REDUX WIDGET LIST EDITOR

Dr. Jose Annunziato

Let's first implement a widget component

```
const Widget
    = () => (
 <li> Widget </li>
)
```

Render the component

```
import React from 'react'
import ReactDOM from 'react-dom'

ReactDOM.render(
    <Widget/>
     document.getElementById('root')
);
```

Let's parameterize the widget with a property

```
const Widget
    = ({ widget }) => (
 <li> {widget.text} </li>
)
```

Pass a widget object

```
import React from 'react'
import ReactDOM from 'react-dom'

ReactDOM.render(
    <Widget widget={{text: 'Hello'}}/>
     document.getElementById('root')
);
```

WIDGET LIST

Dr. Jose Annunziato

Render a list widgets

```
const WidgetList = ({ widgets }) => (
 <ul>
   {widgets.map(widget =>
      <Widget key={widget.id}
              widget={widget}/>)}
 </ul>)
```

Pass an array of widgets

```
ReactDOM.render(
    <WidgetList widgets=
        {[{text: 'Heading'}, {text: 'List'}]}/>
    document.getElementById('root')
);
```

Let's wrap the App

```
const App = () => (
    <WidgetList widgets=
        {[{text: 'Heading'}, {text: 'List'}]}/>
)
ReactDOM.render(
    <App/>
     document.getElementById('root'));
```

Adding Redux

Dr. Jose Annunziato

Reducer Generates Next State

```
const widgets = (state = [], action) => {
    switch (action.type) {
        case 'ADD_WIDGET':
            return [...state,
                {id: state.widgets.length+1,
                 text: action.text}]
    default: return state
}}
```

Map redux state to properties

```
const mapStateToProps = state => ({
 widgets: state.widgets
})
const WidgetListContainer =
connect(mapStateToProps)(WidgetL
ist)
```

The App is just the combination

```
const App = () => (
 <div>
   <WidgetListContainer />
 </div>
)
```

Create the store and provide it to the app

```
const rootReducer = combineReducers({ widgets })
const store = createStore(rootReducer)
export default class WidgetsComponent
          extends Component
{   render() {
      return (   <Provider store={store}>
                   <App />
                 </Provider>)}}
```

Adding a Form

Dr. Jose Annunziato

Add widget form

```
let nextWidgetId = 0
const AddWidgetComponent =
    ({ dispatch }) => {
    let input
    return (<div>
      <input ref={node => input = node} />
```

Add widget form

```
  <button type="submit" onClick={e => {
      dispatch({ type: 'ADD_WIDGET',
          id: nextWidgetId++,
          text: input.value})}}>Add Widget
  </button></div>)}
const AddWidget = connect()
(AddWidgetComponent)
```

The App is just the combination

```
const App = () => (
 <div>
   <WidgetList />
   <AddWidget  />
 </div>
)
```

Delete Widget

Dr. Jose Annunziato

Add delete widget button

```
const WidgetComponent
      = ({ widget, deleteWidget }) => (
   <li>{widget.text}
      <button onClick={e => {
         dispatch({type: 'DELETE_WIDGET',
                   id: widget.id})}}>
         Delete</button></li>)
```

Connect widget to reducer

```
const Widget = connect()(WidgetComponent)
const WidgetListComponent
        = ({ widgets }) => (
 <ul>
  {widgets.map(widget =>
    <Widget key={widget.id}
            widget={widget}/>
  )}</ul>)
```

Add DELETE_WIDGET to reducer

```
const widgets = (state = [], action) => {
 switch (action.type) {
   case 'ADD_WIDGET': ...
   case 'DELETE_WIDGET':
     return state.filter(widget => widget.id != action.id)
   default: return state
}}
```

Action creators

Dr. Jose Annunziato

Create action objects in a separate function

```
<button onClick={() => {
  dispatch({type: 'DELETE_WIDGET',
                  id: widget.id})
  dispatch(deleteWidget(widget.id))
}}>Delete</button>
```

Action creators abstract their payload

```
const deleteWidget = id => {
 return {
   type: 'DELETE_WIDGET', id: id
 }
}
```

Do the same for the AddWidget component

```
<button type="submit" onClick={e => {
   dispatch({
      type: 'ADD_WIDGET',
      id: nextWidgetId++,
      text: input.value
   })
   dispatch(addWidget(input.value))
}}>
Add Widget</button>
```

Action creators are portable

```
const addWidget = text => {
 return {
   type: 'ADD_WIDGET',
   id: nextWidgetId++,
   text: text
}}
```

Reordering widgets

Dr. Jose Annunziato

Action creators are portable

```
Array.prototype.move
    = function (from, to) {
    this.splice(to, 0, this.splice(from, 1)[0]);
};
```

```
const moveUp = widget => {
 return {
   type: 'MOVE_UP', widget: widget
 }
}
```

```
<button onClick={() => {
  dispatch(moveUp(widget))
}}>^</button>
```

```
case 'MOVE_UP':
 let index = state.indexOf(action.widget);
 state.move(index, index - 1);
 return state.splice(0);
```

Different types of widgets

Dr. Jose Annunziato

Add a dropdown on each widget to select type

```
const WidgetComponent = ({ widget, dispatch }) => {
<select ref={node => select = node}
        value={widget.widgetType}
        onChange={e => {
 dispatch(setWidgetType(widget.id, select.value))
}}>
 <option>Heading</option><option>Paragraph</option>
 <option>HTML</option><option>Link</option>
 <option>iFrame</option>
</select>)}
```

OnChange, dispatch the widget's type

```
const setWidgetType = (id, widgetType) => {
 return {
   type: 'SET_WIDGET_TYPE',
   widgetType: widgetType, id: id
 }
}
```

Set the widget's type in the reducer

```
const widgets = (state = [], action) => {
case 'SET_WIDGET_TYPE':
 let newState = JSON.parse(JSON.stringify(state))
 index = newState.findIndex(function (widget) {
     return widget.id === action.id})
 newState[index].widgetType = action.widgetType
 return newState
}
```

Adding a toggle editing button

Dr. Jose Annunziato

Add a checkbox to the WidgetComponent

```
<label>
 <input ref={node => editing = node}
    type="checkbox"
    onChange={e => {
      dispatch(toggleEditing
            (widget.id, editing.checked))}}
    checked={widget.editing}/> Editing
</label>
```

Add a TOGGLE_EDITING action creator

```
const toggleEditing = (id, checked) => {
 return {
   type: 'TOGGLE_EDITING',
   id: id,
   editing: checked
}}
```

In the reducer, handle the new toggle dispatch

```
case 'TOGGLE_EDITING':
 newState = JSON.parse(JSON.stringify(state))
 index = newState.findIndex(
   function (widget) {
     return widget.id === action.id
   })
 newState[index].editing = action.editing
 console.log(newState)
 return newState
```

Hiding/showing content based on state

Dr. Jose Annunziato

Add a div that will hide/show based on checkbox

```
<div style=
        {{display: widget.editing
                    ? 'block': 'none'}}>
  Widget Editor
</div>
```

Show/Hide editors based on widget type

```
<div style={{display: widget.editing ? 'block': 'none'}}>
  <div style={{display: widget.widgetType
                    ==='Heading' ? 'block': 'none'}}>
    Heading
  </div>
  <div style={{display: widget.widgetType
                    ==='Paragraph' ? 'block': 'none'}}>
    Paragraph
  </div></div>
```

```
const RawTextWidgetComponent = () => {
 return (
   <h1>Raw Text Widget</h1>
 )
}
```

```
<div style={{display: widget.editing ? 'block': 'none'}}>
  <div style={{display: widget.widgetType
                    ==='Heading' ? 'block': 'none'}}>
    Heading
  </div>
   {widget.widgetType==='Raw Text' &&
      <RawTextWidgetComponent widget={widget}/>}
</div>
```

Raw Text Widget

Dr. Jose Annunziato

RawTextWidgetComponent

```
<textarea ref={node => textarea = node}
          value={widget.rawtext}
          onChange={e => {
            dispatch(setTextWidget(widget.id,
                                   textarea.value))
            preview.innerHTML = textarea.value
          }}></textarea>
<p ref={node => preview = node}></p>
```

```
const setTextWidget = (id, text) => (
   {type: 'SET_TEXT_WIDGET', id: id, text: text})
const RawTextWidget = connect()
(RawTextWidgetComponent)
```

```
case 'SET_TEXT_WIDGET':
  newState = JSON.parse(JSON.stringify(state))
  index = newState.findIndex(
    function (widget) { return widget.id === action.id })
  newState[index].rawtext = action.text
  console.log(newState)
  return newState
```