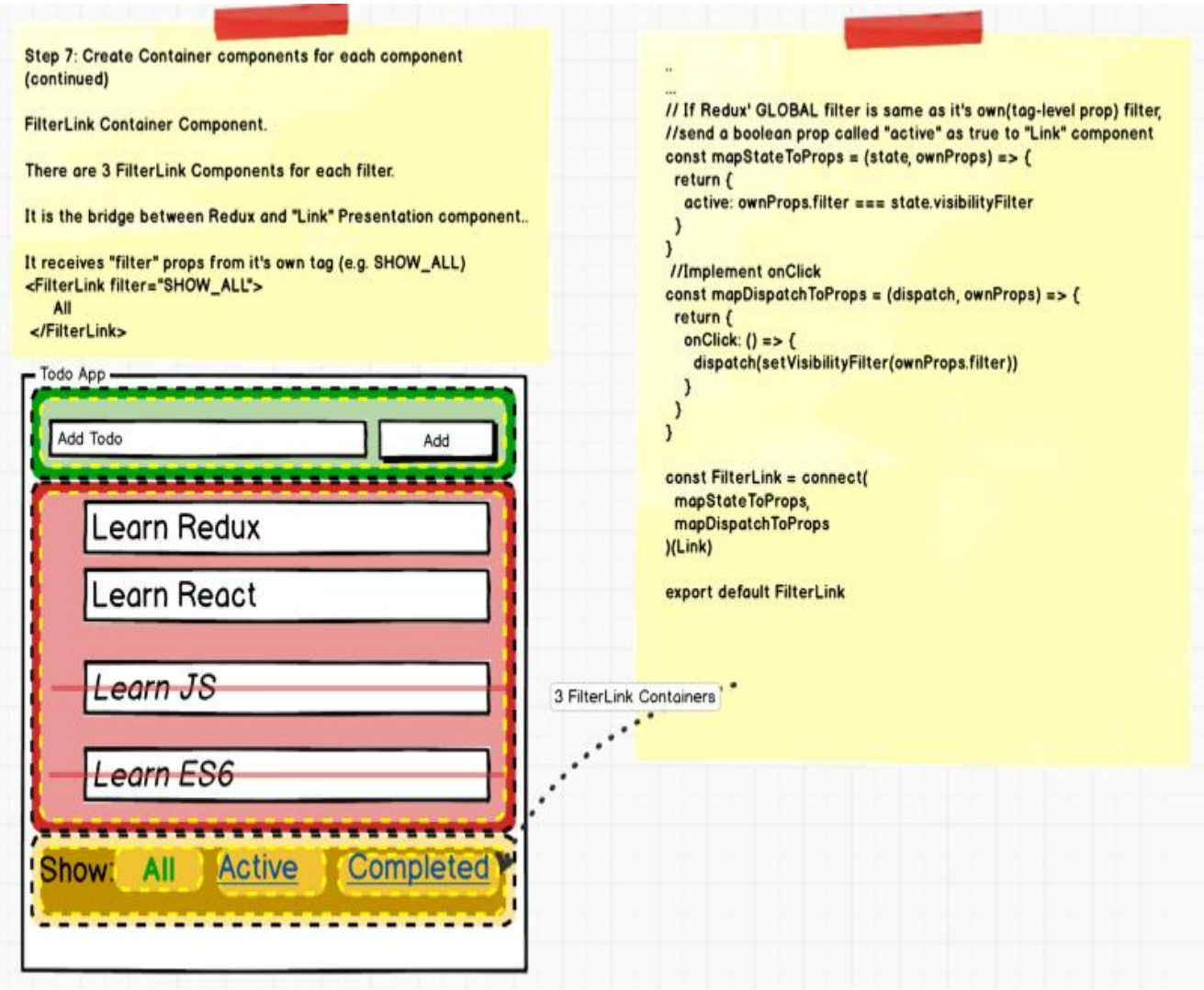
7.1 Create Container Component — AddTodo



7.2 Create Container Component—TodoList Container



7.3 Create Container Component—Filter Container



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"
)
```