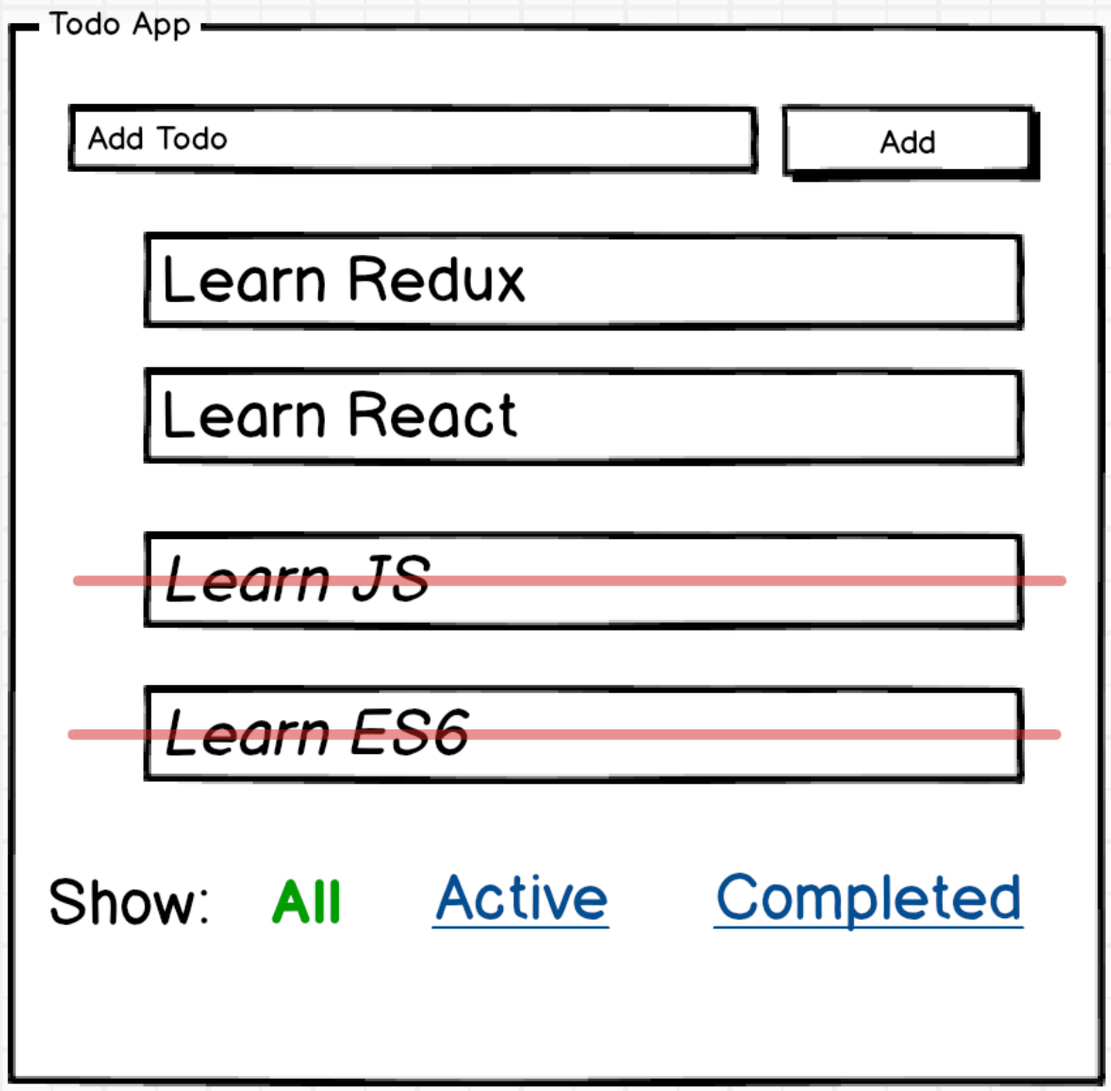
React

Roli Butron

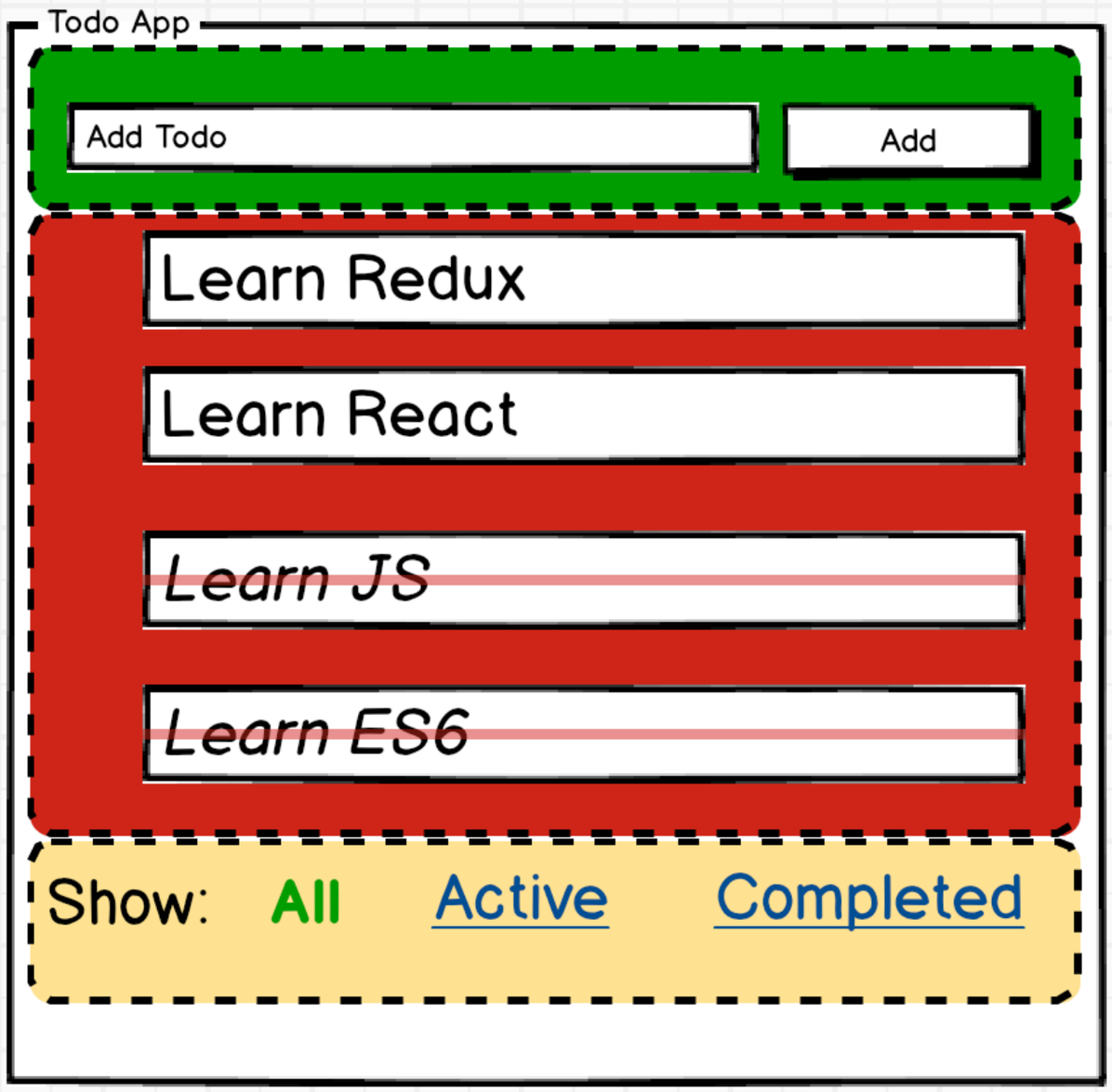
Write A Detailed Mock of the Screen



Step 1: Write Mock of each screen

HOW:
1. Be as detailed as possible
2. Make sure to represent all the data and visual effect (like strikethrough of a Todo item)

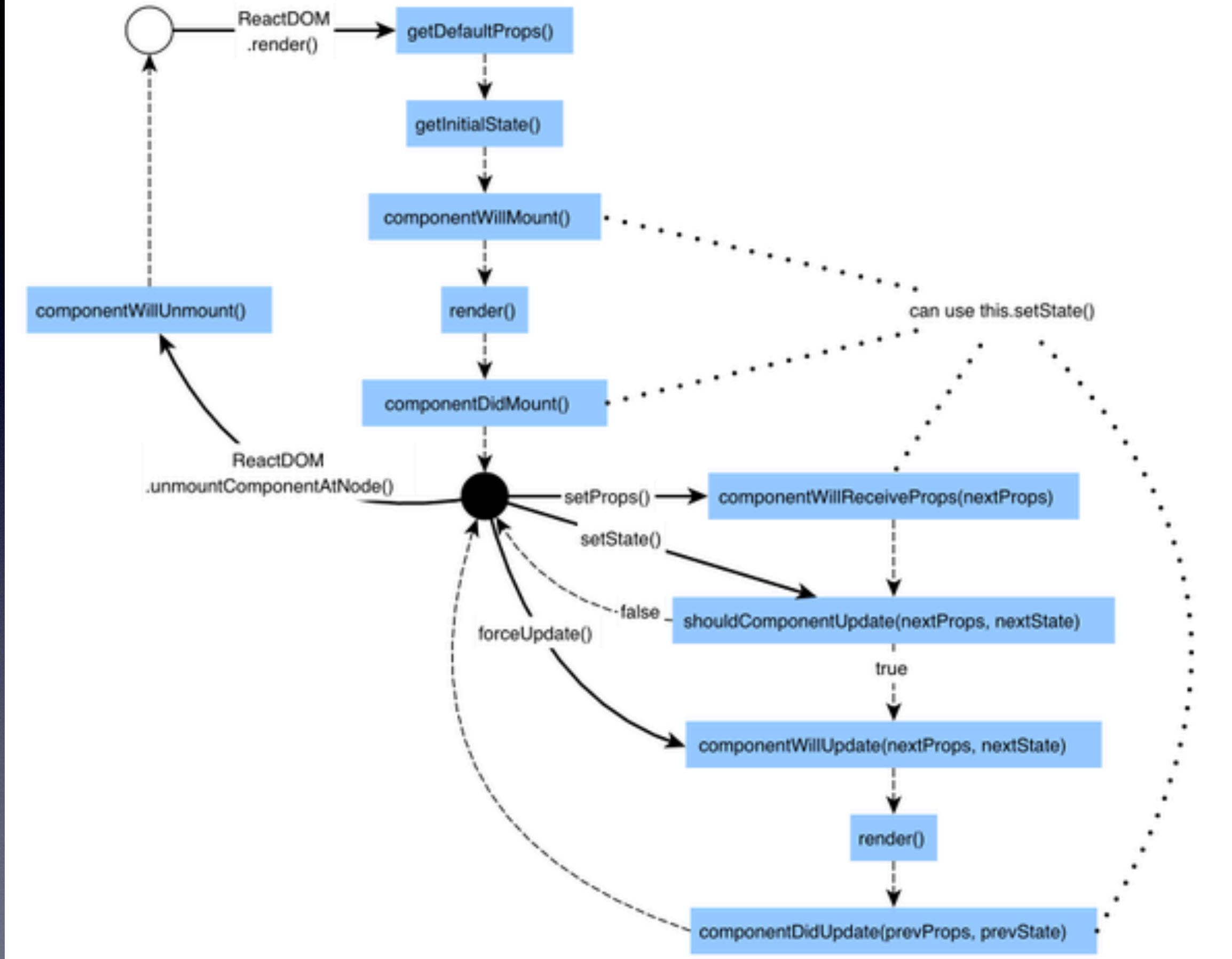
Divide the App into Components



Step 2: Divide the app into components

How: Divide them based on their overall purpose. This is usualla 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)

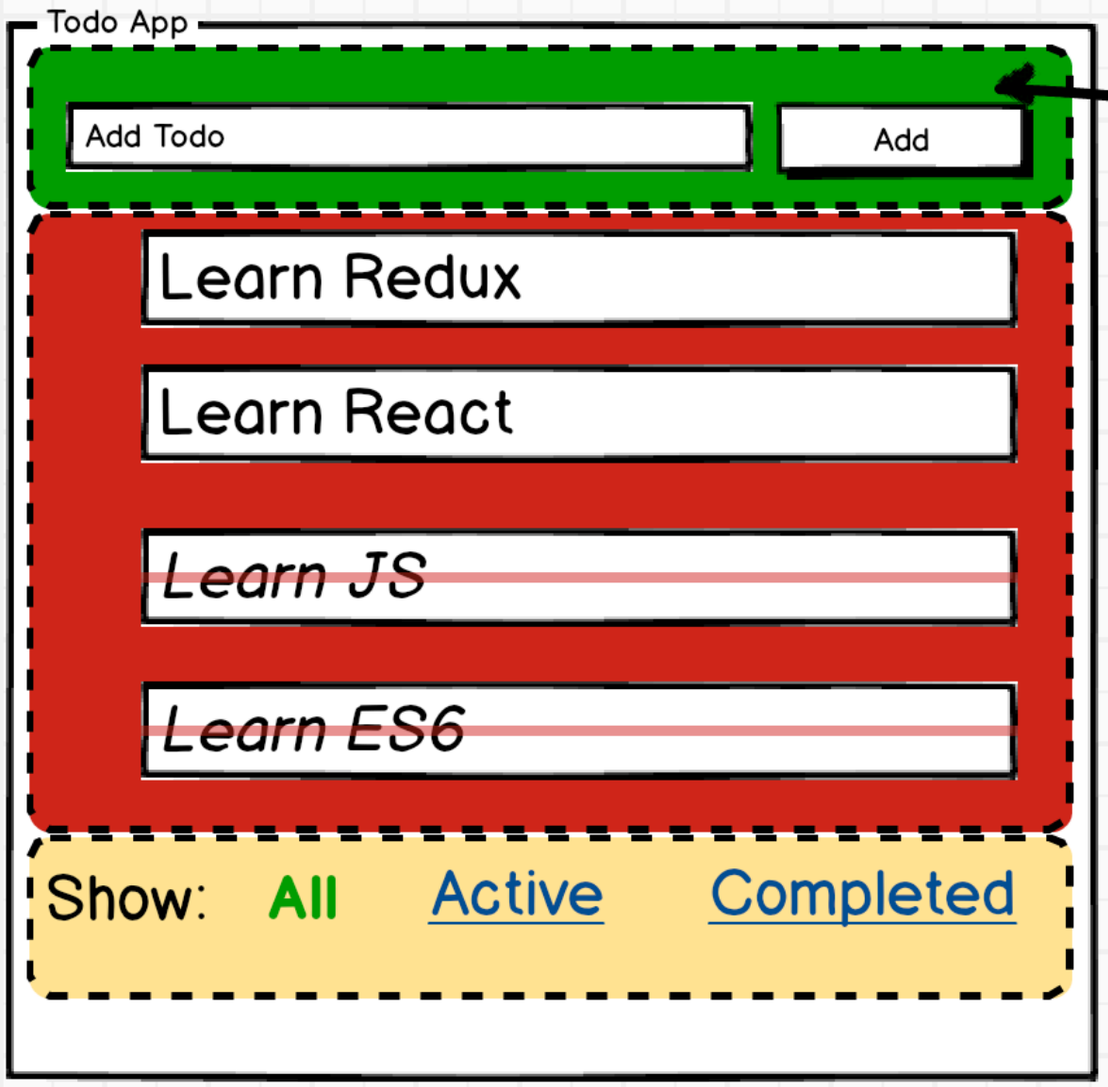

ReactDOM
.render()
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
can use this.setState()
componentWillUnmount()
ReactDOM
.unmountComponentAtNode()
setProps()
componentWillReceiveProps(nextProps)
setState()
false
shouldComponentUpdate(nextProps, nextState)
forceUpdate()
true
componentWillUpdate(nextProps, nextState)
render()
componentDidUpdate(prevProps, prevState)

```
import React, {Component} from 'react'

const getCurrentPath = () => {
  const path = document.location.pathname
  return path.substring(path.lastIndexOf('/'))
}

export class Router extends Component {
  state = {
    route: getCurrentPath()
  }

  handleLinkClick = (route) => {
    this.setState({route})
    window.history.pushState(null, '', route)
  }

  static childContextTypes = {
    route: React.PropTypes.string,
    linkHandler: React.PropTypes.func
  }

  getChildContext() {
    return {
      route: this.state.route,
      linkHandler: this.handleLinkClick
    }
  }

  componentDidMount() {
    window.onpopstate = () => {
      this.setState({route: getCurrentPath()})
    }
  }

  render () {
    return <div>{this.props.children}</div>
  }
}
```

```
import React, {Component} from 'react'

export class Link extends Component {
  static contextTypes = {
    route: React.PropTypes.string,
    linkHandler: React.PropTypes.func
  }

  handleClick = (evt) => {
    evt.preventDefault()
    this.context.linkHandler(this.props.to)
  }

  render() {
    const activeClass = this.context.route === this.props.to ? 'active' : ''
    return <a href="#"
              className={activeClass}
              onClick={this.handleClick}>{this.props.children}</a>
  }
}

Link.propTypes = {
  to: React.PropTypes.string.isRequired
}
```

AddTodo Component - State and Actions

Step 3: Make a detailed List of state and actions (continued)



AddTodo component

State:

1. **N/A (it's a simple field and a button and their display doesn't change based on any data)**

Actions (Events)

1. **AddTodo component allows us to create new Todo items by listening to DOM events and by reading data from the input field. This event is mapped into a JSON object called Action.**

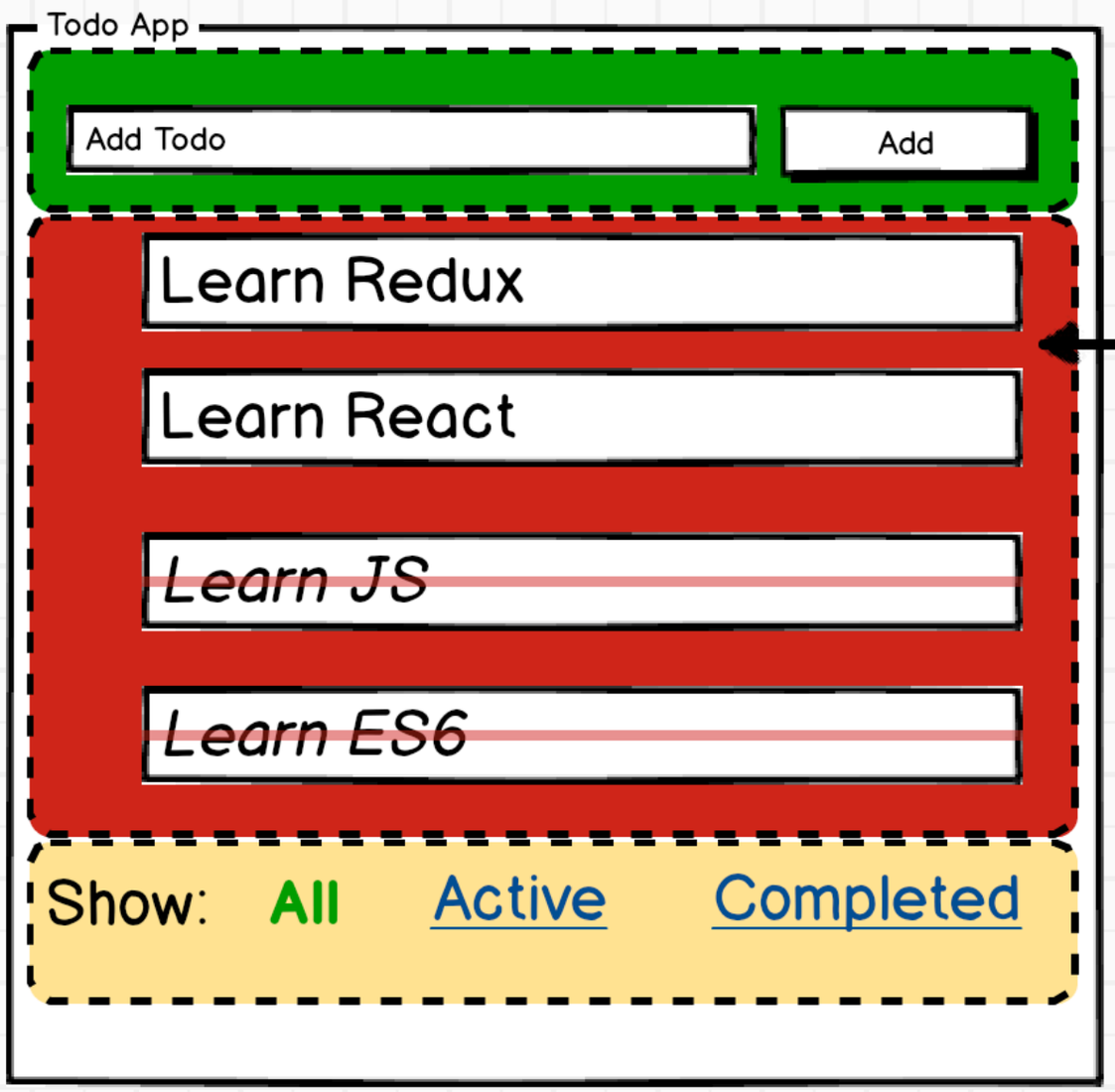
In our case we can describe our AddTodo action by creating a JSON object with a type: "ADD_TODO" and adding the data like below:

```
{
 type: "ADD_TODO"
 payload: {data: "Learn Redux", id: 1", "completed: false}
}
```

TodoList Component - State And Actions

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.

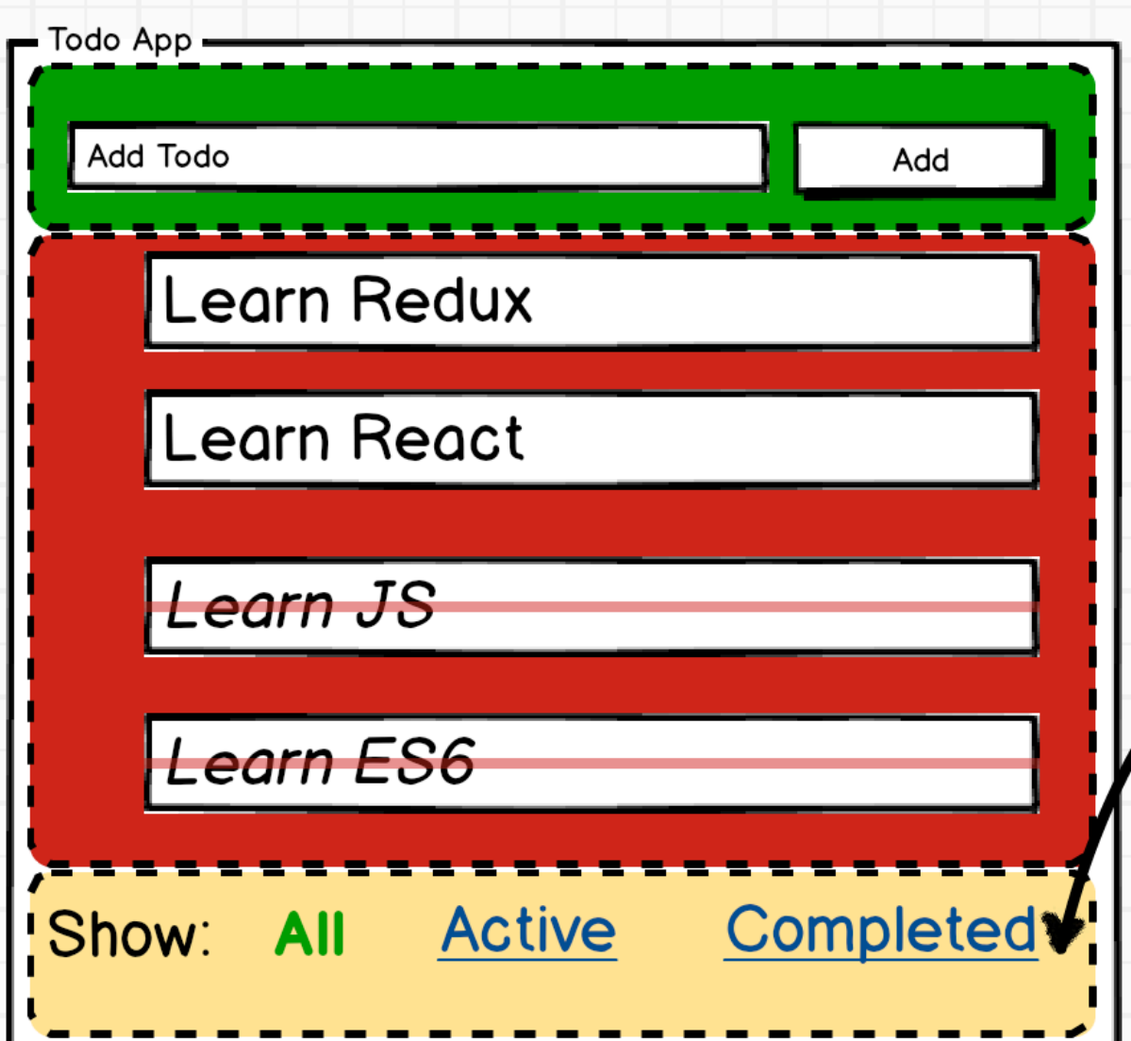
Again we need to describe the ToggleTodo action JSON object by giving it a name and payload. In this case it may look like below:

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

Filter Component - State And Actions

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 showas 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"}
}
```

Create Action Creators for Each Action

```
//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
 }
}
```

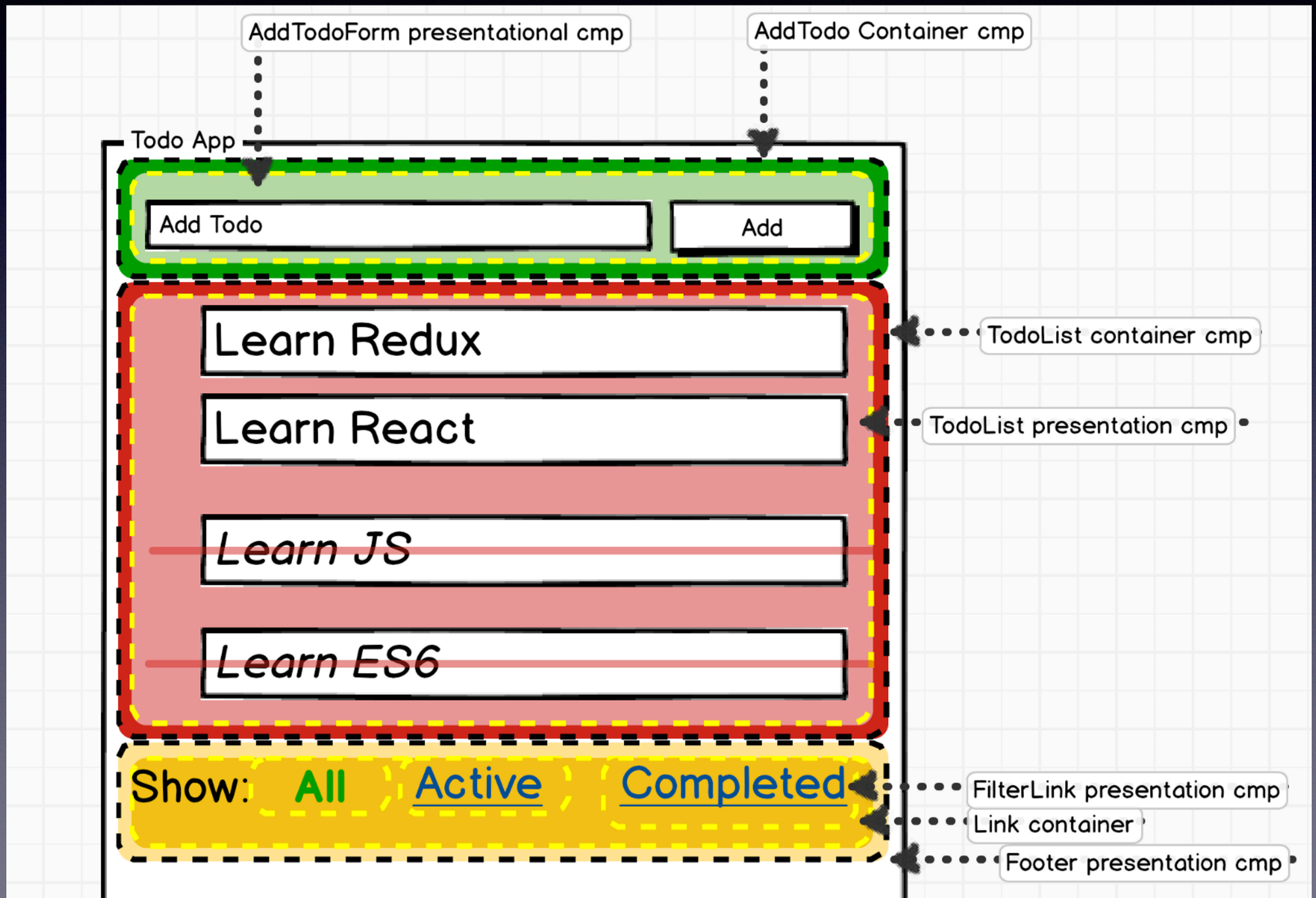
Reducers

```
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}];
 }
```

Write Reducers For Each Action

```
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
    }
}
```

Presentational and Container Components



Implement AddTodoForm Presentational Component

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

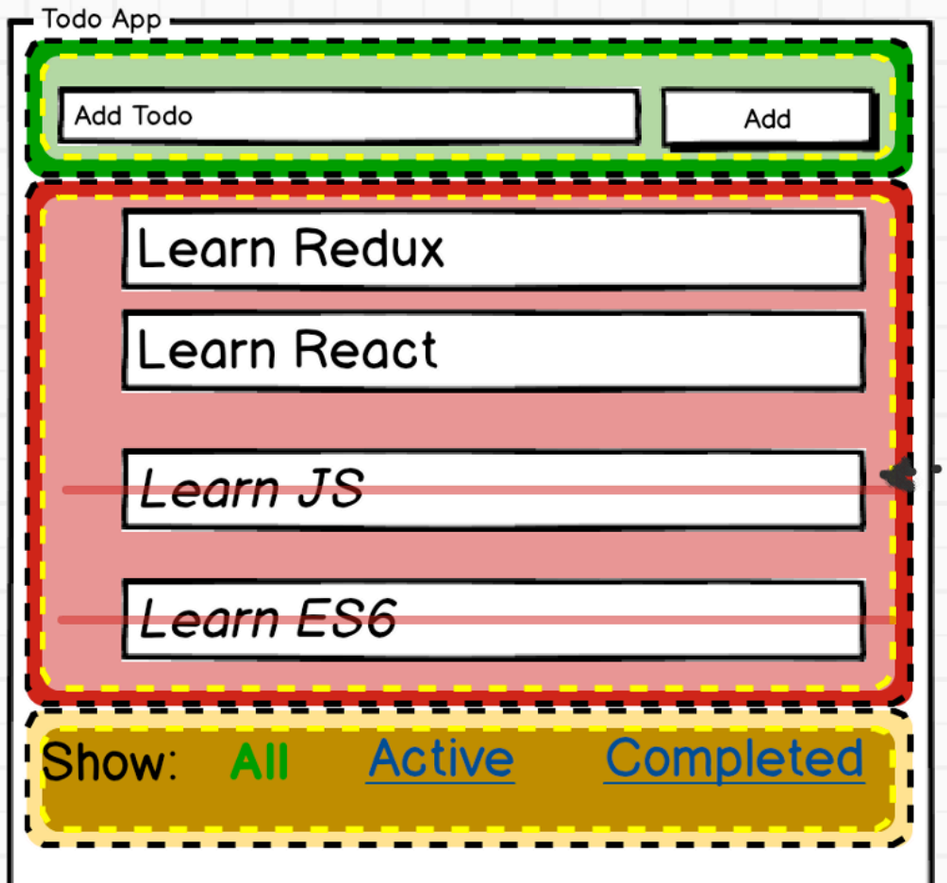
AddTodoForm Component:
This component renders a field and button. It receives onSubmit callback function (from parent Container component) and when the user submits new Todo, it sends calls that onSubmit function w/ the Todo text.


```
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
```

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.



TodoList Cmp

```
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
```

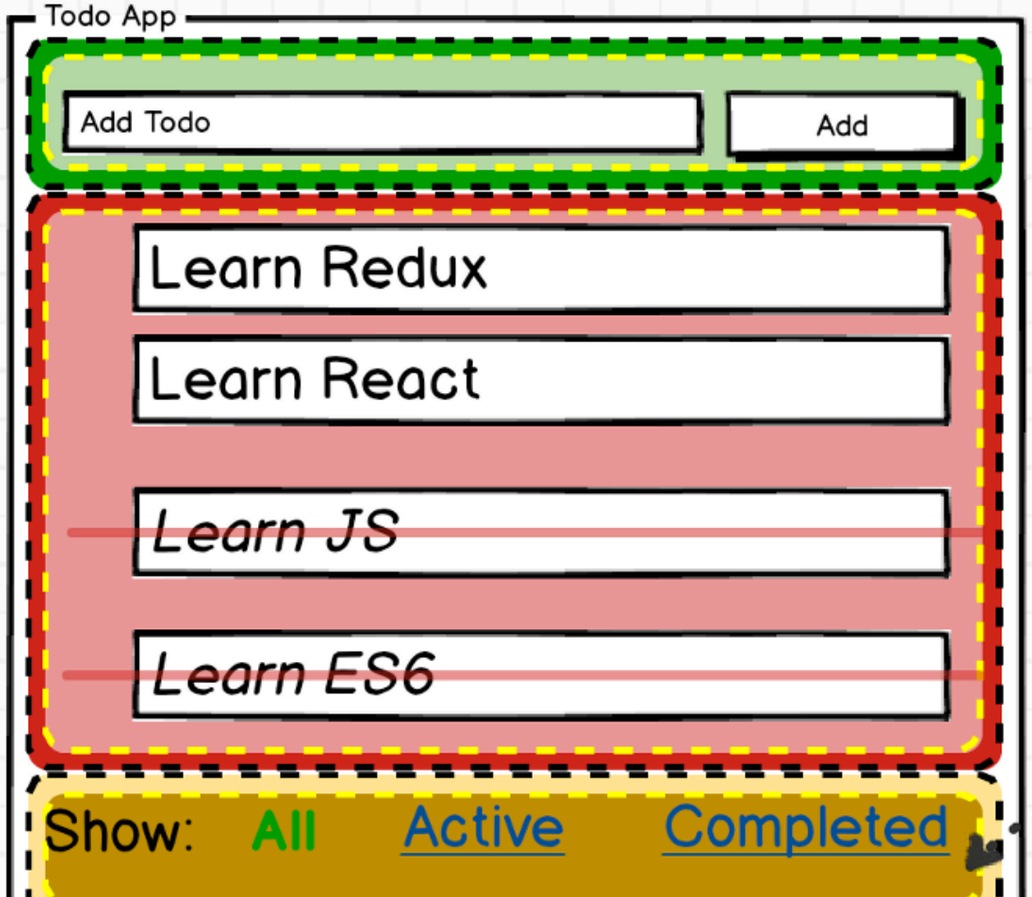
Finally Bring Them All Together

Finally Bring Them All Together

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
```

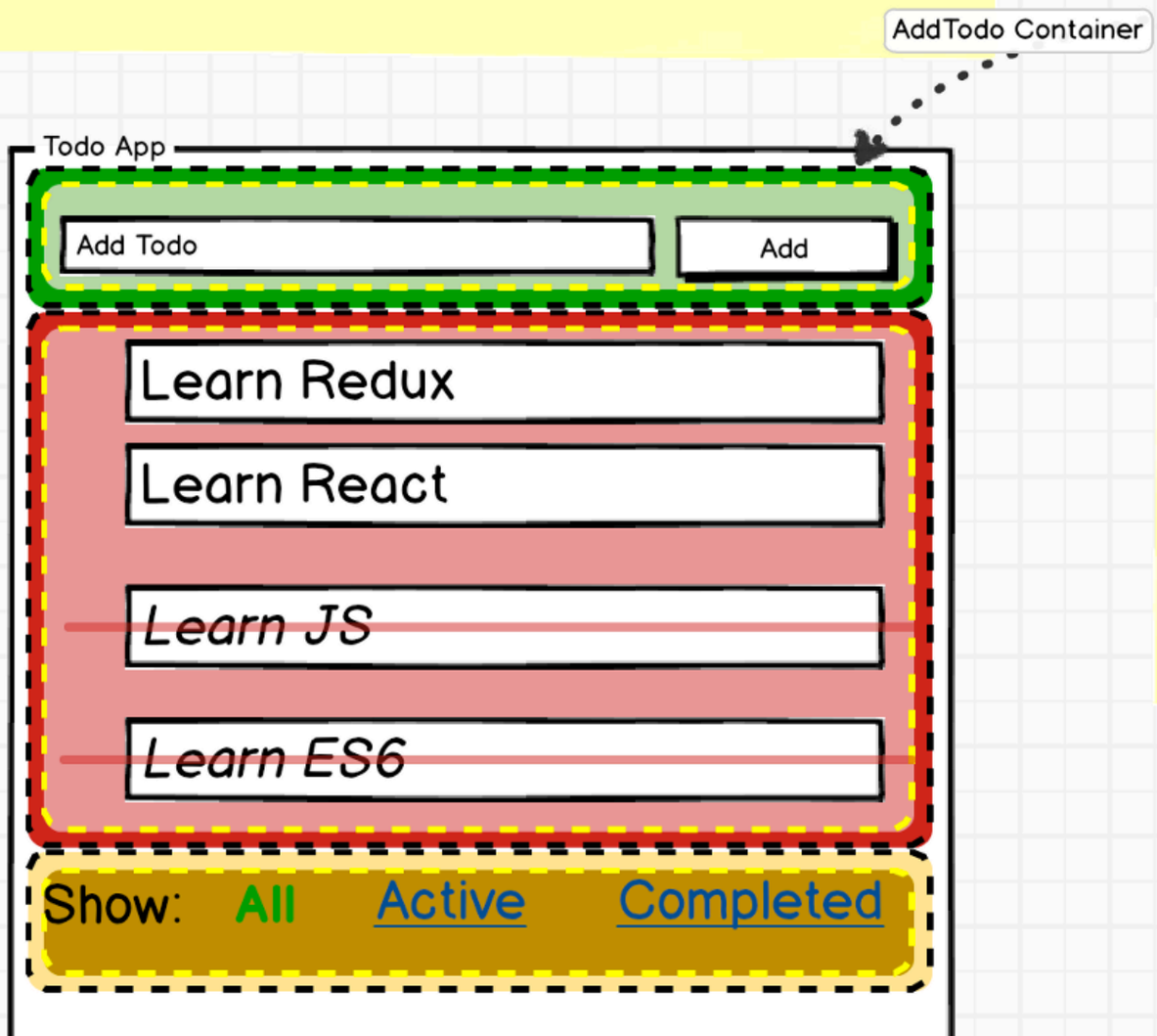
Create Container Component - AddTodo

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.




```
//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
```

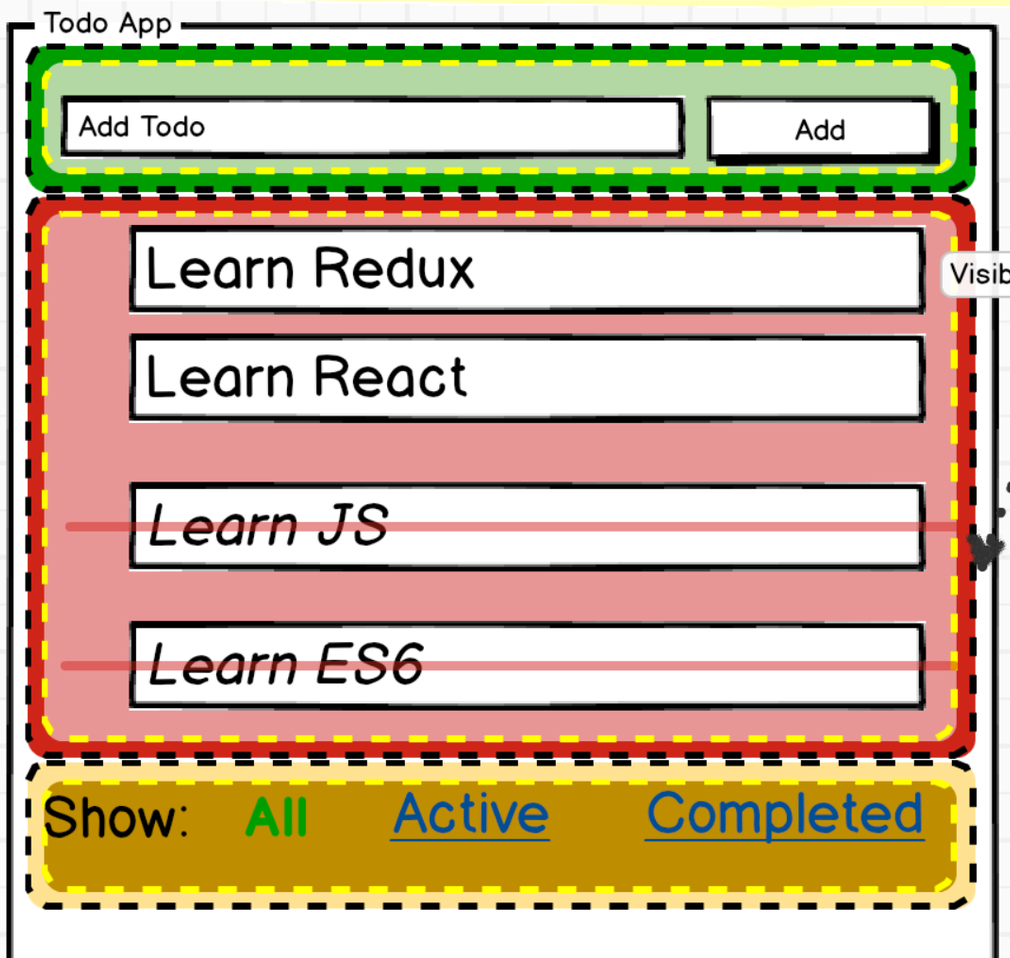
Create Container Component - TodoList

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
const VisibleTodoList = connect(
 mapStateToProps,
 mapDispatchToProps
)(TodoList)

export default VisibleTodoList
```

Create Container Component - Filter

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

FilterLink Container Component.

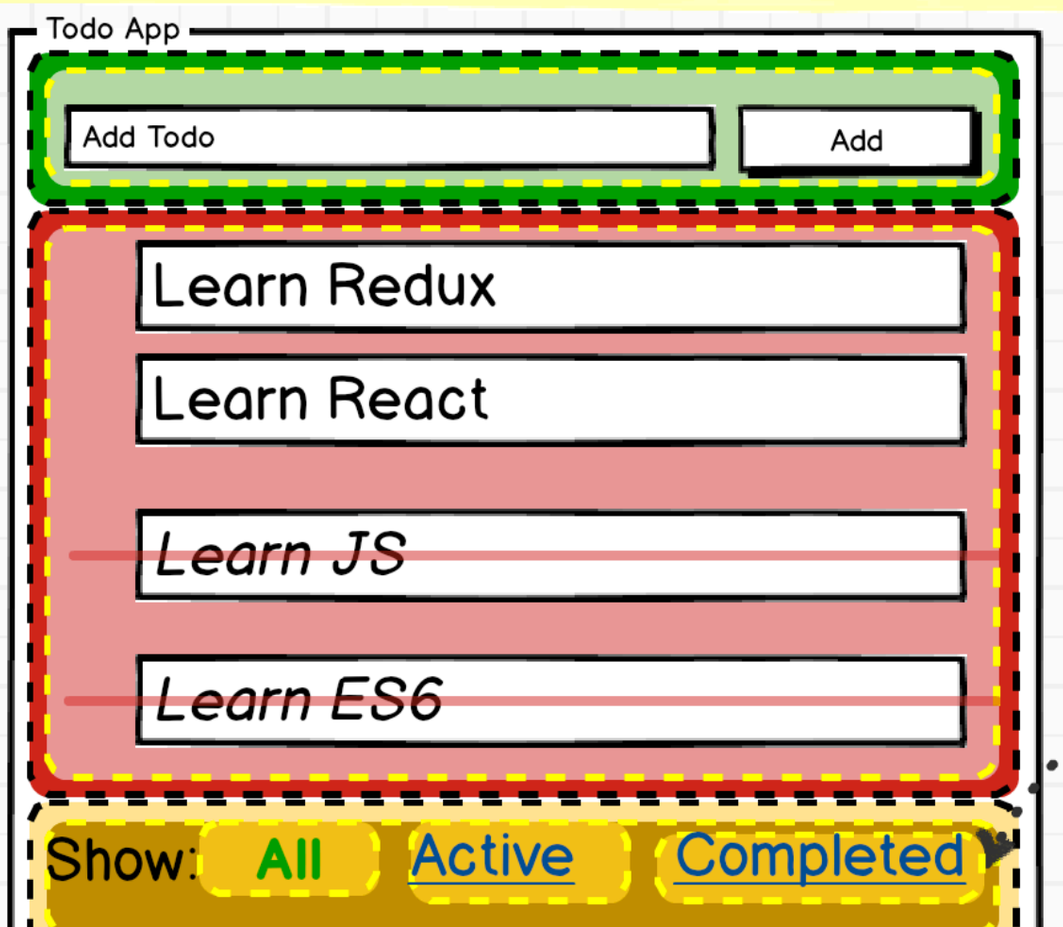
There are 3 FilterLink Components for each filter.

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
```



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"
)
```