

3 StephenGrider Modeling Application State

book detail fork

<https://codesandbox.io/s/486zn8n5j9>

<https://github.com/StephenGrider/BookList>

Redux predictable state container for JavaScript applications

<https://redux.js.org/>

On top of redux you have

webpack

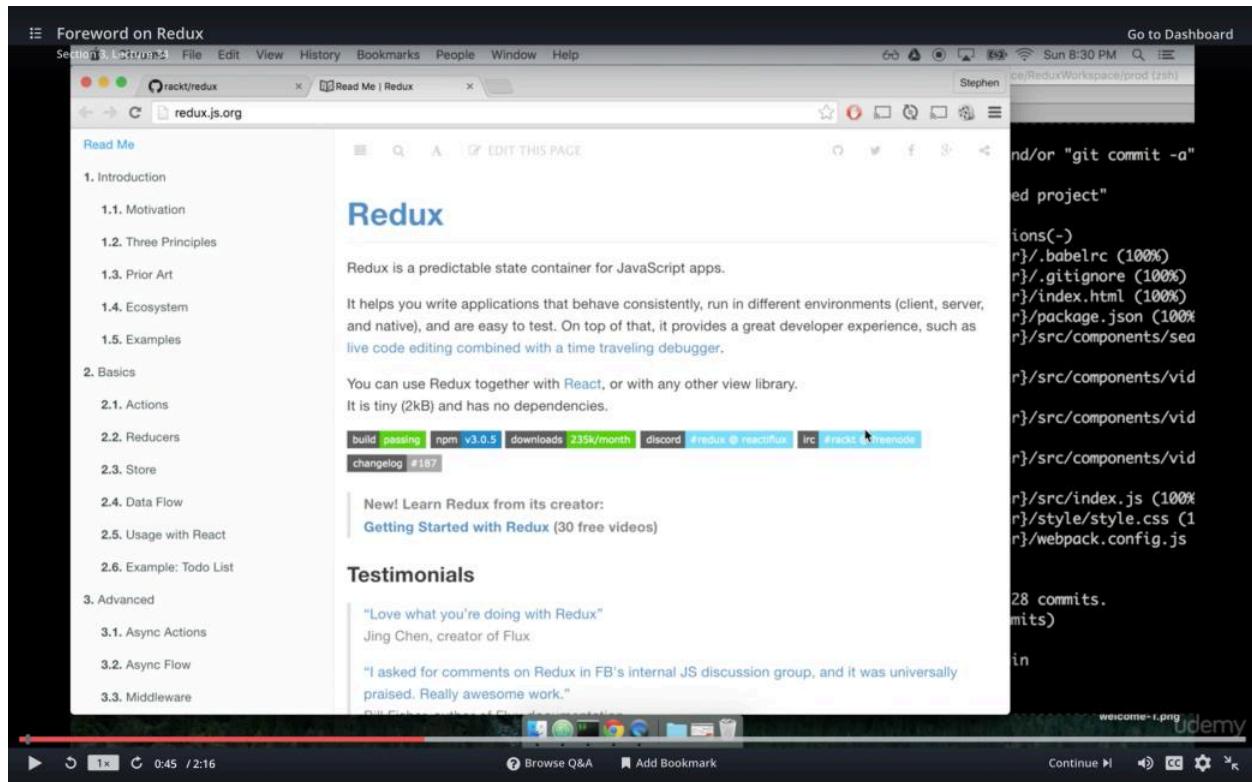
redux router

react router

redux promise

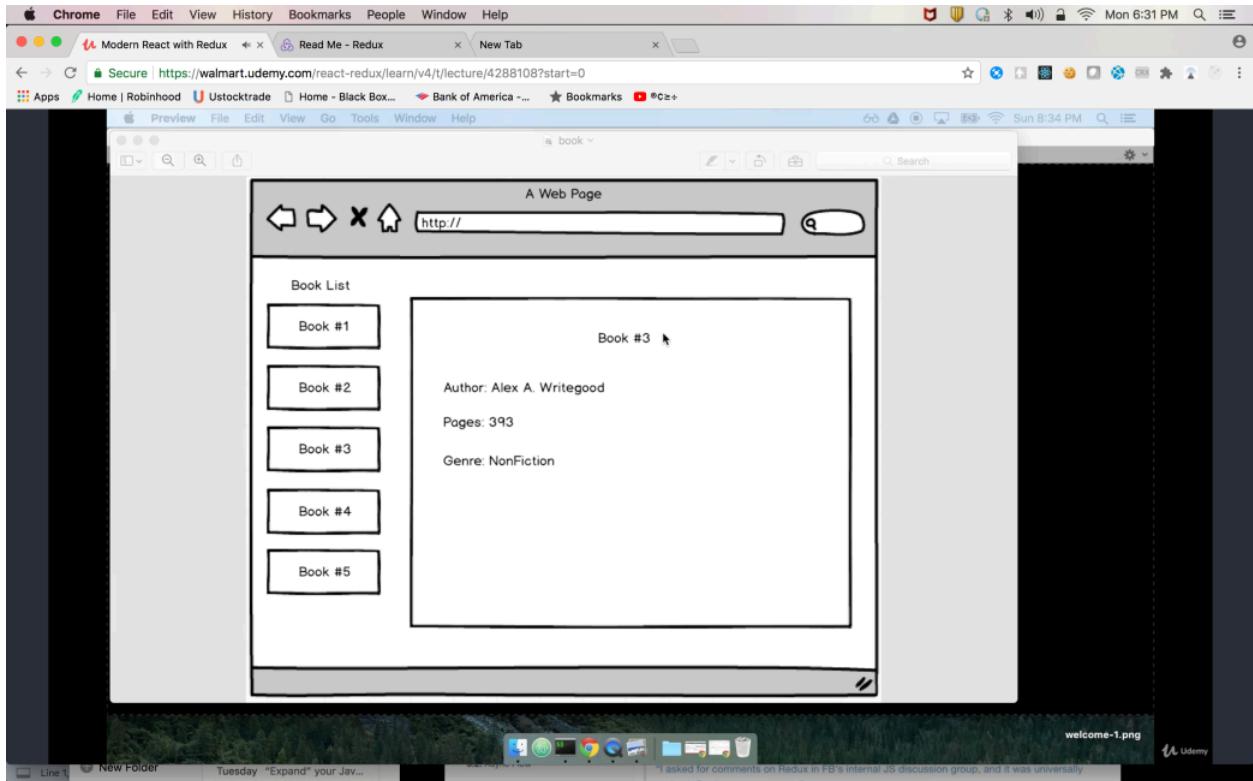
redux thunk

hot reloading



show list of books on left hand side

if user clicks on any book it shows details

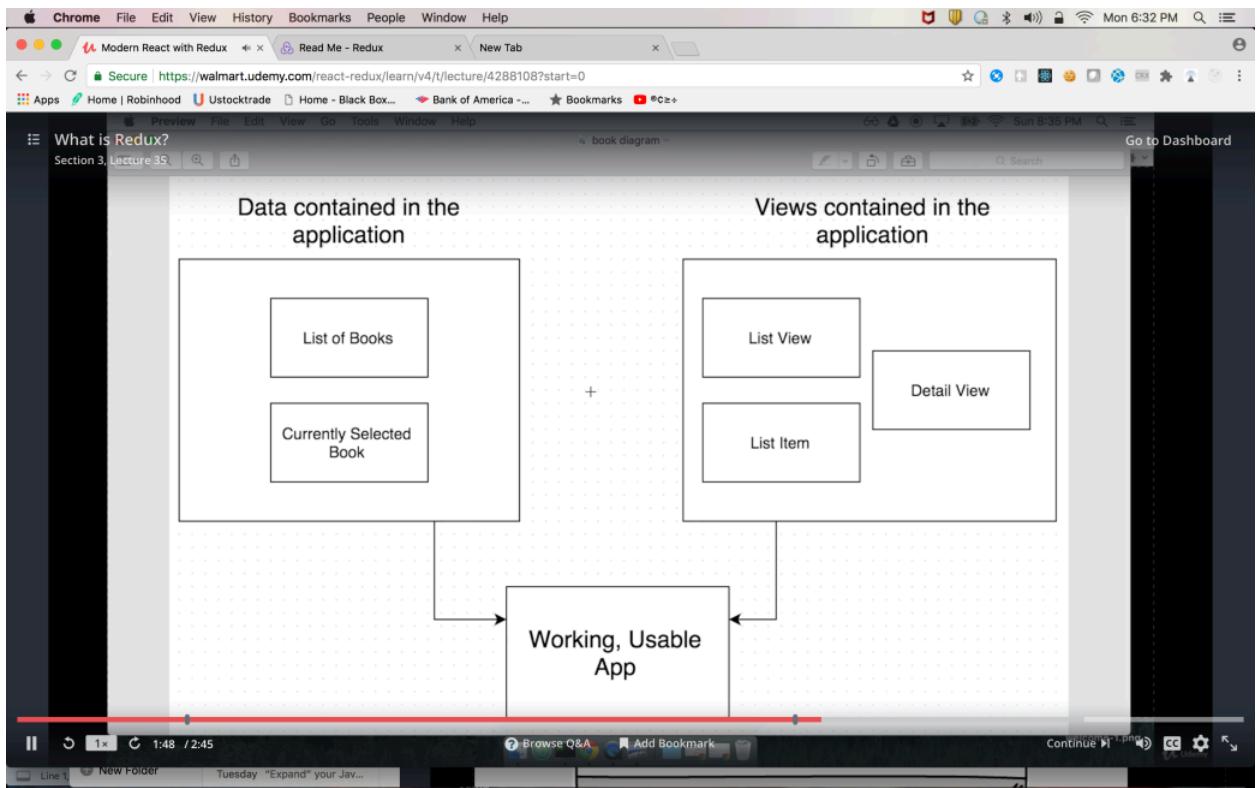


first separate data and view

Redux is data container (State Container) in application box
React is view container in application box]

Redux centralized all the application data in single object, which referred to state

Redux maintains application level state
React view level state



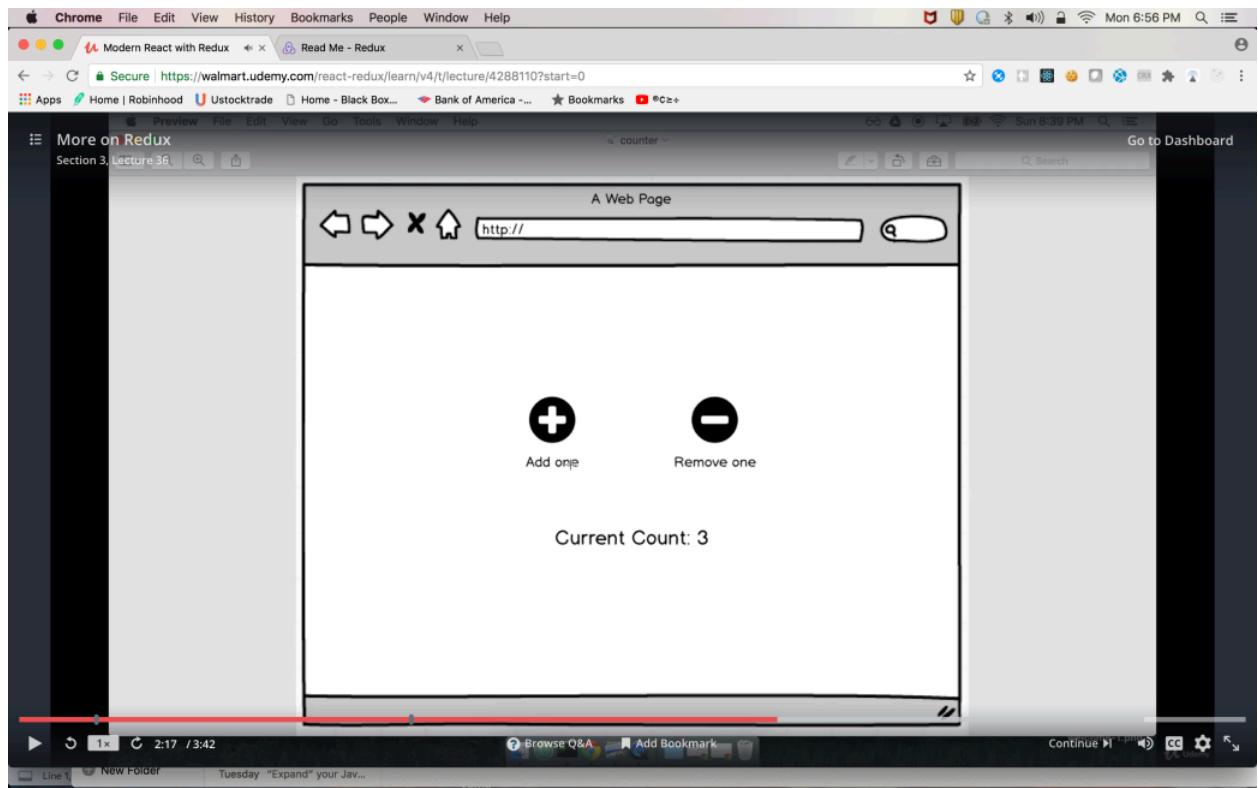
In other javascript lib they have separate collections of data.
backbone has collections, flux has stores

Example of counter application

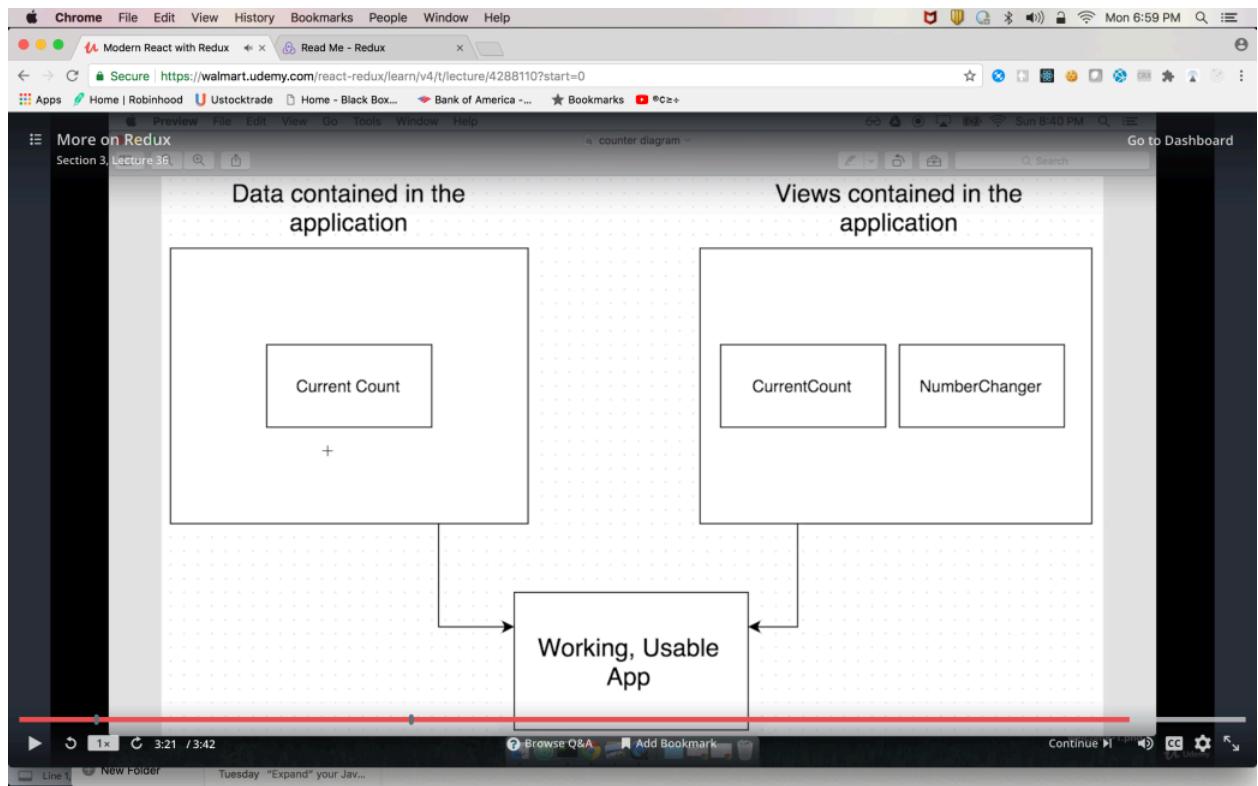
at bottom counter start at 0

when user press + plus button number goes up by 1

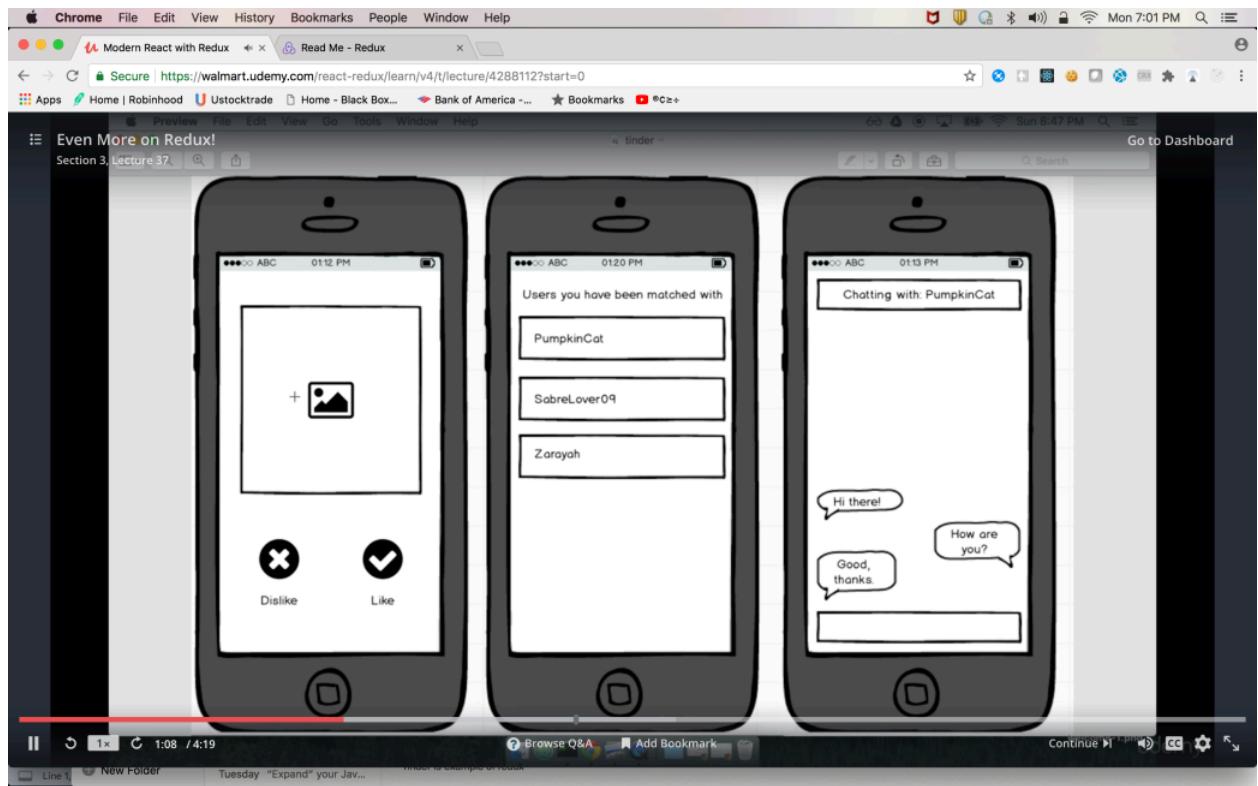
when user press - minus button number goes down by 1

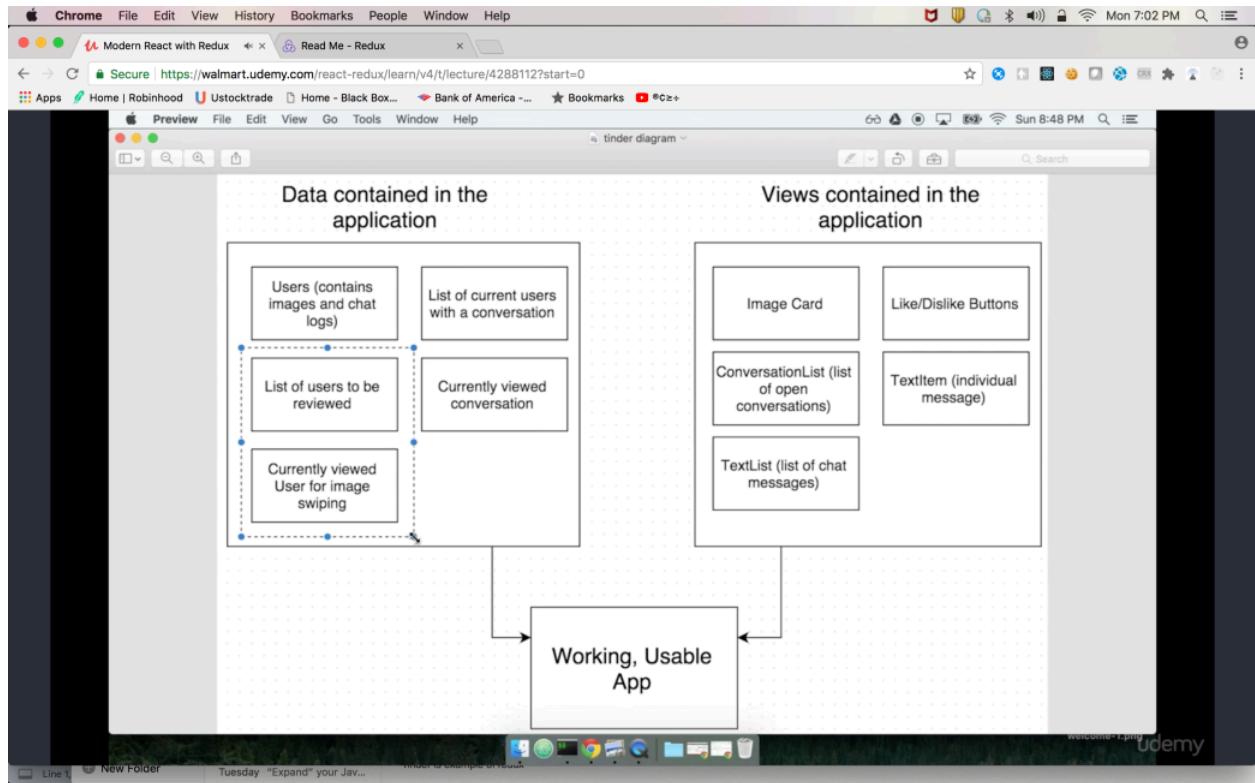


React will be showing 2 components
Redux will maintain state of counter



Tinder is example of redux





The screenshot shows a Chrome browser window with the following details:

- Address Bar:** https://walmart.udemy.com/react-redux/learn/v4/t/lecture/6052280?start=0
- Page Title:** Putting Redux to Practice
- Section:** Section 4, Lecture 38
- Content:** A text block explaining that after looking at different app layouts, the user will start working on their next app by cloning or downloading the Redux Simple Starter package from GitHub, and running `npm install` in the project directory.
- Buttons:** "Go to Dashboard" and "Continue >"

```
import React from "react";
import ReactDOM from "react-dom";

import { Provider } from "react-redux";
import { createStore, applyMiddleware } from "redux";

import reducers from "./reducers";

import App from "./components/app";

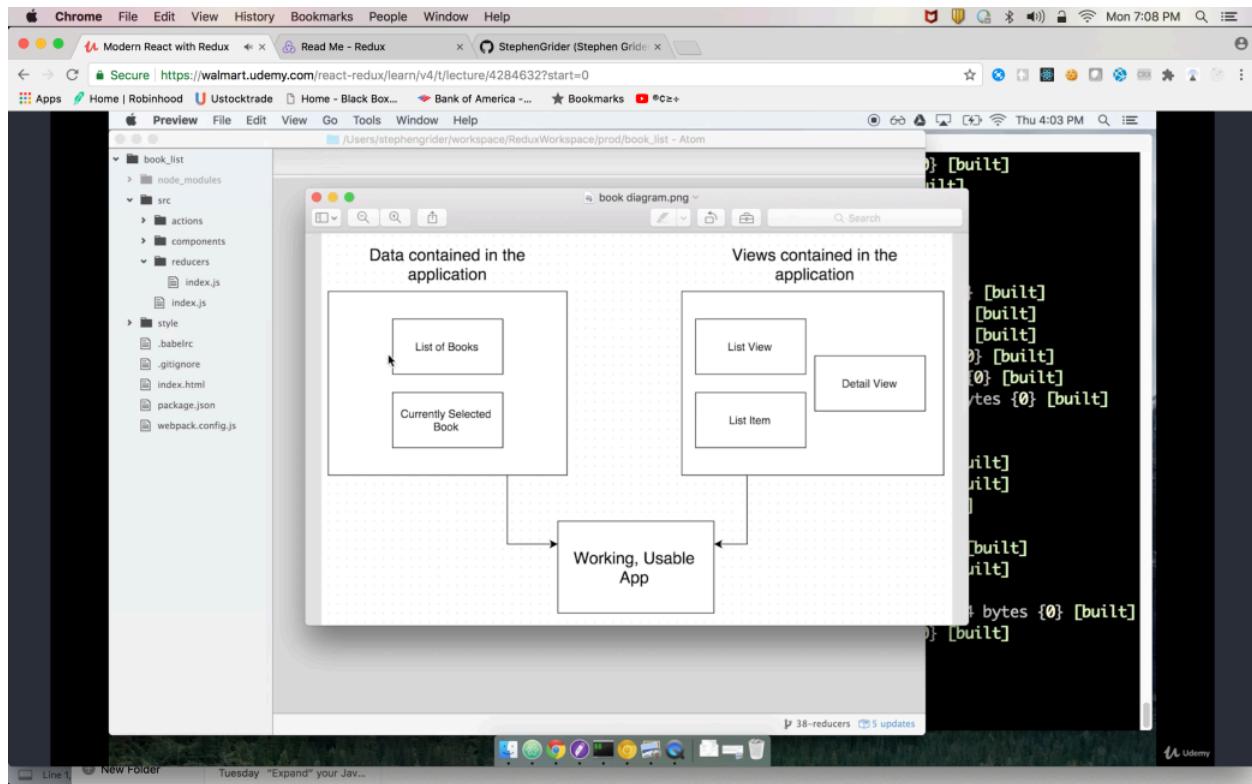
import "./styles/base.css";

const createStoreWithMiddleware = applyMiddleware()(createStore)(reducers);

ReactDOM.render(
<Provider store={createStoreWithMiddleware}>
<App />
</Provider>,
document.getElementById("root")
);
```

Reducer is function that returns the piece of applications state

since our application has 2 pieces of state
we will need 2 reducers in below example



Application state is a pure javascript object

it has 2 keys
2 pieces of state

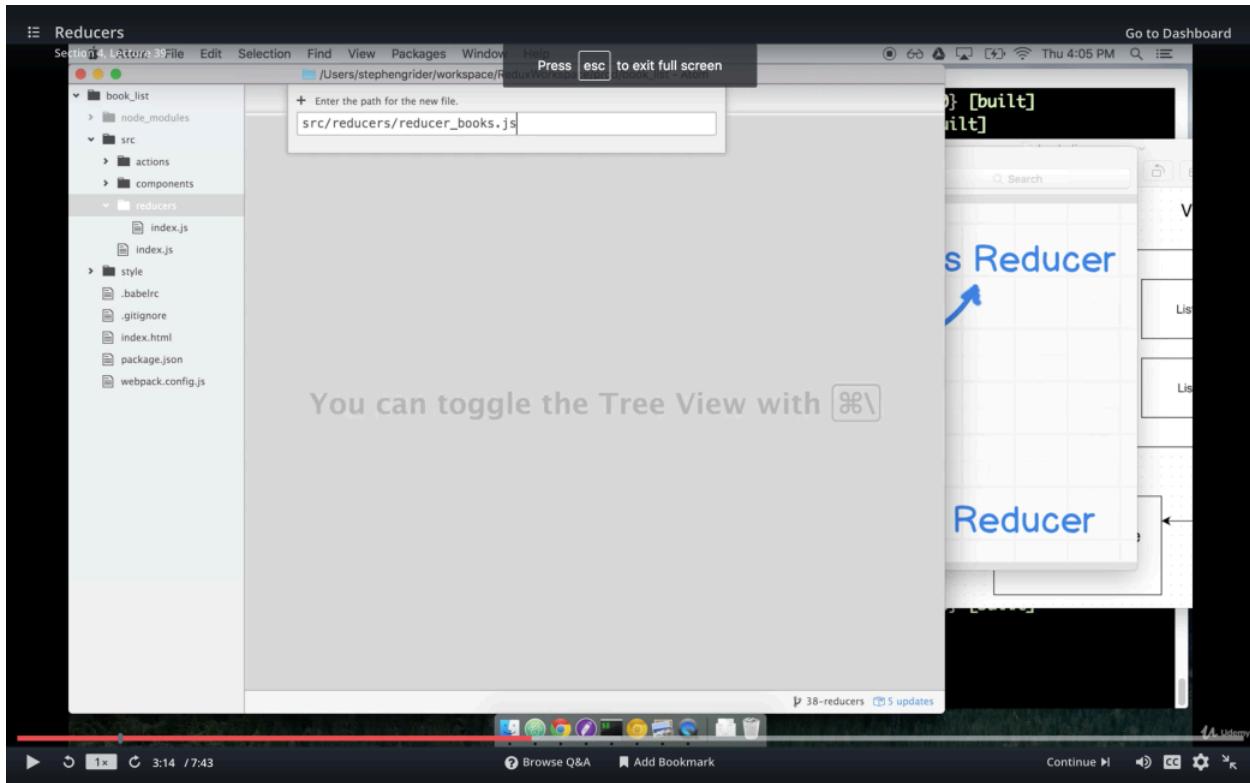
2 Reducers will produce value of state

The screenshot shows a Chrome browser window displaying a Redux reducer code example. The code is as follows:

```
// Application State - Generated by reducers
{
  books: [ { title: 'Harry Potter' }, { title: 'Javascript' } ],
  activeBook: { title: 'Javascript: The Good Parts' }
}
```

Two blue arrows point from the text "Books Reducer" and "ActiveBook Reducer" to the respective parts of the state object. The "Books Reducer" arrow points to the "books" key, and the "ActiveBook Reducer" arrow points to the "activeBook" key.

create a reducer
reducer_books.js



start with simple function
that just return array of books

A screenshot of the Atom code editor showing a file named `reducer_books.js`. The code defines a function that returns an array of book objects:`function() {
 return [
 { title: 'Javascript: The Good Parts' },
 { title: 'Harry Potter' },
 { title: 'The Dark Tower' },
 { title: 'Eloquent Ruby' }
]
}`The sidebar shows a project structure with files like `index.js`, `actions`, `components`, and `reducers`. A hand-drawn diagram is overlaid on the right side of the screen, showing a box labeled "Reducer" with arrows pointing to it from the text "Reducer" and "s Reducer".

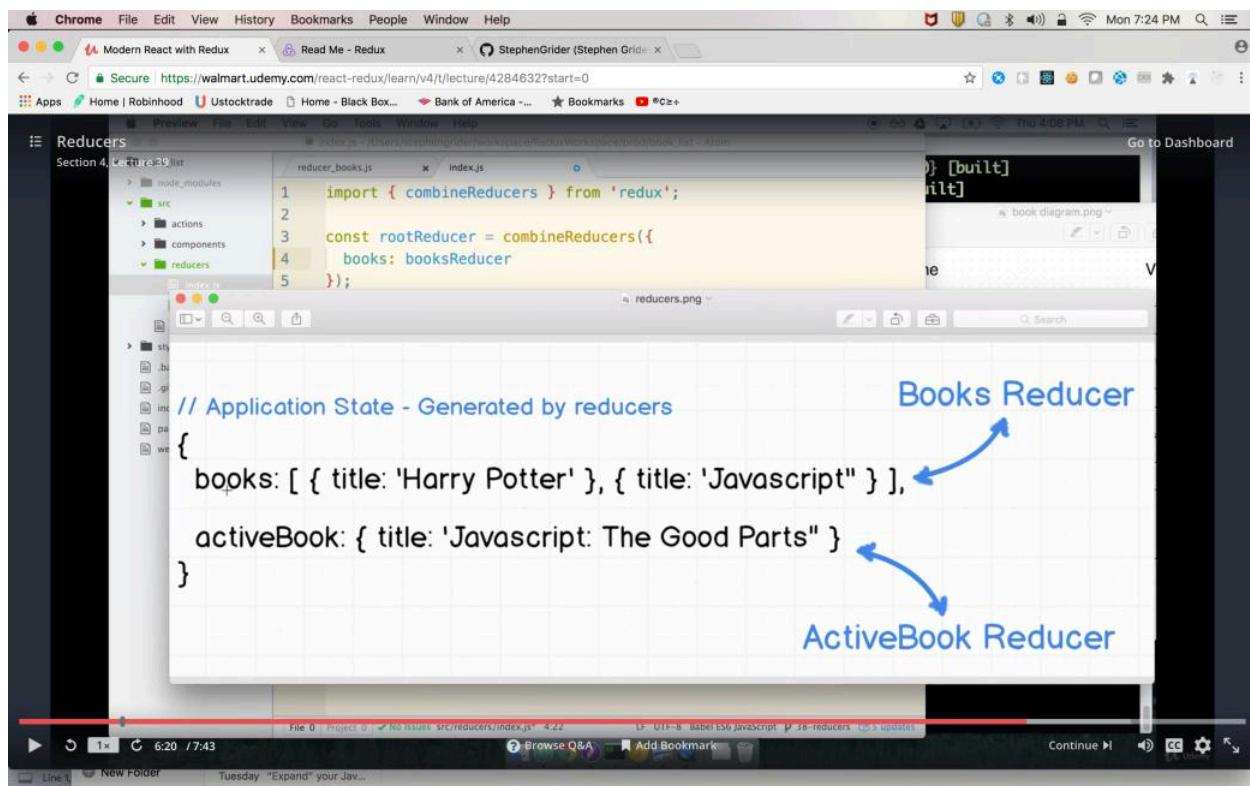
export the function
so it can be used anywhere else in project

A screenshot of the Atom code editor showing the same `reducer_books.js` file, but now with an `export` statement at the top:`export default function() {
 return [
 { title: 'Javascript: The Good Parts' },
 { title: 'Harry Potter' },
 { title: 'The Dark Tower' },
 { title: 'Eloquent Ruby' }
]
}`The sidebar and hand-drawn diagram are identical to the previous screenshot.

now wire this to our application
we will do into index.js
where we import combineReducers from 'redux'

combineReducers({state: reducer})
in combineReducers function we will pass a object {state: reducer}

we will really map the state to reducer
books is state
booksReducer is reducer



```
import { combineReducers } from 'redux';
const rootReducer = combineReducers({
  books: booksReducer
});
// Application State - Generated by reducers
{
  books: [ { title: 'Harry Potter' }, { title: 'Javascript' } ],
  activeBook: { title: 'Javascript: The Good Parts' }
}
```

Handwritten annotations:

- An arrow points from the word "Books Reducer" to the "books" key in the state object.
- An arrow points from the word "ActiveBook Reducer" to the "activeBook" key in the state object.

import the reducer_books

and capitalize the BooksReducers

The screenshot shows a browser window with a Udemy course page for 'Modern React with Redux'. The browser tabs include 'Secure https://walmart.udemy.com/react-redux/learn/v4/t/lecture/4284632?start=0' and 'StephenGrider [Stephen Grider]'. Below the browser is the Atom code editor. The project structure on the left shows 'Reducers' and 'Section 4, Lecture 4: Book List'. The 'reducer_books.js' file is open, containing the following code:

```
1 import { combineReducers } from 'redux';
2 import BooksReducer from './reducer_books';
3
4 const rootReducer = combineReducers({
5   books: BooksReducer
6 });
7
8 export default rootReducer;
9
```

The right side of the screen displays a hand-drawn style diagram on a grid background. It features the words 'Books Reducer' and 'Book Reducer' in blue ink, with a blue arrow pointing from 'Books Reducer' to 'Book Reducer'.

To test this we will create book-list component in React

A screenshot of the Atom IDE interface. The left sidebar shows a project structure for a 'book_list' folder, including 'node_modules', 'src' (with 'actions', 'components' containing 'app.js' and 'book-list.js', 'reducers', 'index.js', 'style', '.babelrc', '.gitignore', 'index.html', 'package.json', and 'webpack.config.js'), and 'index.html'. The right pane displays the contents of 'book-list.js':

```
1 import React, { Component } from 'react';
2
3 export default class BookList extends Component {
4 }
5 }
```

The status bar at the bottom indicates 'File 0 Project 0 No Issues src/components/book-list.js* 4 LF UTF-8 Babel ES6 JavaScript 39-containers 1 5 updates'. The bottom right corner shows the Udemy logo.

A screenshot of the Atom IDE interface, identical to the one above but with more code added to 'book-list.js'. The right pane now shows:

```
1 import React, { Component } from 'react';
2
3 export default class BookList extends Component {
4   render() {
5     return (
6       <ul className="list-group col-sm-4">
7
8
9     </ul>
10  }
11}
```

The status bar at the bottom indicates 'File 0 Project 0 No Issues src/components/book-list.js* 7 LF UTF-8 Babel ES6 JavaScript 39-containers 1 5 updates'. The bottom right corner shows the Udemy logo.

if want to call a separate javascript function in jsx
we will wrap in curly braces

```
{this.renderList()}
```

The screenshot shows the Atom IDE interface. On the left is a file tree for a 'book_list' project. The 'book-list.js' file is open in the center editor pane, containing the following code:

```
1 import React, { Component } from 'react';
2
3 export default class BookList extends Component {
4   renderList() {
5     return (
6       <ul className="list-group col-sm-4">
7         {this.renderList()}
8       </ul>
9     )
10  }
11}
12
13
14
15}
```

The right side of the interface shows a terminal window with the output of a command, likely related to the code above. The output includes several '[built]' entries and a file size of '854 bytes'. At the bottom of the screen, there is a browser toolbar with various icons.

we will assume this.props will return books array
map over that array
for each element in array we will return a li.
don't forget to add key because it is a list . we will title to give unique value to keys

The screenshot shows the Atom IDE interface with the following details:

- Title Bar:** Containers - Connecting Redux to React
- File Explorer:** Shows the project structure with files like book-list.js, reducer_books.js, index.js, and app.js.
- Code Editor:** The file book-list.js is open, displaying the following code:

```
1 import React, { Component } from 'react';
2
3 export default class BookList extends Component {
4   renderList() {
5     return this.props.books.map((book) => {
6       |
7     });
8   }
9
10  render() {
11    return (
12      <ul className="list-group col-sm-4">
13        {this.renderList()}
14      </ul>
15    )
16  }
17}
```

- Terminal:** Shows the output of a build process, indicating successful compilation of various files into built artifacts.
- Bottom Status Bar:** Shows the file is saved (File 0), no issues found (No Issues), the current file (src/components/book-list.js), encoding (UTF-8), Babel ES6 JavaScript support, and other project details.

The screenshot shows the Atom code editor interface with the following details:

- File Structure:** The left sidebar shows a project structure for "book_list".
- Open Files:** Three files are open in tabs:
 - `reducer_books.js`
 - `index.js`
 - `book-list.js`
- Code View:** The `book-list.js` file contains the following code:

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

export default class BookList extends Component {
  renderList() {
    return this.props.books.map((book) => {
      return (
        <li className="list-group-item">{book.title}</li>
      );
    });
  }

  render() {
    return (
      <ul className="list-group col-sm-4">
        {this.renderList()}
      </ul>
    );
  }
}
```
- Output Panel:** On the right, the output panel displays the results of the Babel compilation, showing the generated JavaScript code.
- Bottom Status Bar:** The status bar at the bottom shows the file path (`src/components/book-list.js*`), line count (7), character count (1854), and other build-related information.

The screenshot shows the Atom IDE interface. On the left is a file tree for a project named 'book_list'. The main area contains three tabs: 'reducer_books.js', 'index.js', and 'book-list.js'. The 'book-list.js' tab is active and displays the following code:

```
1 import React, { Component } from 'react';
2
3 export default class BookList extends Component {
4   renderList() {
5     return this.props.books.map((book) => {
6       return (
7         <li key={book.title} className="list-group-item">{book.title}</li>
8       );
9     });
10 }
11
12 render() {
13   return (
14     <ul className="list-group col-sm-4">
15       {this.renderList()}
16     </ul>
17   )
18 }
19 }
```

To the right of the code editor is a terminal window showing build output:

```
0} [built]
0} [built]
built]
[0} [built]
0} [built]
lt]
is 854 bytes {0} [built]
0} [built]
```

At the bottom of the screen, there is a toolbar with various icons.

now we need to connect
react views with redux state

connecting react and redux lib done with separate lib react-redux

we will define one of our components as container

A Container is react component that has a direct connection with state manage by redux

we will put all containers in a separate dir calls Conainer

```
1  src/containers
2
3  export default class BookList extends Component {
4    renderList() {
5      return this.props.books.map((book) => {
6        return (
7          <li key={book.title} className="list-group-item">{book.title}</li>
8        );
9      );
10     }
11   }
12
13   render() {
14     return (
15       <ul className="list-group col-sm-4">
16         {this.renderList()}
17       </ul>
18     )
19   }
20 }
```

cut our book-list and from components and past it containers

A screenshot of the Atom code editor interface. The main window displays two files: 'reducer_books.js' and 'index.js'. A context menu is open over the file 'book-list.js' in the left sidebar, listing options like 'Cut' and 'Paste'. The status bar at the bottom shows 'File 0 - Project 0 | No issues src/reducers/index.js 4:17'.

```
1 import { combineReducers } from 'redux';
2 import BooksReducer from './reducer_books';
3
4 const rootReducer = combineReducers({
5   books: BooksReducer
6 })
7
8 export default rootReducer;
```

A second screenshot of the Atom editor interface, identical to the first one, showing the context menu over the 'book-list.js' file. The status bar at the bottom shows 'File 0 - Project 0 | No issues src/reducers/index.js 4:17'.

```
1 import { combineReducers } from 'redux';
2 import BooksReducer from './reducer_books';
3
4 const rootReducer = combineReducers({
5   books: BooksReducer
6 })
7
8 export default rootReducer;
```

The screenshot shows the Atom IDE interface with the following details:

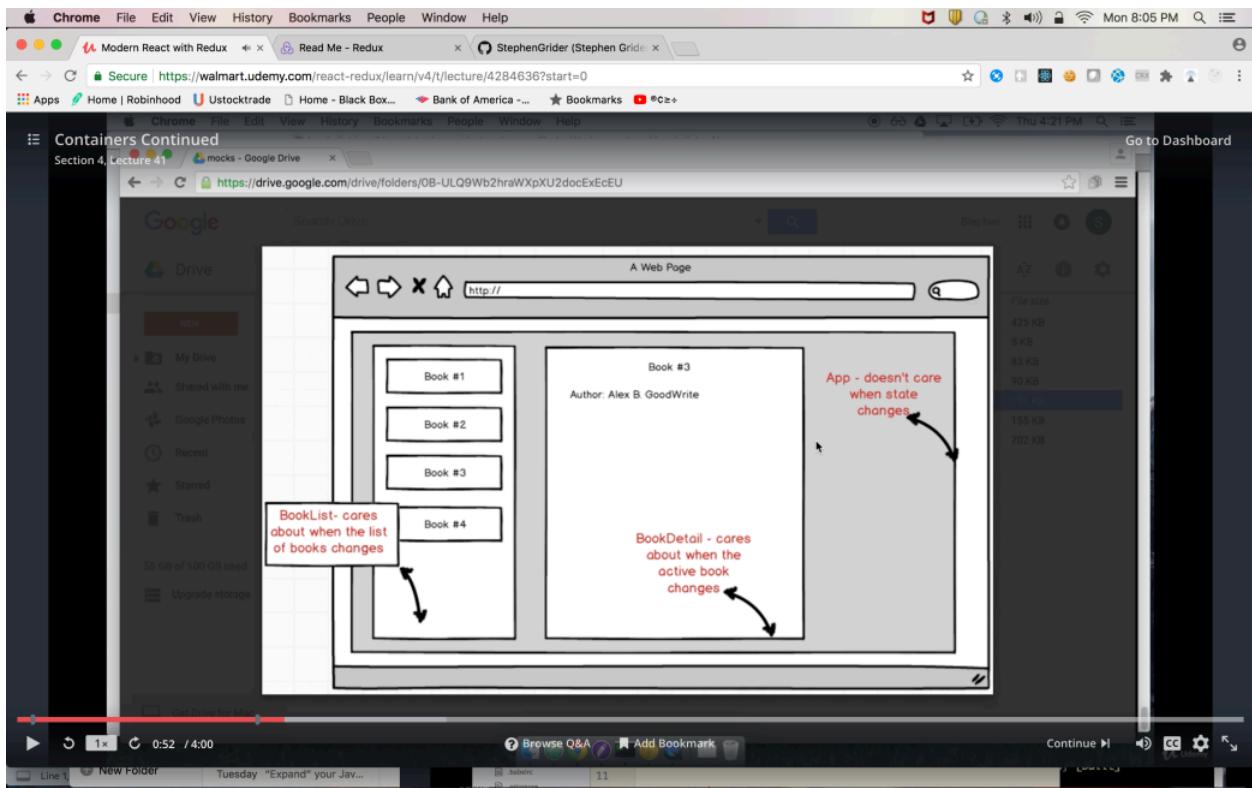
- File Explorer:** On the left, it shows a project structure for "Section 4, Lecture 01". The "book-list.js" file is selected.
- Code Editor:** The main area displays two files:
 - `reducer_books.js`:

```
1 import { combineReducers } from 'redux';
2 import BooksReducer from './reducer_books';
3
4 const rootReducer = combineReducers({
5   books: BooksReducer
6 });
7
8 export default rootReducer;
9
```
 - `index.js`:

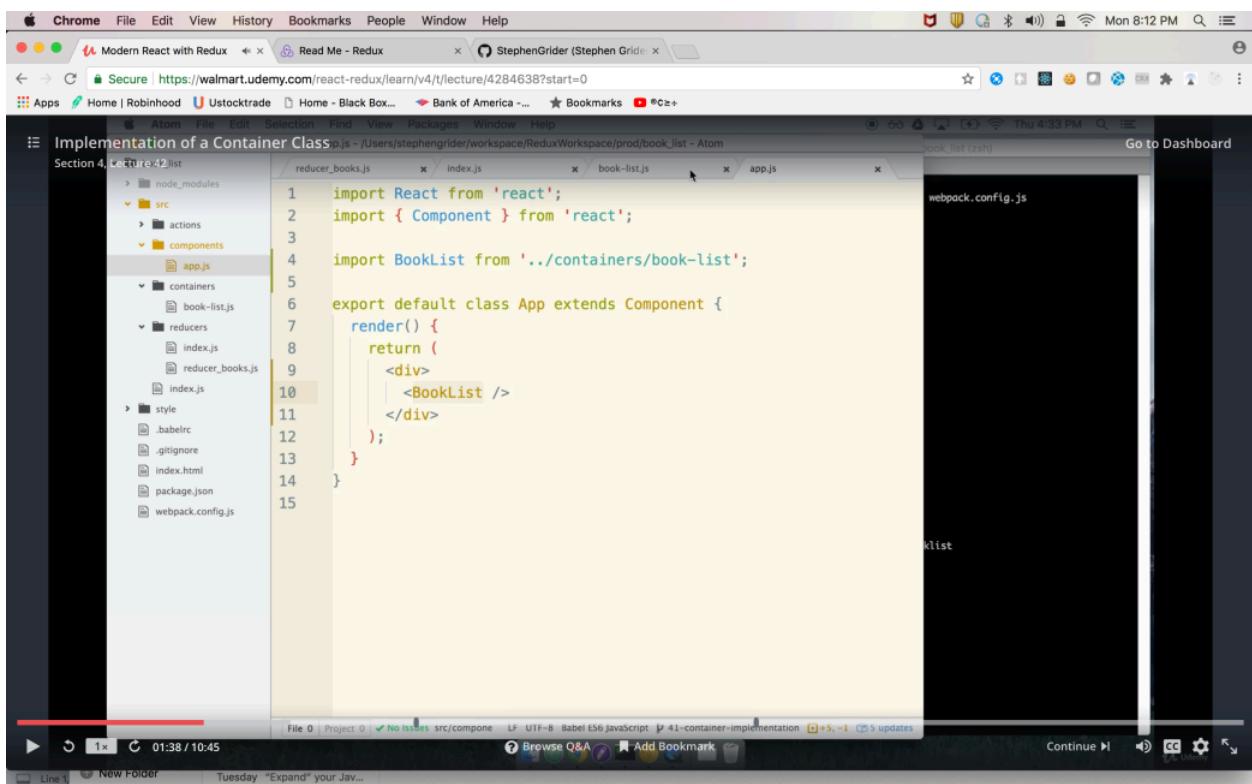
```
1 [built]
2 [built]
3 [built]
4 [built]
5 [built]
6 [built]
7 [built]
8 [built]
9 [built]
```
- Status Bar:** At the bottom, it shows "File 0 - Project 0 | No issues src/reducers/index.js 4/17" and "Tuesday "Expand" your Java...".

Now q : which component we want to move to container
in general we keep most parent component in container

App does



Now first add BookList to app.js



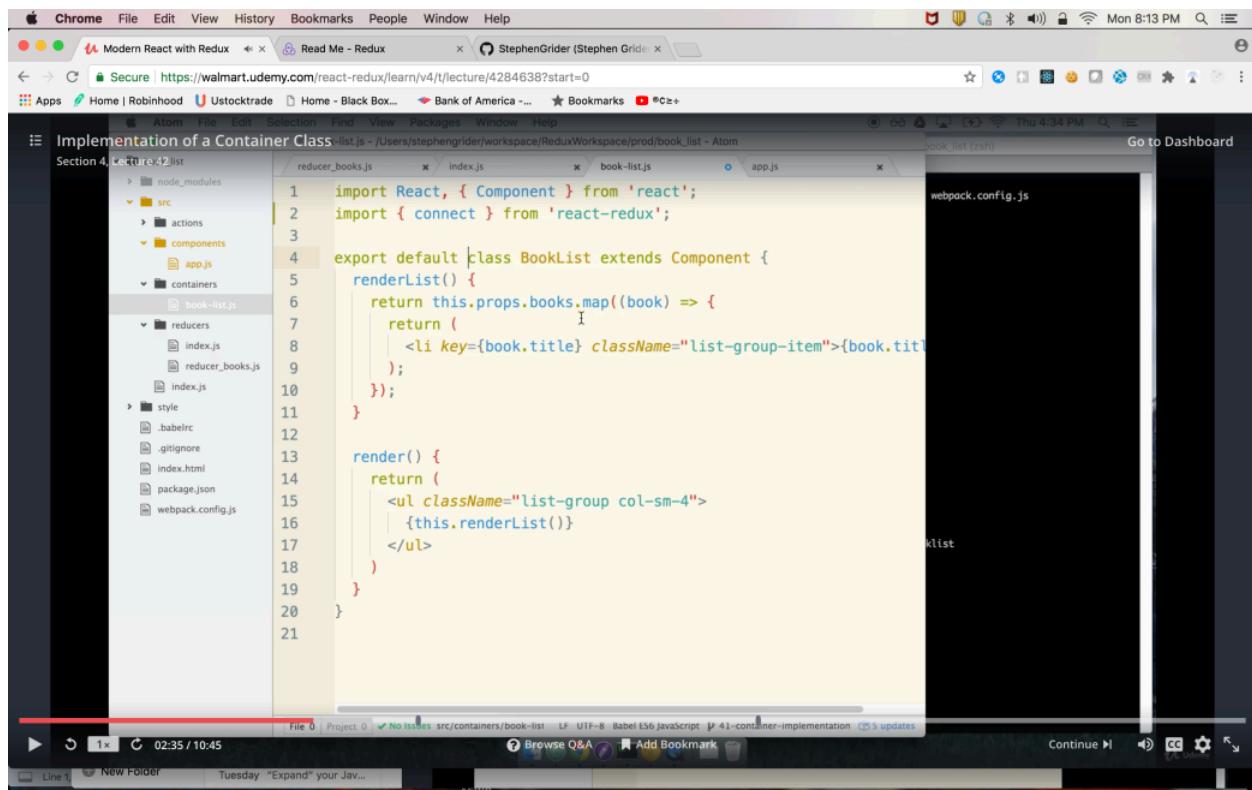
```

import queryString from "query-string";
import extend from "lodash/extend";
import isEmpty from "lodash/extend";
import Promise from "bluebird";

import { fetchJSON } from "@walmart/electrode-fetch";
import {
  transformItemBasePayload,
  transformCustomerBasePayload
} from "../helpers/apollo/apollo-helper";

```

{connect} import a single property from 'react-redux'
 if import a word like **React** it imports **entire object** from file



```

1  import React, { Component } from 'react';
2  import { connect } from 'react-redux';
3
4  export default class BookList extends Component {
5    renderList() {
6      return this.props.books.map((book) => {
7        return (
8          <li key={book.title} className="list-group-item">{book.title}</li>
9        );
10      });
11    }
12
13    render() {
14      return (
15        <ul className="list-group col-sm-4">
16          {this.renderList()}
17        </ul>
18      )
19    }
20  }
21

```

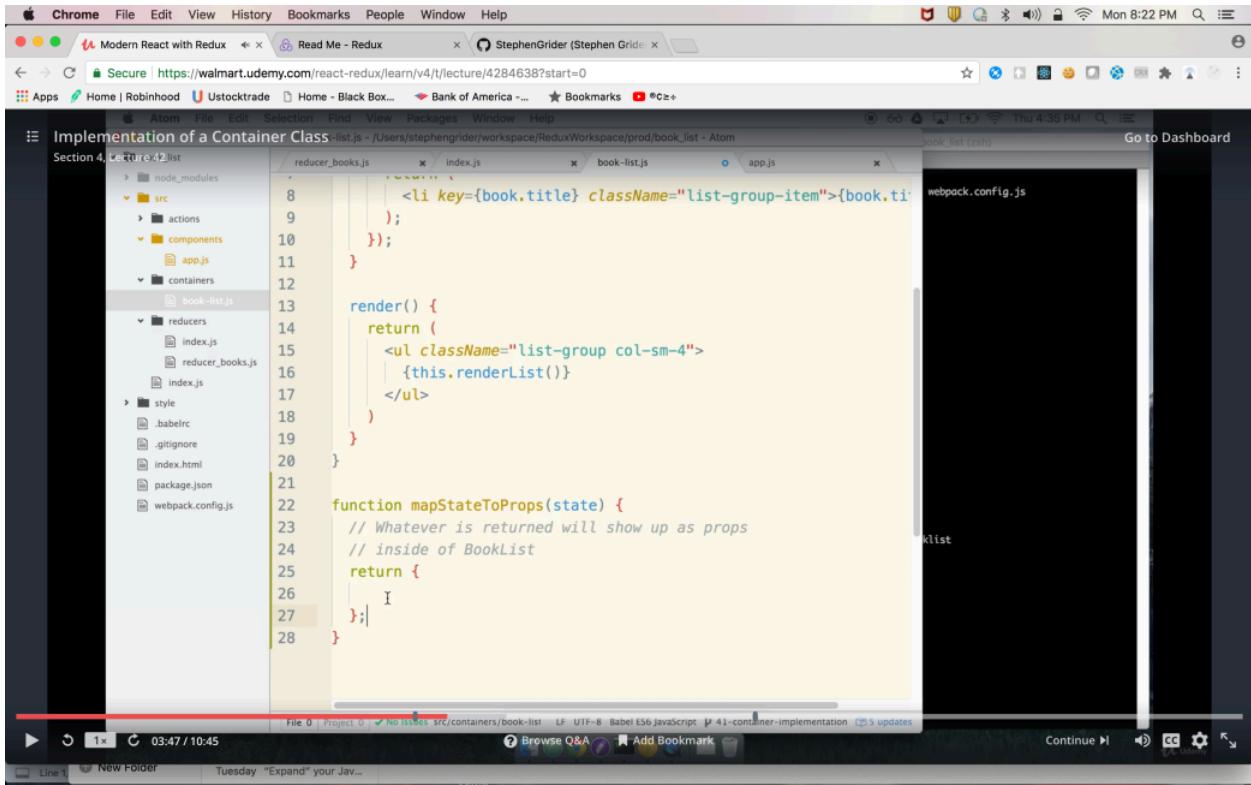
remove export default from class

The screenshot shows the Atom code editor interface. On the left, the file tree displays a project structure under 'Section 4, Lecture 4'. The current file is 'book-list.js' in the 'containers' folder. The code in the editor is:

```
1 import React, { Component } from 'react';
2 import { connect } from 'react-redux';
3
4 class BookList extends Component {
5   renderList() {
6     return this.props.books.map((book) => {
7       return (
8         <li key={book.title} className="list-group-item">{book.title}</li>
9       );
10    });
11  }
12
13  render() {
14    return (
15      <ul className="list-group col-sm-4">
16        {this.renderList()}
17      </ul>
18    )
19  }
20}
21
```

The right side of the screen shows a dark terminal window with the title 'book-list (zsh)' and the command 'webpack.config.js' entered. The bottom status bar shows the file path 'File 0 - Project 0 | No issues src/containers/book-list' and the commit message 'Tuesday "Expand" your Java...'. The bottom right corner of the status bar has a small number '3'.

outside of class we will define a function `mapStateToProps`
it will have one argument that is state
and it returns a object
whatever is return as object from `mapStateToProps` it
will show up in props inside BookList class



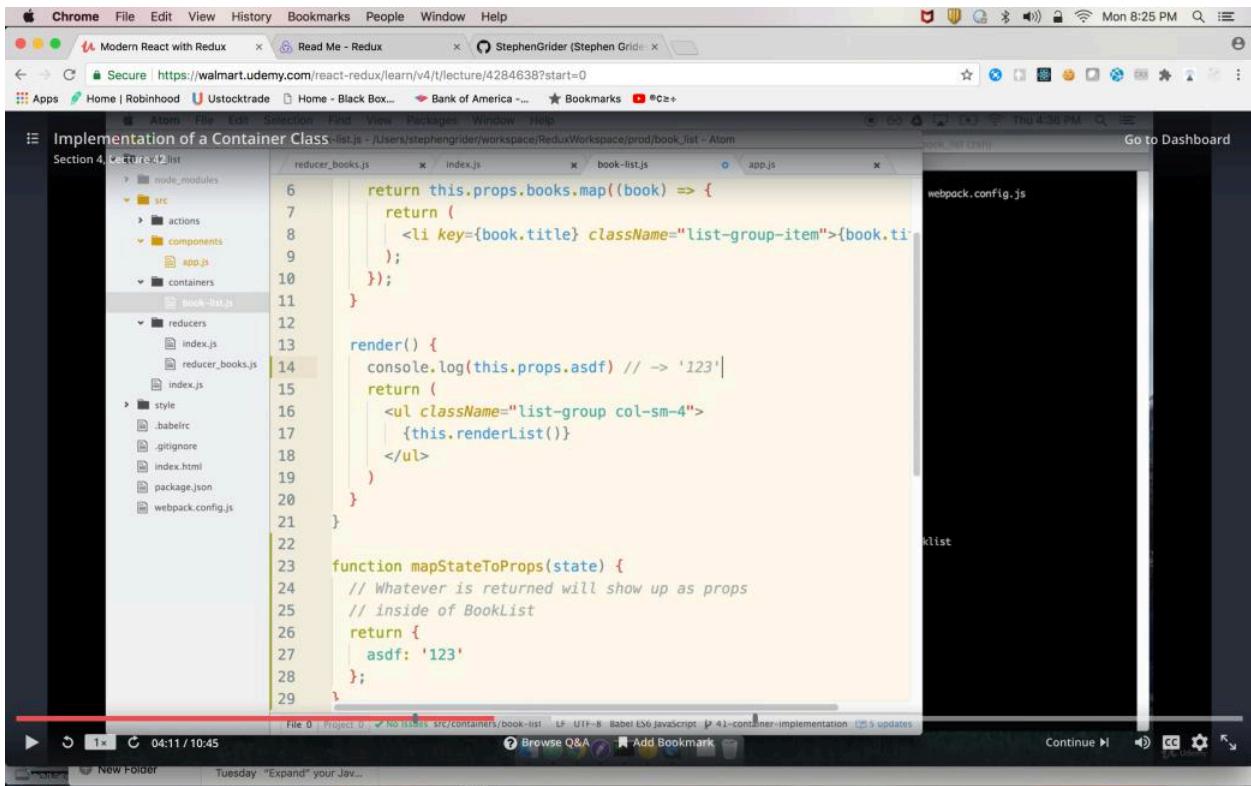
A screenshot of a Chrome browser window displaying code in a file named `book-list.js`. The code is part of a container class implementation:

```
    <li key={book.title} className="list-group-item">{book.title}</li>
  );
}

render() {
  return (
    <ul className="list-group col-sm-4">
      {this.renderList()}
    </ul>
  )
}

function mapStateToProps(state) {
  // Whatever is returned will show up as props
  // inside of BookList
  return {
    books: state.books
  };
}
```

The browser interface shows the file path `src/containers/book-list` in the address bar. The code editor has syntax highlighting for JavaScript and CSS classes.



A screenshot of a Chrome browser window displaying code in `book-list.js`. A `console.log` statement has been added to the code:

```
return this.props.books.map((book) => {
  return (
    <li key={book.title} className="list-group-item">{book.title}</li>
  );
}

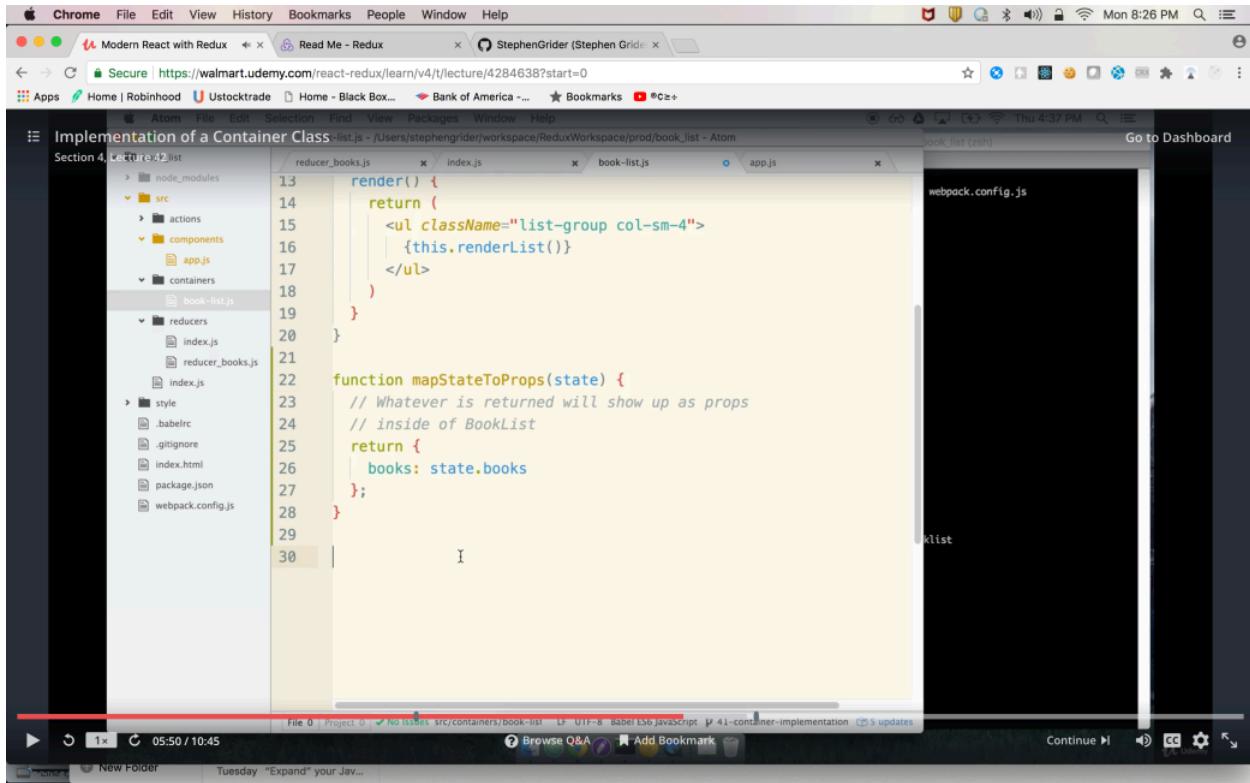
render() {
  console.log(this.props.asdf) // -> '123'
  return (
    <ul className="list-group col-sm-4">
      {this.renderList()}
    </ul>
  )
}

function mapStateToProps(state) {
  // Whatever is returned will show up as props
  // inside of BookList
  return {
    asdf: '123'
  };
}
```

The browser interface shows the file path `src/containers/book-list` in the address bar. The code editor has syntax highlighting for JavaScript and CSS classes.

since we are using `this.props.books`

we will return { **books** : books from state }



```
render() {
  return (
    <ul className="list-group col-sm-4">
      {this.renderList()}
    </ul>
  )
}

function mapStateToProps(state) {
  // Whatever is returned will show up as props
  // inside of BookList
  return {
    books: state.books
  };
}
```

we don't return export default BookList

instead of this we export default connect which expects a function and component

export default connect(mapStateToProps) it returns a function that which is called with BookList class

export default connect(mapStateToProps)(BookList)

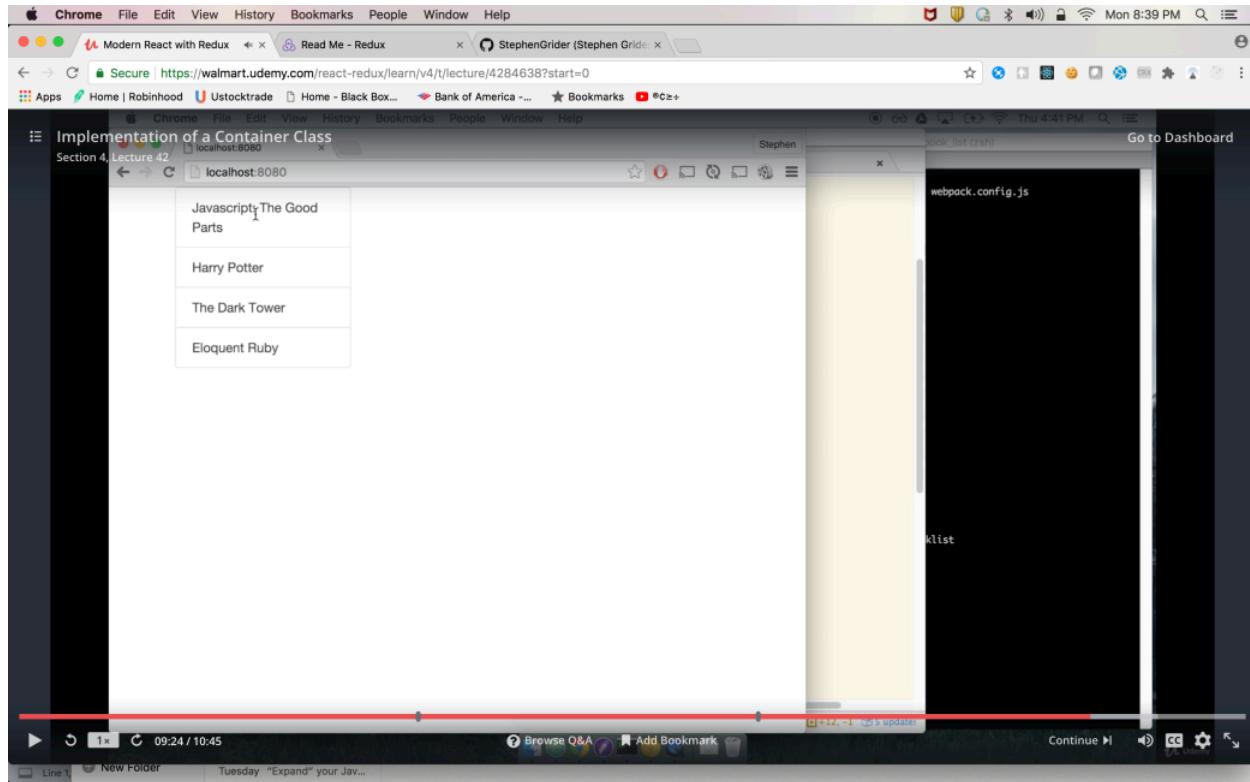
```
12
13     render() {
14         return (
15             <ul className="list-group col-sm-4">
16                 {this.renderList()}
17             </ul>
18         )
19     }
20 }
21
22 function mapStateToProps(state) {
23     // Whatever is returned will show up as props
24     // inside of BookList
25     return {
26         books: state.books
27     };
28 }
29
30 export default connect(mapStateToProps)(BookList);
```

```
12
13     render() {
14         return (
15             <ul className="list-group col-sm-4">
16                 {this.renderList()}
17             </ul>
18         )
19     }
20 }
21
22 function mapStateToProps(state) {
23     // Whatever is returned will show up as props
24     // inside of BookList
25     return {
26         books: state.books
27     };
28 }
29
30 export default connect(mapStateToProps)(BookList);
```

2 Imp notes

whenever application **states changes** (like list of books changes)
this container will re render with new list of books

and object in **mapStateToProps** will assign as props to component



User click on Book 2

we call an Action Creator .

Action Creator is a function which **return a object** which we called **Action { ACTION.TYPE: actionname, DATA: {data object}, }**

```
Action :{  
  type: BOOK_SELECTED // object has a TYPE of ACTION is triggered  
  book : {title: 'Book 2'}           // object also has DATA that further describes the action.
```

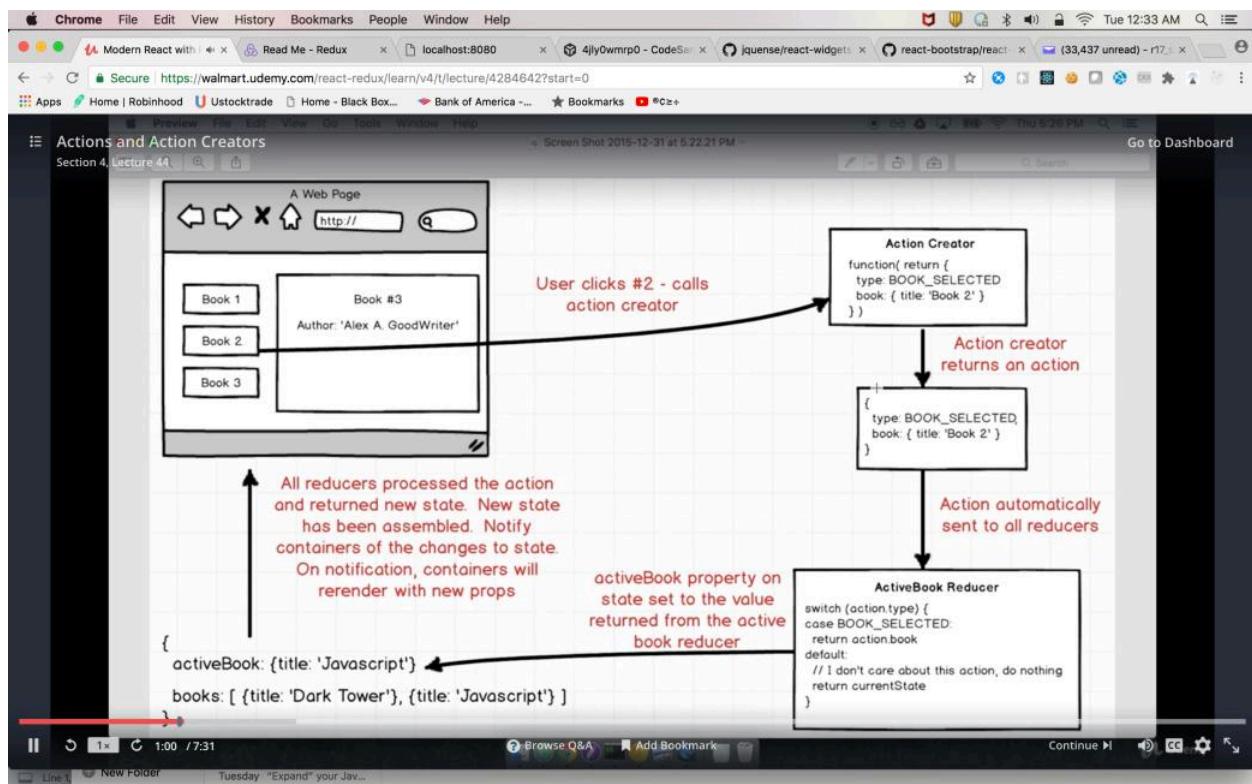
```
}
```

Action sent To all reducers in the application

Reducer has **switch case** based on action.type
it returns the value which is set to state

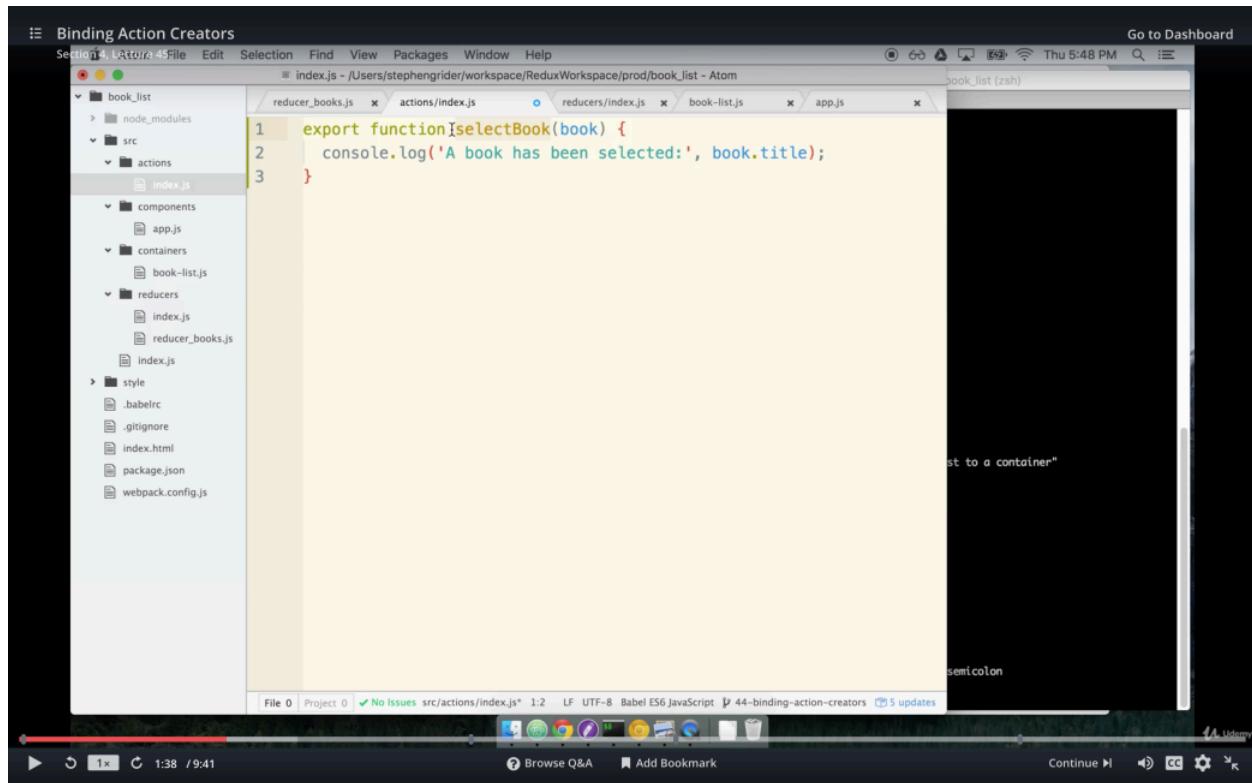
```
switch (action.type) {  
  case "": {  
  }  
  default "":{  
  }  
}
```

once reducer return the state to **mapStateToProps**
New state flow to all the containers
which it will cause **re render** of page.



go to **actions/index.js**
create single **action creator selectBook**

action Creator is just function
it will only take one book object with title property
we will export the function



The screenshot shows the Atom code editor interface. The left sidebar displays a project structure for a 'book_list' application, including 'src' and 'actions' directories. The main editor window shows the contents of 'actions/index.js':

```
1 export function selectBook(book) {
2   console.log('A book has been selected:', book.title);
3 }
```

The status bar at the bottom indicates the file is 1/2, uses LF line endings, and is written in Babel ES6 JavaScript. A tooltip 'semicolon' is visible near the end of the third line.

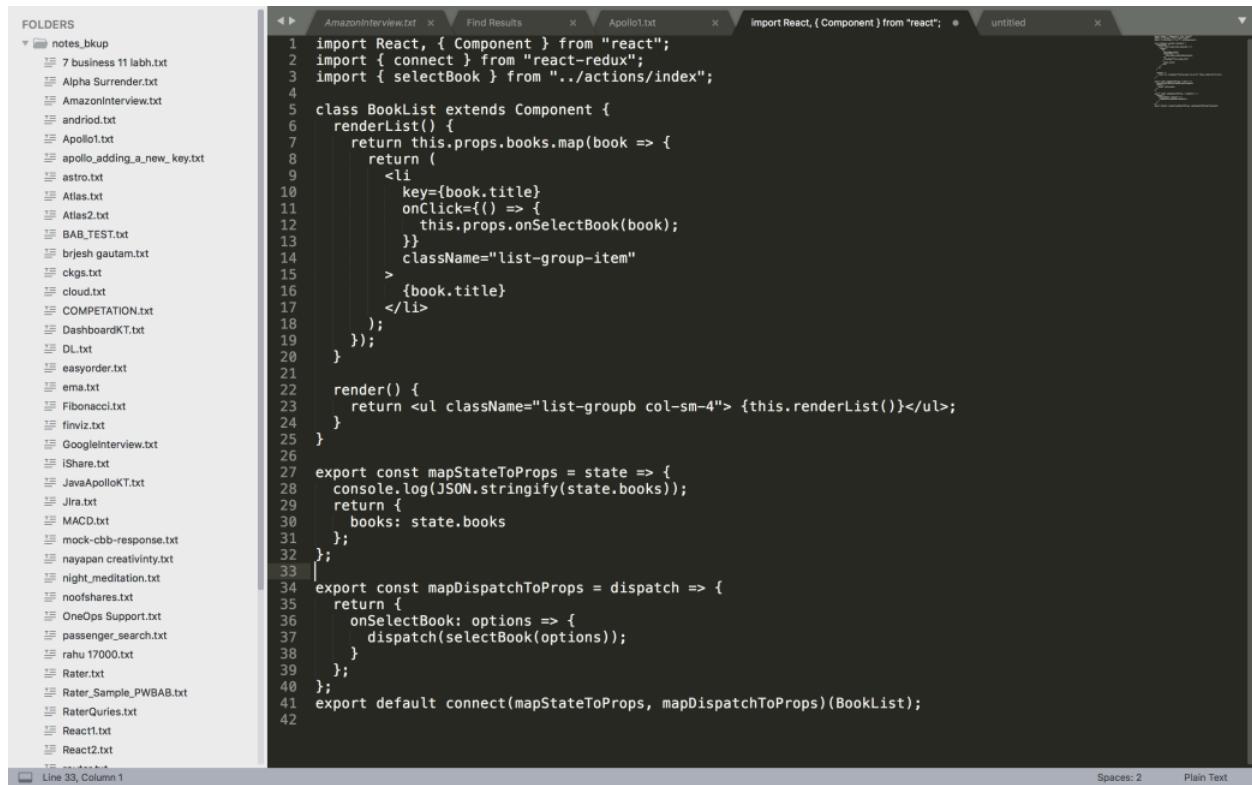
import {selectBook} action to this BookList component

1 // we will add **mapDispatchToProps** outside of class BookList (similar to **mapStateToProps**)

2 //

```
add function mapDispatchToProps = (dispatch) => {
  return {
    onSelectBook : (options) => dispatch (selectBook(options));
  }
}
```

3 in connect method we will add another argument with mapStateToProps, mapDispatchToProps



```
1 import React, { Component } from "react";
2 import { connect } from "react-redux";
3 import { selectBook } from "../actions/index";
4
5 class BookList extends Component {
6   renderList() {
7     return this.props.books.map(book => {
8       return (
9         <li
10           key={book.title}
11           onClick={() => {
12             this.props.onSelectBook(book);
13           }}
14           className="list-group-item"
15         >
16           {book.title}
17         </li>
18       );
19     });
20   }
21   render() {
22     return <ul className="list-group col-sm-4"> {this.renderList()}</ul>;
23   }
24 }
25
26 export const mapStateToProps = state => {
27   console.log(JSON.stringify(state.books));
28   return {
29     books: state.books
30   };
31 };
32
33 export const mapDispatchToProps = dispatch => {
34   return {
35     onSelectBook: options => {
36       dispatch(selectBook(options));
37     }
38   };
39 };
40
41 export default connect(mapStateToProps, mapDispatchToProps)(BookList);
```

whenever selectBook is called, the result should be passed to all the our reducers using dispatch function

anything return from this function will ends up in props on BookList container.

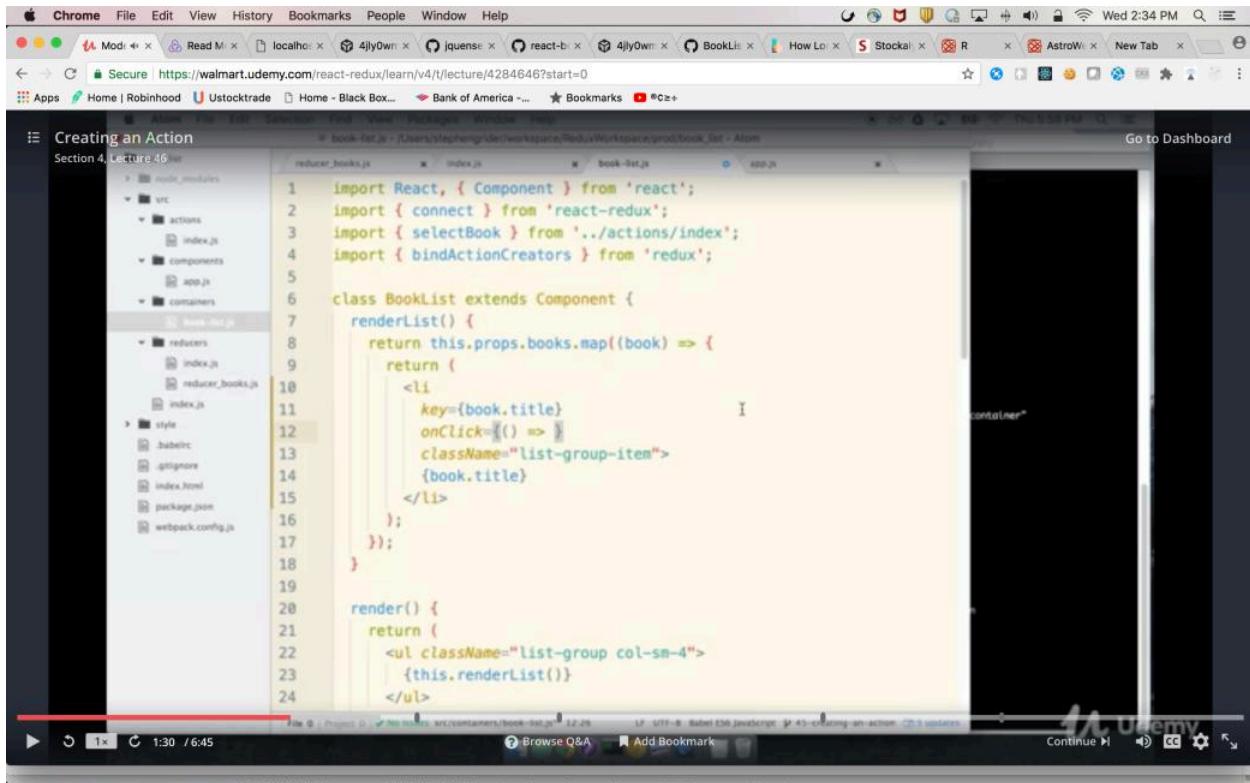
now we called this.props.onSelectBook() in our container that will call our action creator

we will add onClick on Li

onClick = {}

we will add arrow function

```
onClick = { () => {} }
```



```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { selectBook } from '../actions/index';
import { bindActionCreators } from 'redux';

class BookList extends Component {
  renderList() {
    return this.props.books.map(book) => {
      return (
        <li
          key={book.title}
          onClick={() =>}
          className="list-group-item">
          {book.title}
        </li>
      );
    });
  }

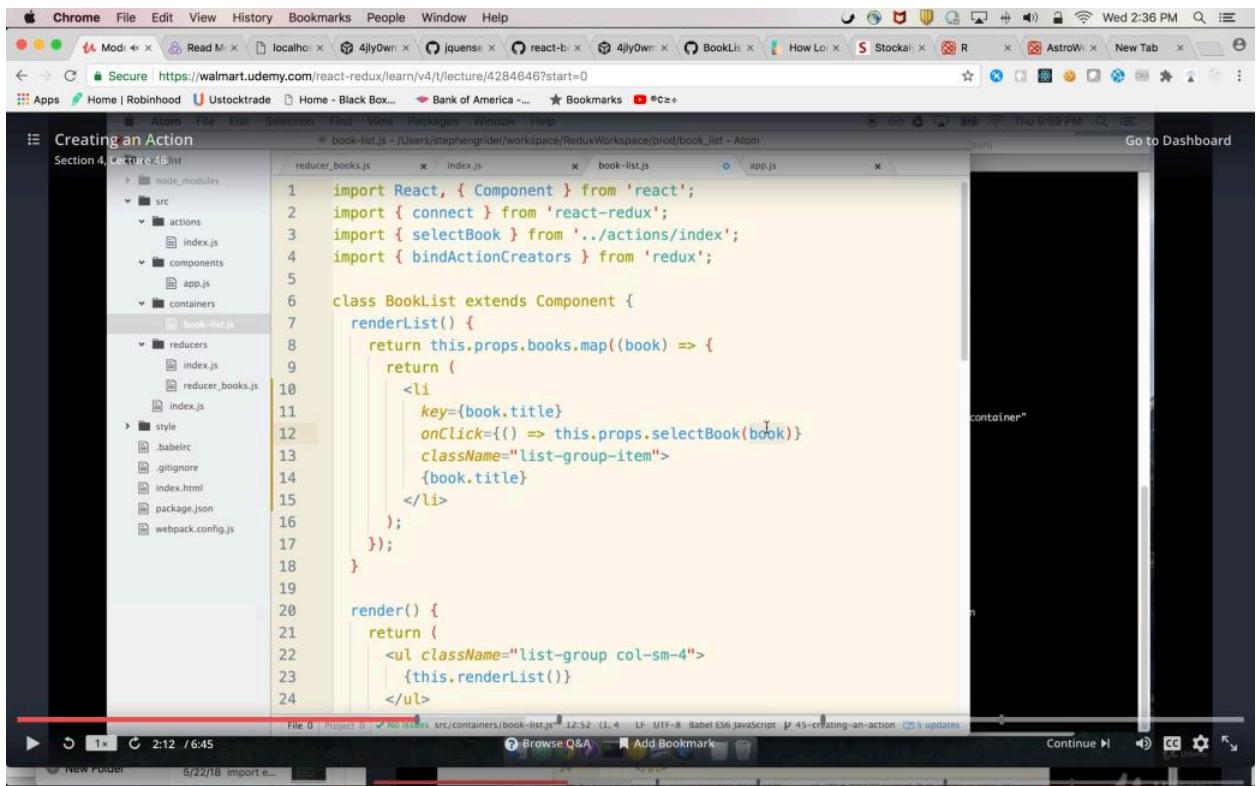
  render() {
    return (
      <ul className="list-group col-sm-4">
        {this.renderList()}
      </ul>
    );
  }
}

function mapStateToProps(state) {
  return { books: state.books };
}

function mapDispatchToProps(dispatch) {
  return bindActionCreators(selectBook, dispatch);
}

export default connect(mapStateToProps, mapDispatchToProps)(BookList);
```

we will call `this.props.selectBook()` action with book argument
`props.selectBook(book)`



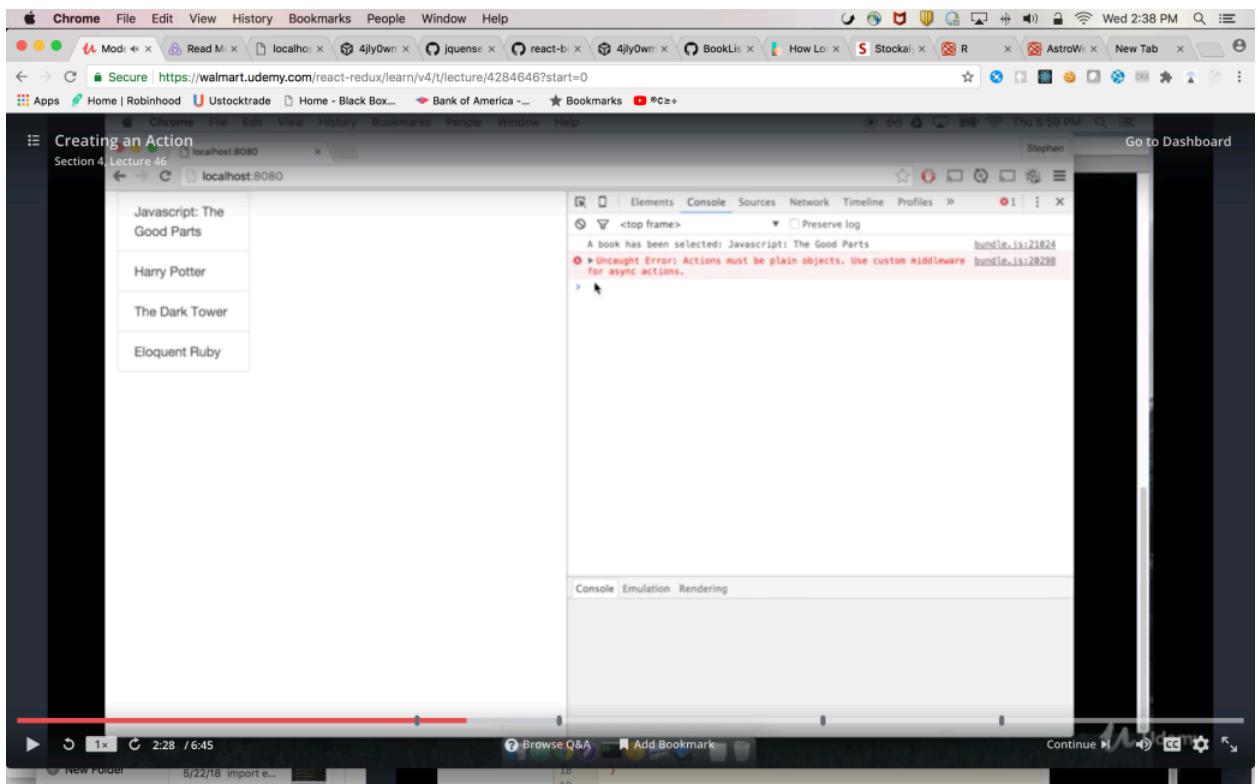
```
Creating an Action
Section 4, Lecture 46
File 0 - Project 0 - No issues  src/containers/book-list.js  12:52 (1, 4)  LF - UTF-8  Babel ES6 JavaScript  45 - creating-an-action  255 updates
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { selectBook } from '../actions/index';
import { bindActionCreators } from 'redux';

class BookList extends Component {
  renderList() {
    return this.props.books.map((book) => {
      return (
        <li
          key={book.title}
          onClick={() => this.props.selectBook(book)}
          className="list-group-item">
          {book.title}
        </li>
      );
    });
  }

  render() {
    return (
      <ul className="list-group col-sm-4">
        {this.renderList()}
      </ul>
    );
  }
}

export default BookList;
```

now click on container's li will show on action's console.log

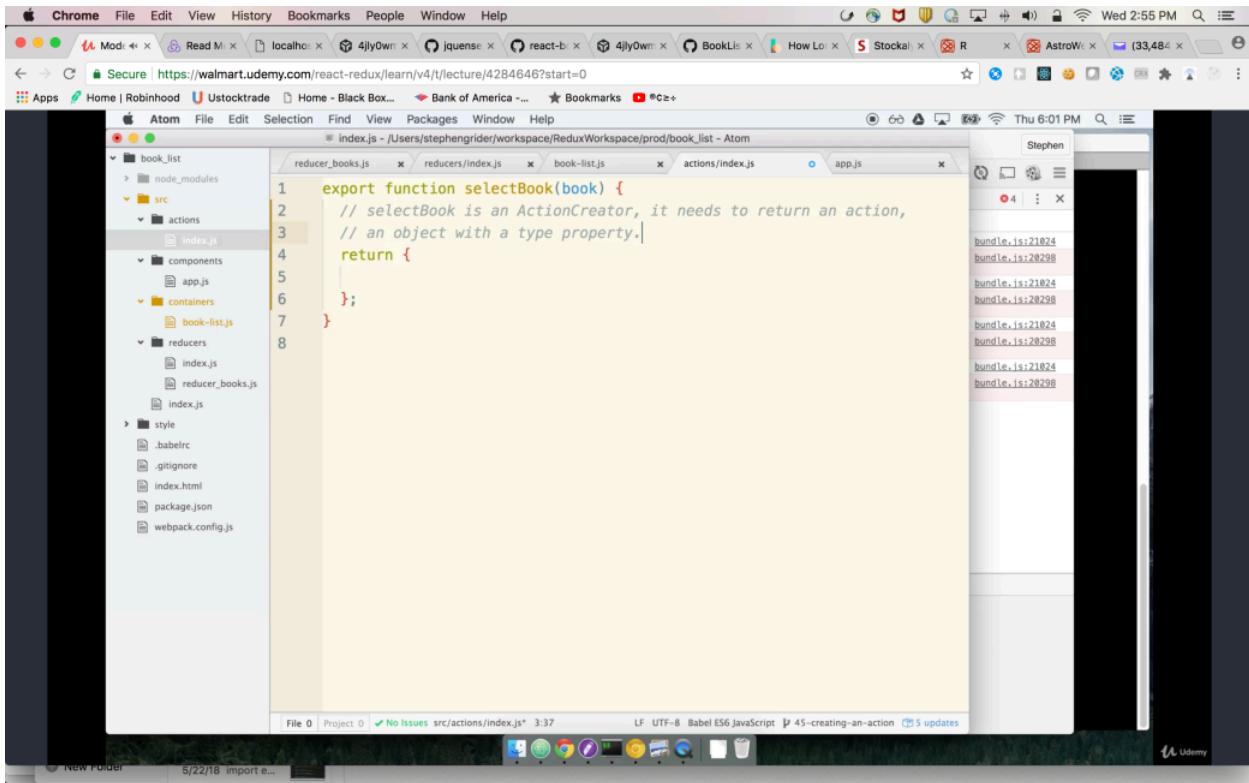


Javascript: The Good Parts
Harry Potter
The Dark Tower
Eloquent Ruby

<top frame> A book has been selected: Javascript: The Good Parts bundle.js:21828
Uncaught Error: Actions must be plain objects. Use custom middleware for async actions. bundle.js:28298

now selectBook we will change selectBook

```
//selectBook is a ActionCreator , it needs to return an action,  
// an object with type property and payload
```



```
index.js - /Users/stephengrider/workspace/ReduxWorkspace/prod/book_list - Atom
1  export function selectBook(book) {
2    // selectBook is an ActionCreator, it needs to return an action,
3    // an object with a type property.
4    return {
5      type: 'SELECT_BOOK',
6      payload: book
7    }
8  }
```

File 0 | Project 0 | No Issues src/actions/index.js* 3:37 LF UTF-8 Babel ES6 JavaScript 45-creating-an-action 5 updates

type: upper case string

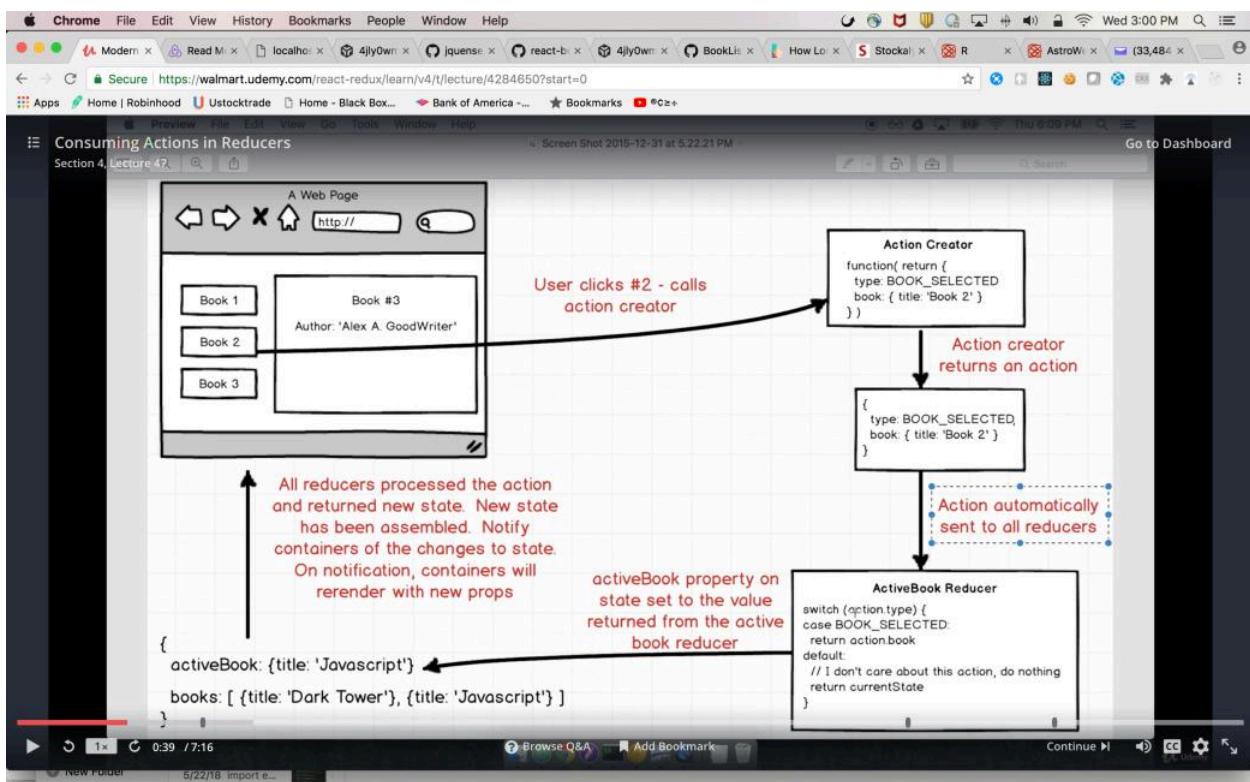
payload: data

```

1  export function selectBook(book) {
2      // selectBook is an ActionCreator, it needs to return an action,
3      // an object with a type property.
4      return {
5          type: 'BOOK_SELECTED',
6          payload: book
7      }
8  }

```

after this we will call ActiveBook Reducer



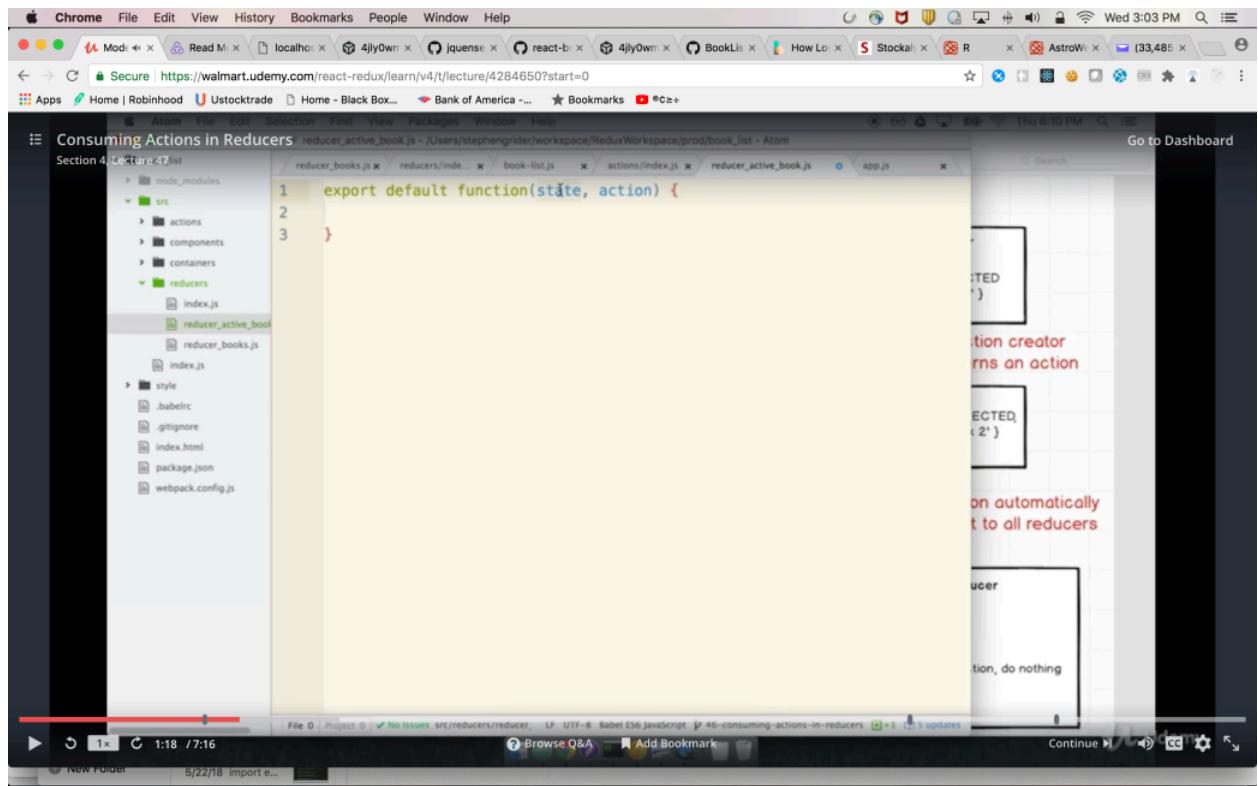
we will create new file in reducers dir
reducer_active_book.js

The screenshot shows a Mac desktop environment. In the background, a Chrome browser window is open to a Udemy course page about Redux. In the foreground, an Atom code editor window is active. A file dialog is open in the Atom window, prompting 'Enter the path for the new file.' with the path 'src/reducers/reducer_active_book.js' already typed in. The main code editor area contains the following JavaScript code:

```
1 // Enter the path for the new file.
2 // src/reducers/reducer_active_book.js
3 // an object with a type property.
4 return {
5   type: 'BOOK_SELECTED',
6   payload: book
7 };
8 }
```

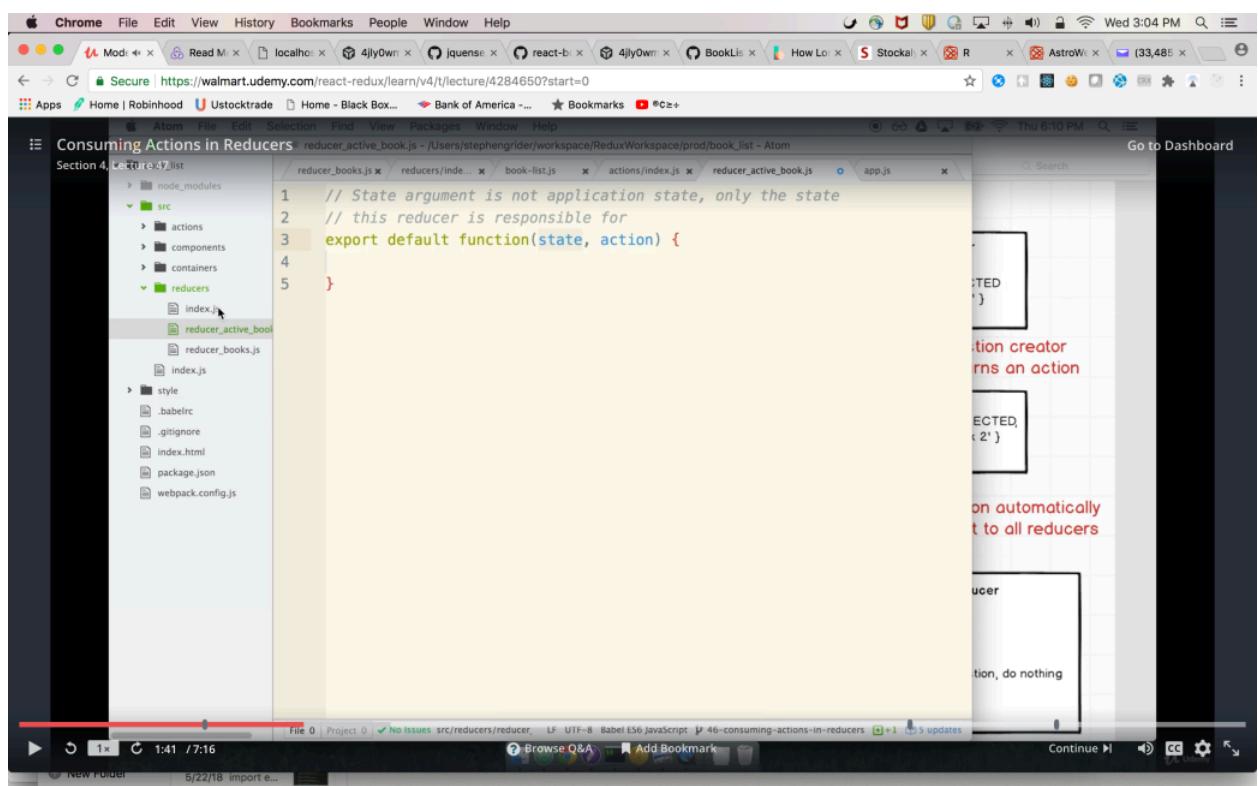
The code editor interface includes a sidebar with project files like 'book_list', 'node_modules', 'src', 'actions', 'components', 'containers', 'reducers', 'style', and various configuration files. The status bar at the bottom of the Atom window indicates 'File 0 | Project 0 | No issues src/actions/index.js 9:1 | UTF-8 | Babel ES6 JavaScript | 46-consuming-actions-in-reducers | 5 updates'. The system tray at the bottom of the screen shows various application icons.

every reducer has 2 arguments
state, action



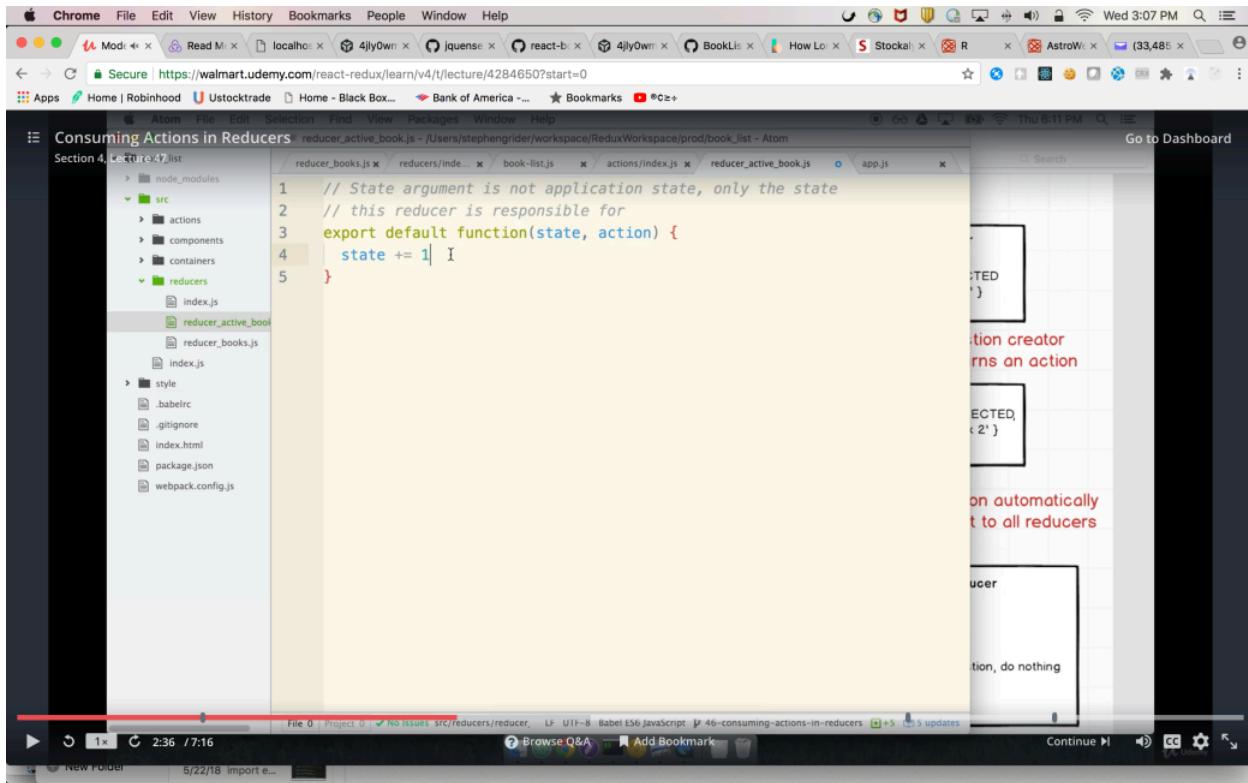
```
1 export default function(state, action) {
```

state is the current state of reducer not the application state



```
1 // State argument is not application state, only the state
2 // this reducer is responsible for
3 export default function(state, action) {
```

for example if state is number
first time action is triggered it will update to 1
next time action will triggered it will update to 2 and so on
same state is flowing into reducer over and over again when action is called



```
// State argument is not application state, only the state
// this reducer is responsible for
export default function(state, action) {
  state += 1
}
```

The floating window contains the following handwritten notes:

- Action creator returns an action
- Selected, <2>
- on automatically sent to all reducers
- Reducer
- on, do nothing

if we don't care about current action pass the current state back through

return state

The screenshot shows a web browser window with multiple tabs open, including one for a Udemy course on React-Redux. The main content area displays a code editor with the following code:

```
// State argument is not application state, only the state
// this reducer is responsible for
export default function(state, action) {
  return state
}
```

A sidebar on the right provides the following explanatory text:

- Action creator returns an action
- SELECTED, { 2 } is passed to all reducers
- Reducer
- If action type is BOOK_SELECTED, do nothing

if action type is `BOOK_SELECTED`
we will return `action.payload` (which is selected book)

```
// State argument is not application state, only the state
// this reducer is responsible for
export default function(state, action) {
  switch(action.type) {
    case 'BOOK_SELECTED':
      return action.payload;
  }
  return state;
}
```

The screenshot shows a browser window with a Udemy React-Redux tutorial. The main content is a code editor displaying `reducer_active_book.js`. The code defines a reducer that handles the `'BOOK_SELECTED'` action by returning the payload. A sidebar on the right contains handwritten-style notes explaining the concepts of actions and reducers.

at beginning if user has not clicked on any thing
state will undefined

redux does not allow us to return undefined

to avoid that we will default state argument to null at top
this is es6 syntax if argument is undefined it will set it null

other wise if action is any thing else we just return current state

The screenshot shows a Mac OS X desktop with a Chrome browser window open to a Udemy course page. The browser tabs include 'Secure https://walmart.udemy.com/react-redux/learn/v4/t/lecture/4284650?start=0'. The main content of the browser shows a slide titled 'Consuming Actions in Reducers' from 'Section 4, Lecture Y: List'. The Atom code editor is running in the background, showing the file 'reducer_active_book.js' with the following code:

```
// State argument is not application state, only the state
// this reducer is responsible for
export default function(state = null, action) {
  switch(action.type) {
    case 'BOOK_SELECTED':
      return action.payload;
  }
  return state;
}
```

A sidebar on the right of the Atom editor provides explanatory notes:

- Action creator returns an action
- SELECTED, <2>
- on automatically it to all reducers
- reducer
- tion, do nothing

now connect his reducer to combineReducers in index.js

import ActiveBook Reducer

```
import { combineReducers } from 'redux';
import BooksReducer from './reducer_books';
import ActiveBook from './reducer_active_book';

const rootReducer = combineReducers({
  books: BooksReducer,
  activeBook: ActiveBook
});

export default rootReducer;
```

and combine it as another piece of state activeBook activeBook: ActiveBook

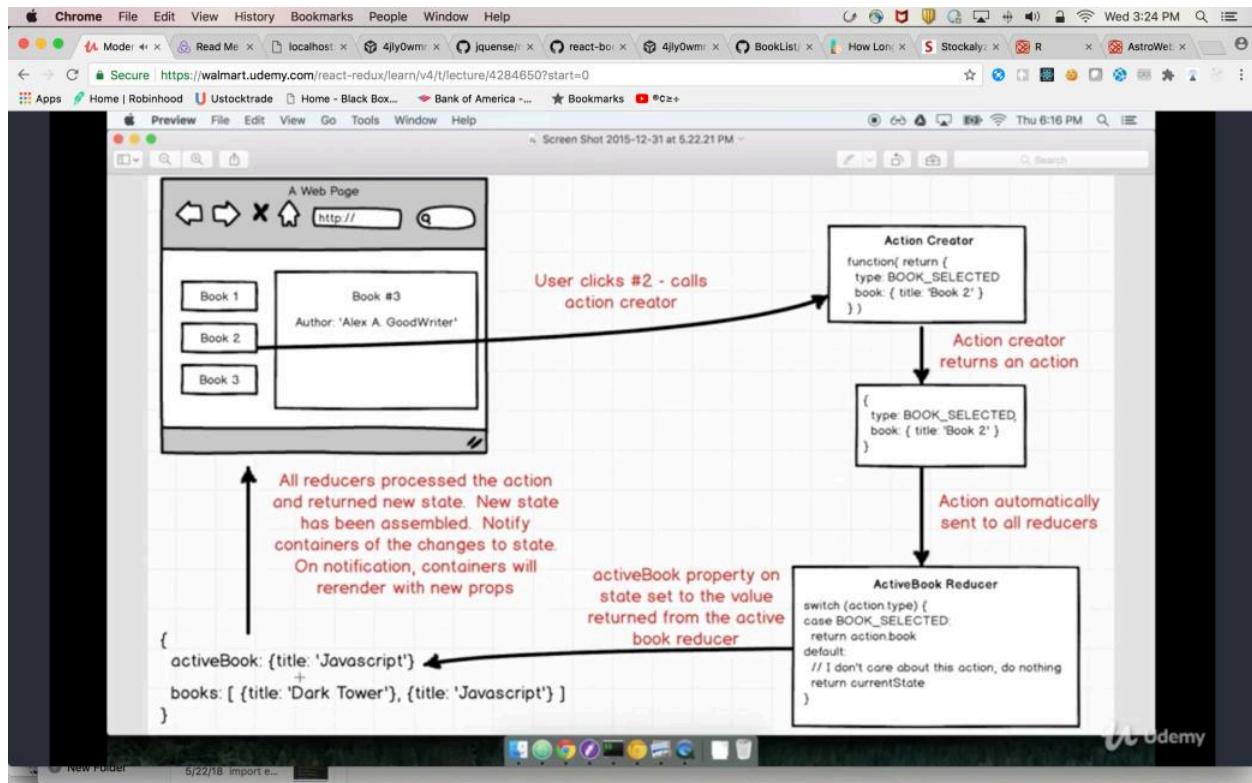
```
import { combineReducers } from 'redux';
import BooksReducer from './reducer_books';
import ActiveBook from './reducer_active_book';

const rootReducer = combineReducers({
  books: BooksReducer,
  activeBook: ActiveBook
});

export default rootReducer;
```

now any key to combineReducers object will ends up in global state

now **books** , **activeBook** will appear in global state



now we will create a new component in container folder book_detail.js (it is in container as it will connect to redux store)

```
import BookList from '../containers/book-list';
export default class App extends Component {
  render() {
    return (
      <div>
        | <BookList />
      </div>
    );
  }
}
```

import BookDetail
and call it in redner() method.

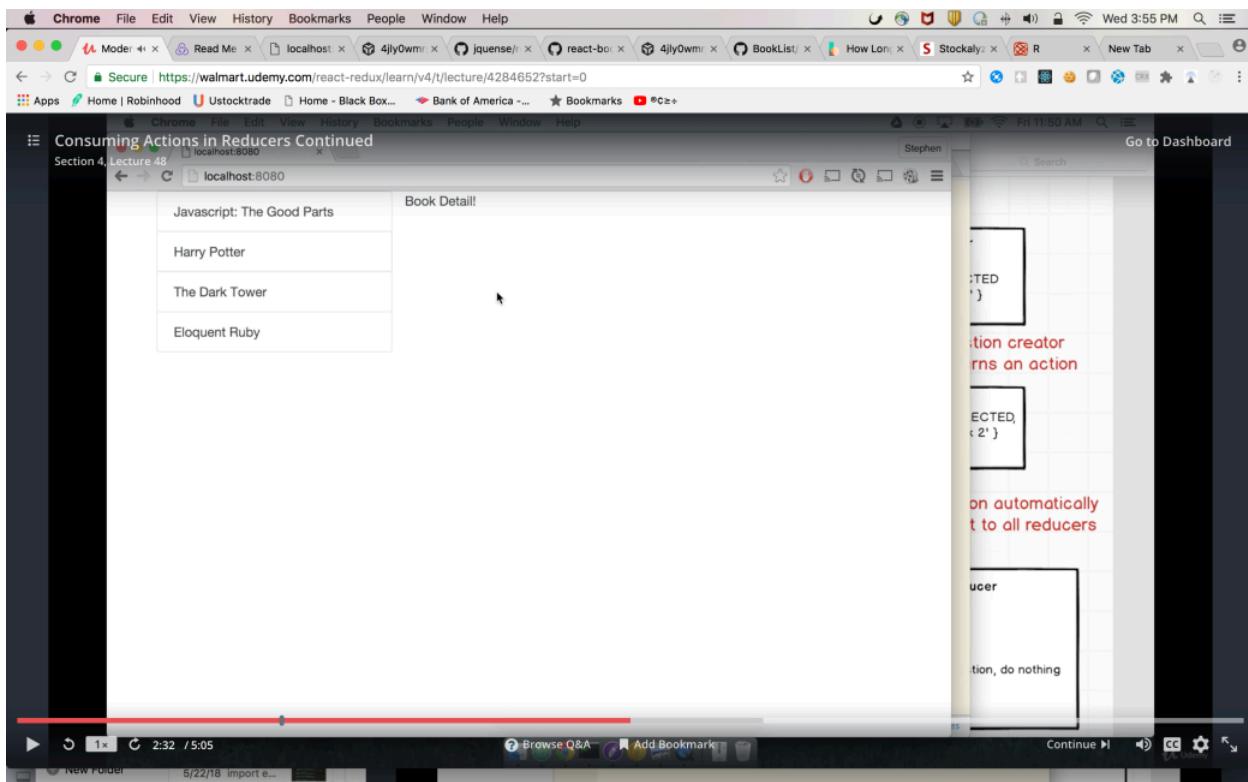
A screenshot of a Chrome browser window displaying a code editor for a React-Redux application. The code editor shows the file `src/containers/app.js` with the following content:

```
1 import React from 'react';
2 import { Component } from 'react';
3
4 import BookList from '../containers/book-list';
5 import BookDetail from '../containers/book-detail';
6
7 export default class App extends Component {
8   render() {
9     return (
10       <div>
11         <BookList />
12         <BookDetail />
13       </div>
14     );
15   }
16 }
17
```

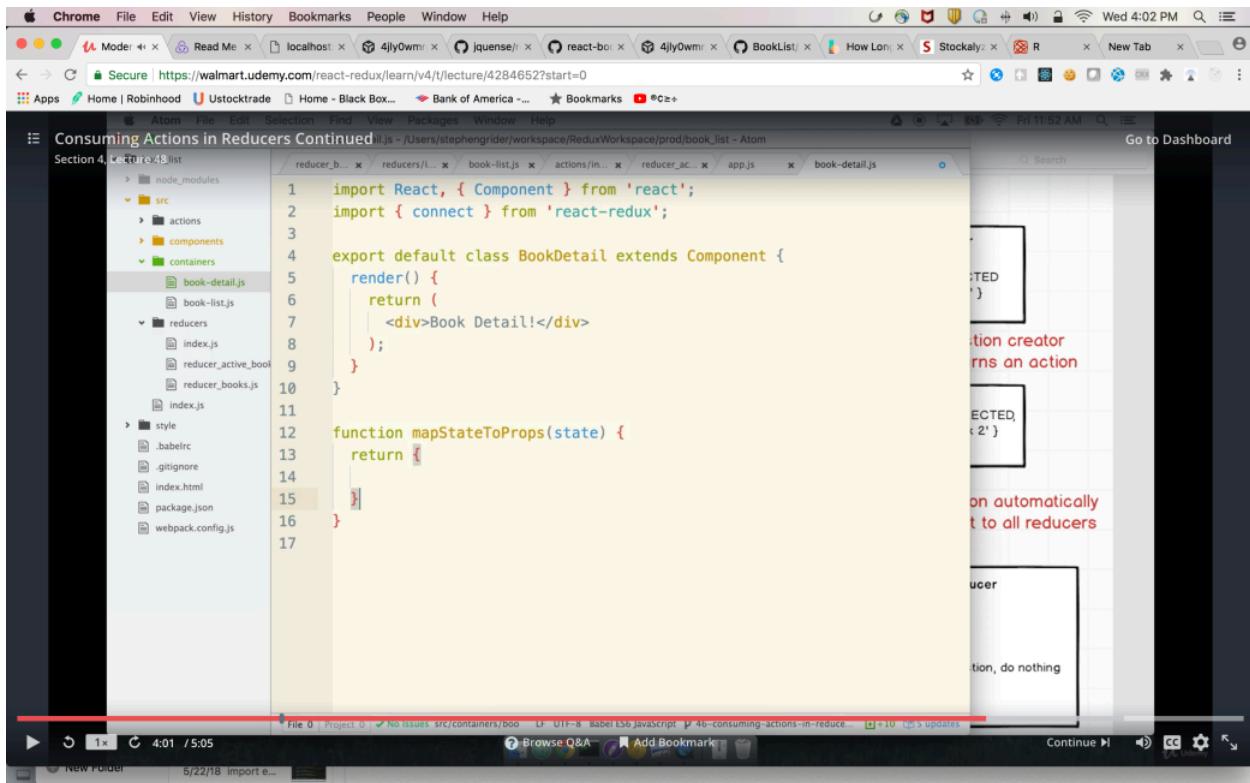
The sidebar on the right contains several handwritten-style notes:

- Top note: "Action creator returns an action"
- Second note: "SELECTED, '2' on automatically it to all reducers"
- Third note: "User action, do nothing"

now refresh the browser now components are showing up here



now we will import {connect} from 'react-redux'
and define mapStateToProps



```
import React, { Component } from 'react';
import { connect } from 'react-redux';

export default class BookDetail extends Component {
  render() {
    return (
      <div>Book Detail!</div>
    );
  }
}

function mapStateToProps(state) {
  return {}
}
```

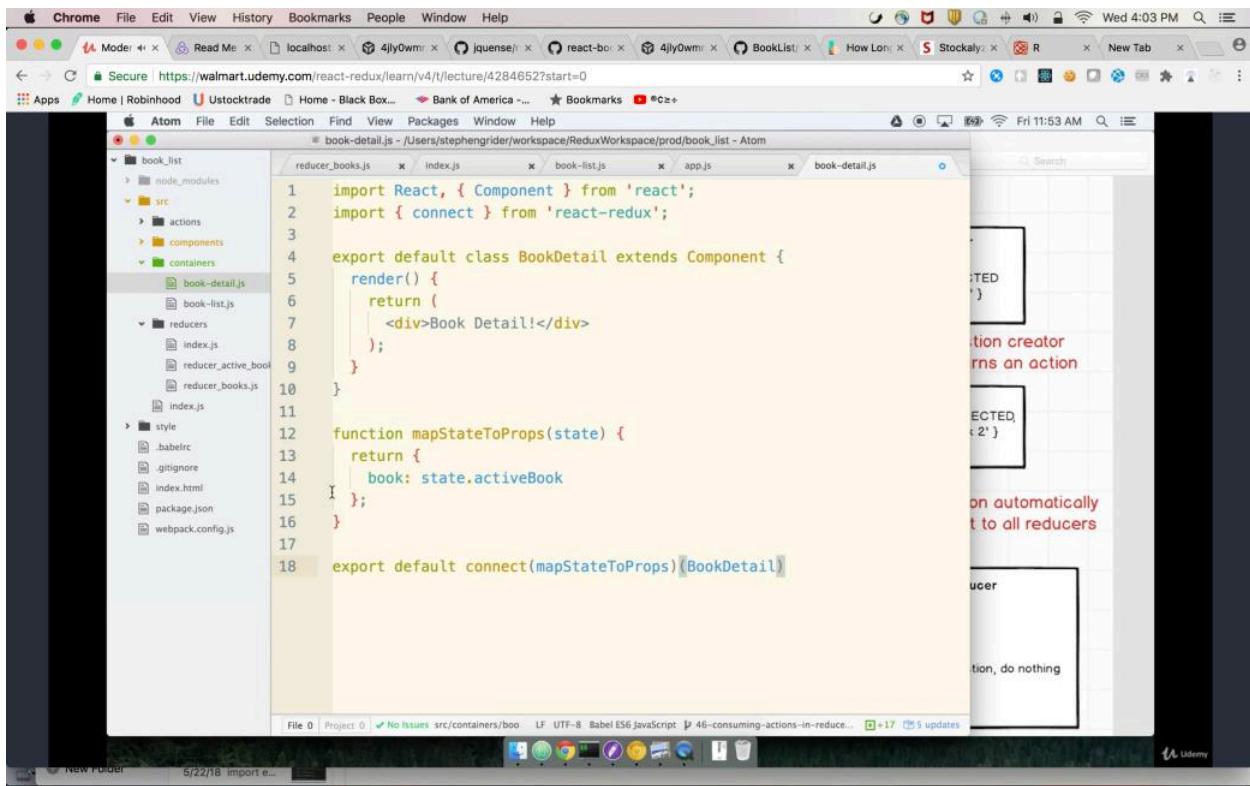
now we need to reference activeBook

```
import { combineReducers } from 'redux';
import BooksReducer from './reducer_books';
import ActiveBook from './reducer_active_book';

const rootReducer = combineReducers({
  books: BooksReducer,
  activeBook: ActiveBook
});

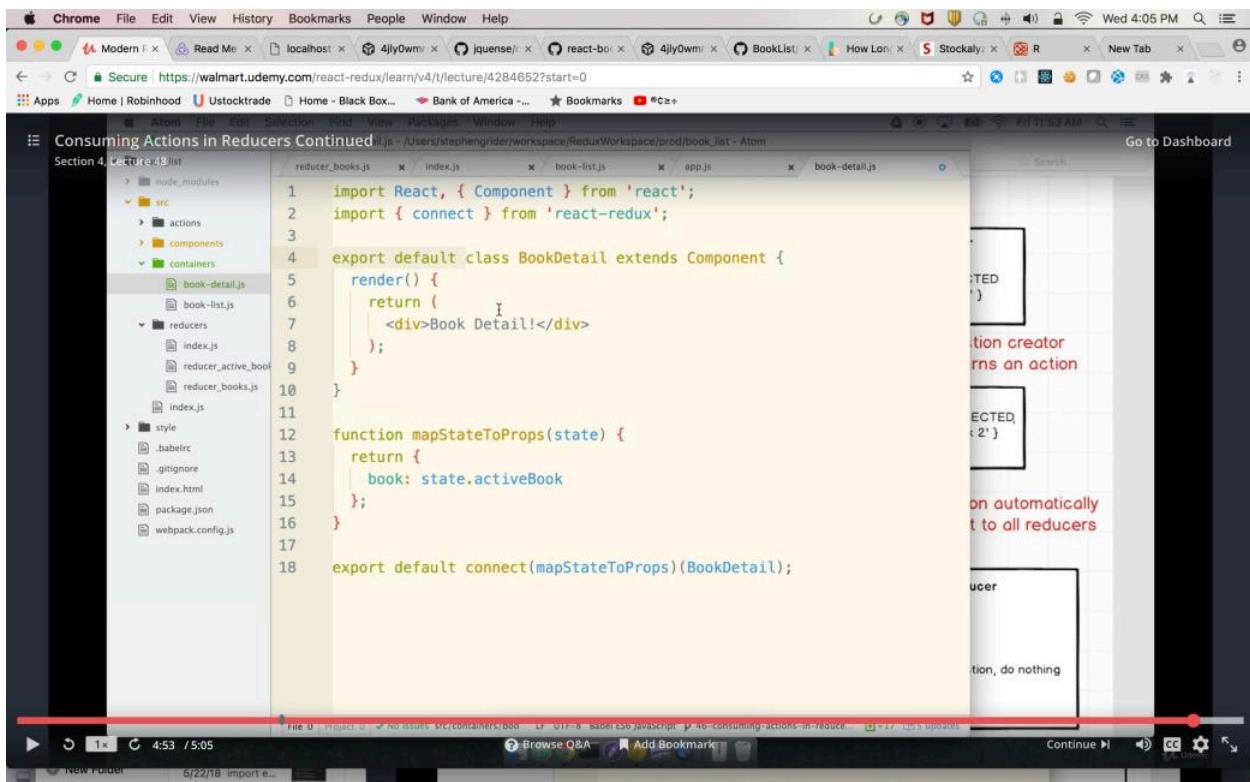
export default rootReducer;
```

we will return book : state.activeBook (activeBook set in state coming from combineReducers({}))
now we export default connect(mapStateToProps)(BookDetail)



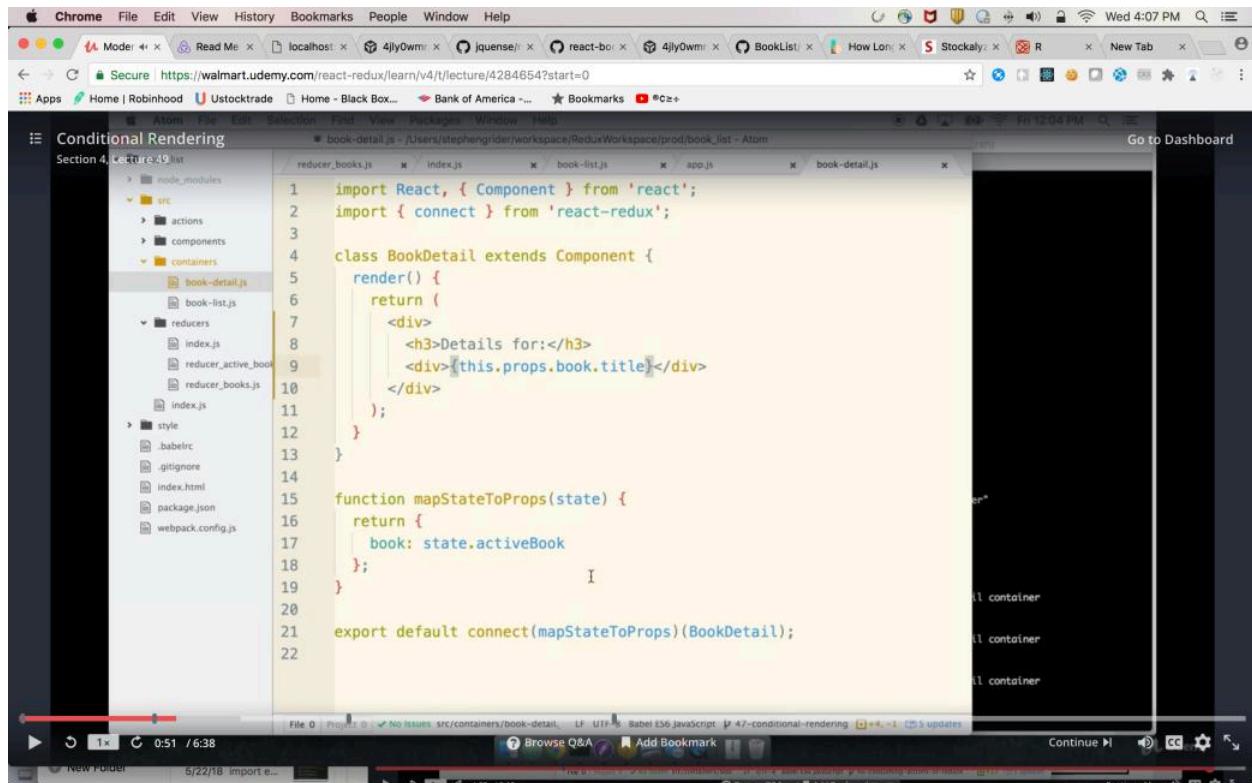
```
1 import React, { Component } from 'react';
2 import { connect } from 'react-redux';
3
4 export default class BookDetail extends Component {
5   render() {
6     return (
7       <div>Book Detail!</div>
8     );
9   }
10
11 function mapStateToProps(state) {
12   return {
13     book: state.activeBook
14   };
15 }
16
17
18 export default connect(mapStateToProps)(BookDetail);
```

remove export default from class name



```
1 import React, { Component } from 'react';
2 import { connect } from 'react-redux';
3
4 export default class BookDetail extends Component {
5   render() {
6     return (
7       I
8       <div>Book Detail!</div>
9     );
10 }
11
12 function mapStateToProps(state) {
13   return {
14     book: state.activeBook
15   };
16 }
17
18 export default connect(mapStateToProps)(BookDetail);
```

now we can reference this.props.book return from mapStateToProps
we will render the title



The screenshot shows the Atom code editor with the file `book-detail.js` open. The code defines a class `BookDetail` that extends `Component`. It contains a `render` method which returns a `<div>` element containing an `<h3>` and a `<div>` with the prop `this.props.book.title`. Below the `render` method is a `mapStateToProps` function that returns an object with a `book` key set to `state.activeBook`. The code ends with an `export default connect(mapStateToProps)(BookDetail);` statement. The browser tab at the top shows a Udemy React-Redux course page.

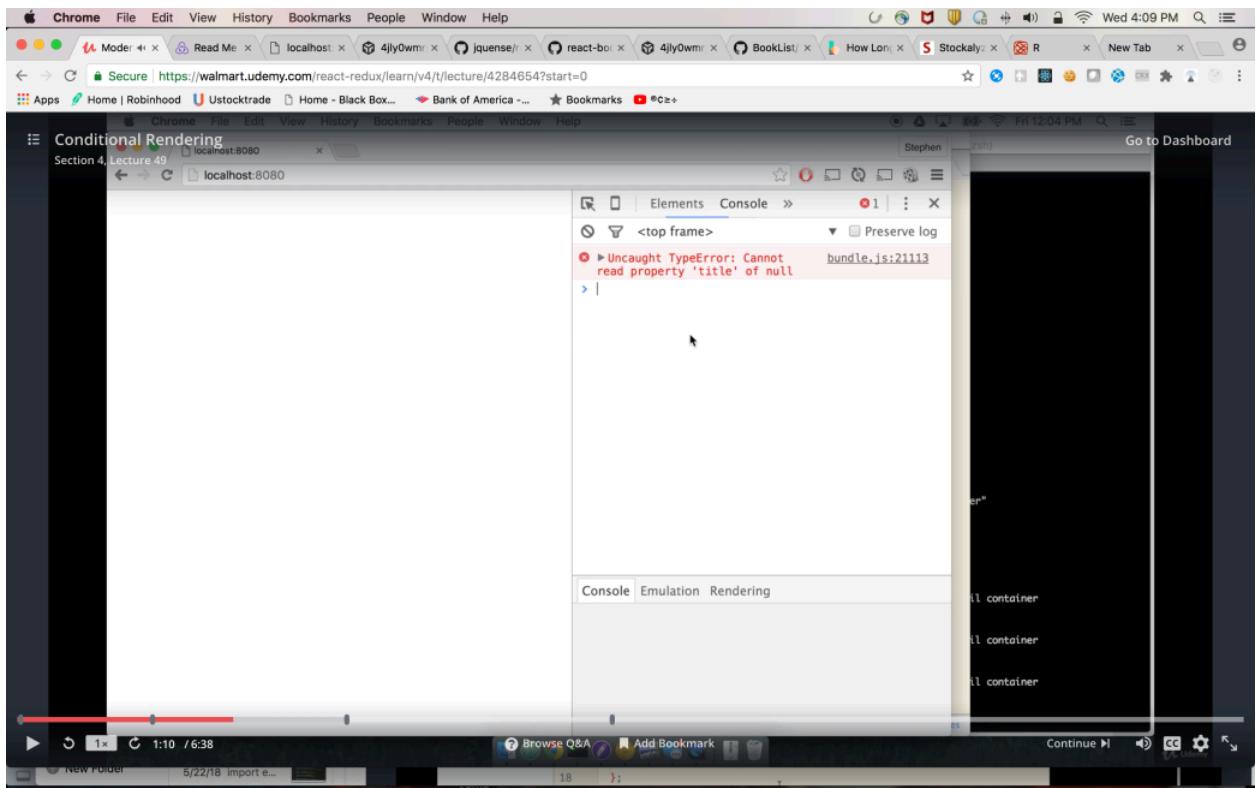
```
import React, { Component } from 'react';
import { connect } from 'react-redux';

class BookDetail extends Component {
  render() {
    return (
      <div>
        <h3>Details for:</h3>
        <div>{this.props.book.title}</div>
      </div>
    );
  }

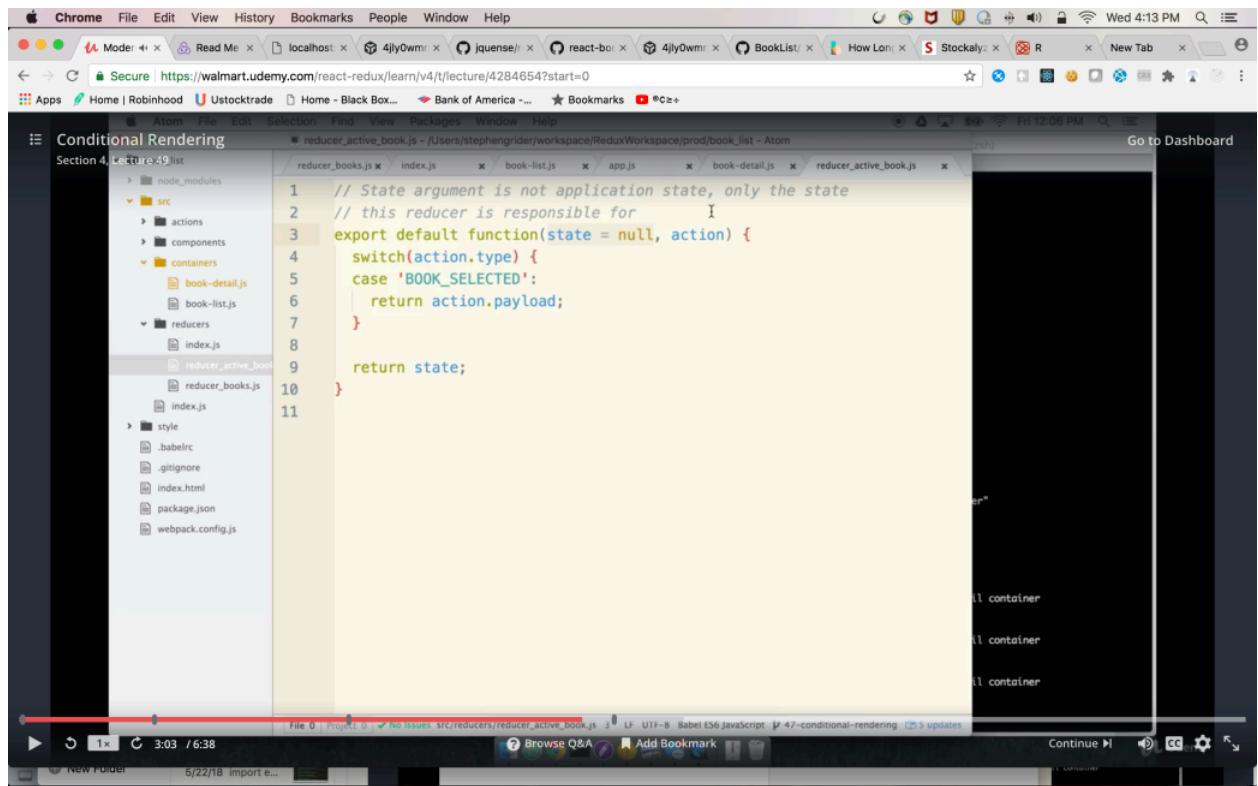
  function mapStateToProps(state) {
    return {
      book: state.activeBook
    };
  }
}

export default connect(mapStateToProps)(BookDetail);
```

now we refresh the browser
and we get the error cannot read property 'title' of null



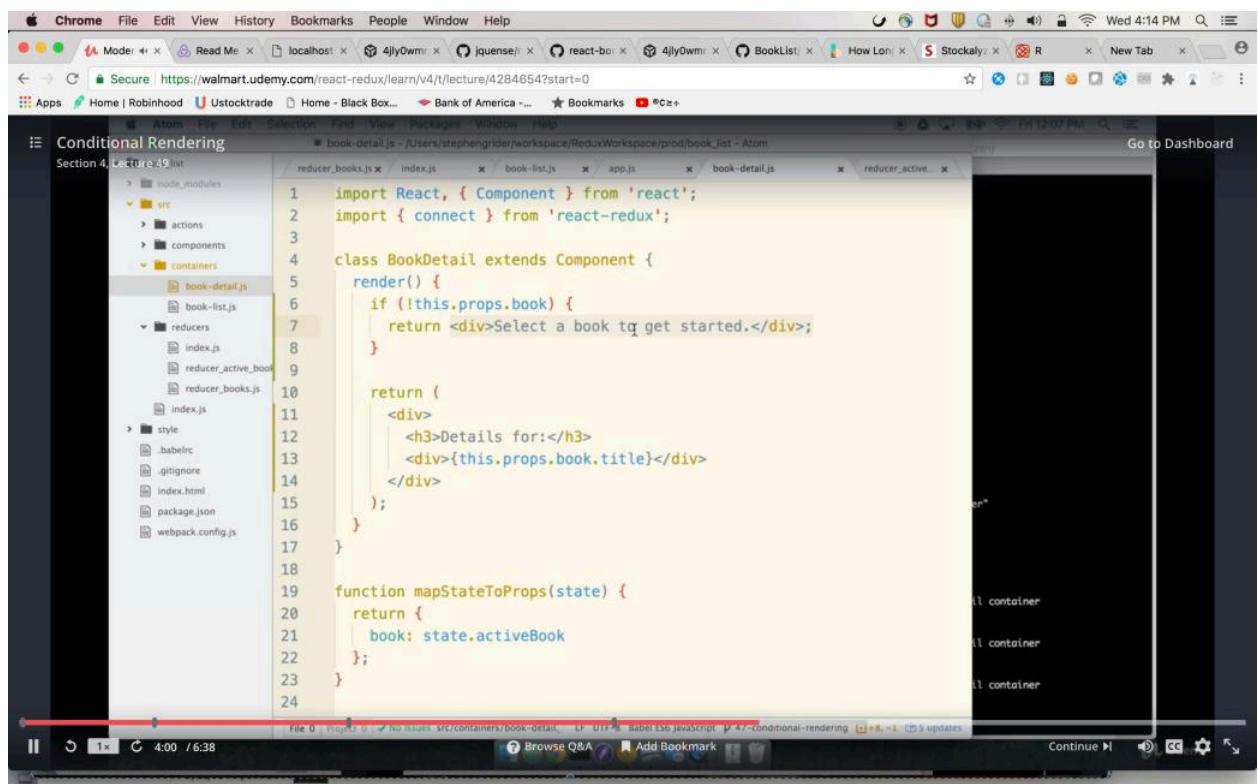
because at beginning when app is loaded there is action and state is default to null
that is by this.props.book is null



```
// State argument is not application state, only the state
// this reducer is responsible for
export default function(state = null, action) {
  switch(action.type) {
    case 'BOOK_SELECTED':
      return action.payload;
  }

  return state;
}
```

we will put a condition when `this.props.book` is null
and return message earlier

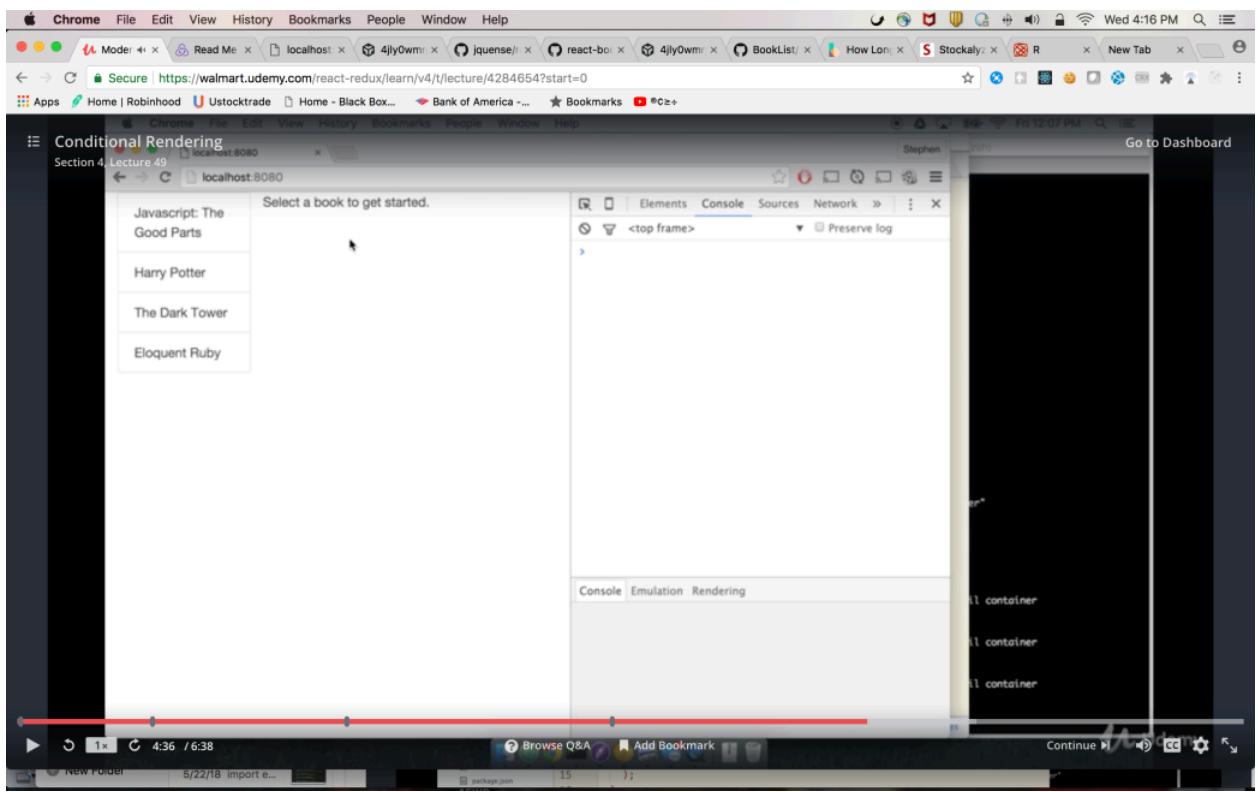


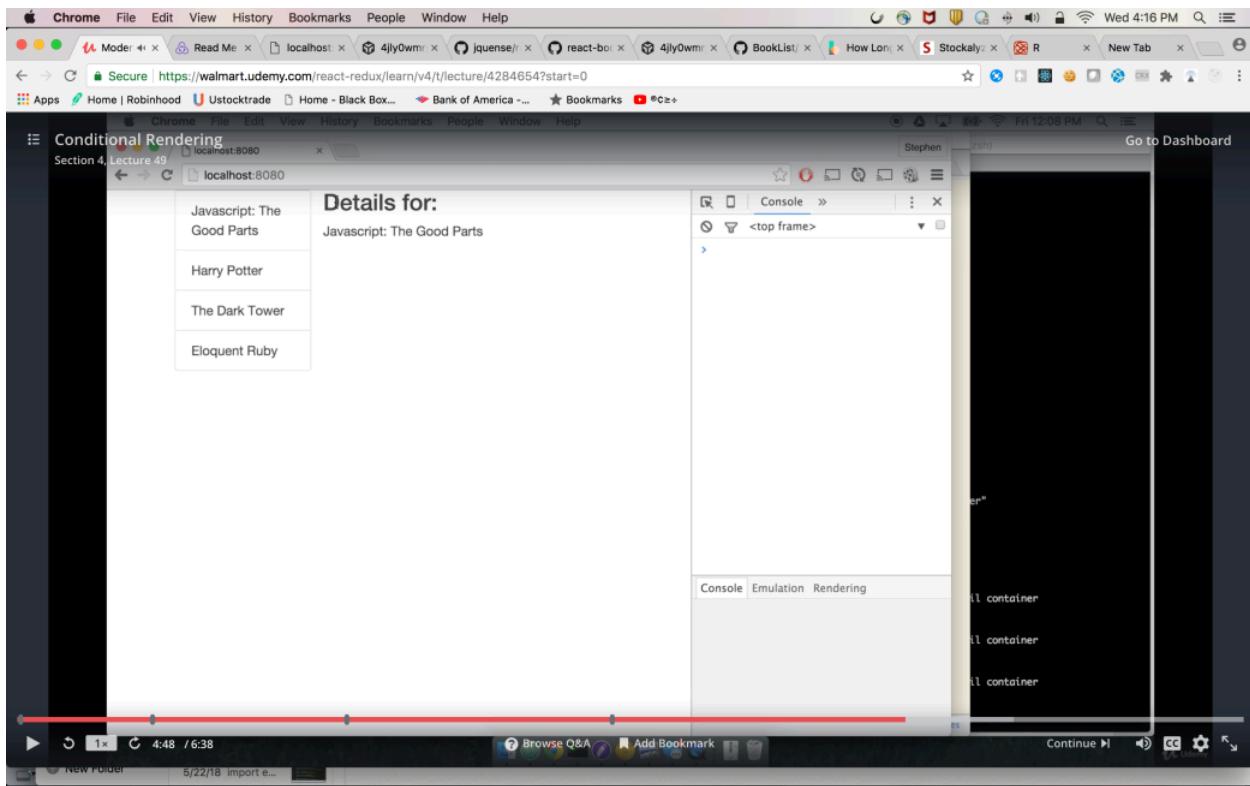
```
import React, { Component } from 'react';
import { connect } from 'react-redux';

class BookDetail extends Component {
  render() {
    if (!this.props.book) {
      return <div>Select a book to get started.</div>;
    }

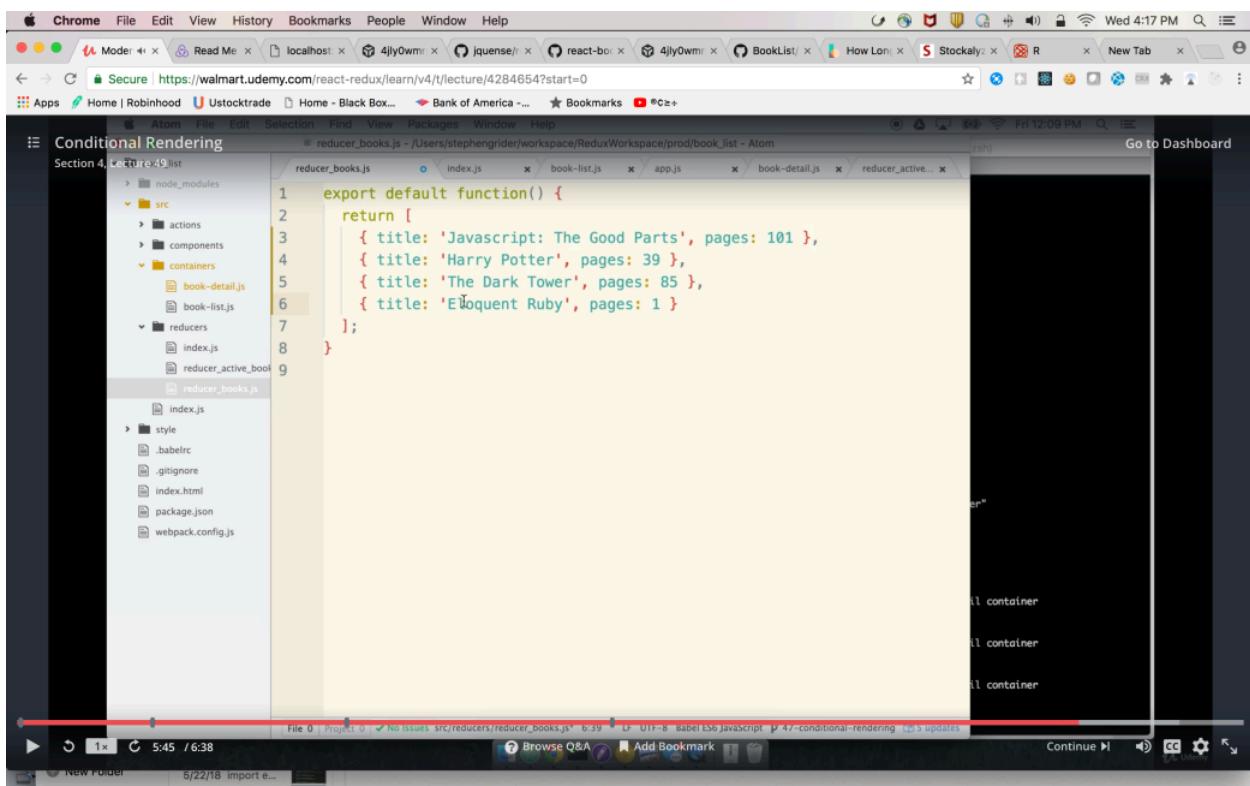
    return (
      <div>
        <h3>Details for:</h3>
        <div>{this.props.book.title}</div>
      </div>
    );
  }
}

function mapStateToProps(state) {
  return {
    book: state.activeBook
  };
}
```





now add number of pages in reducer

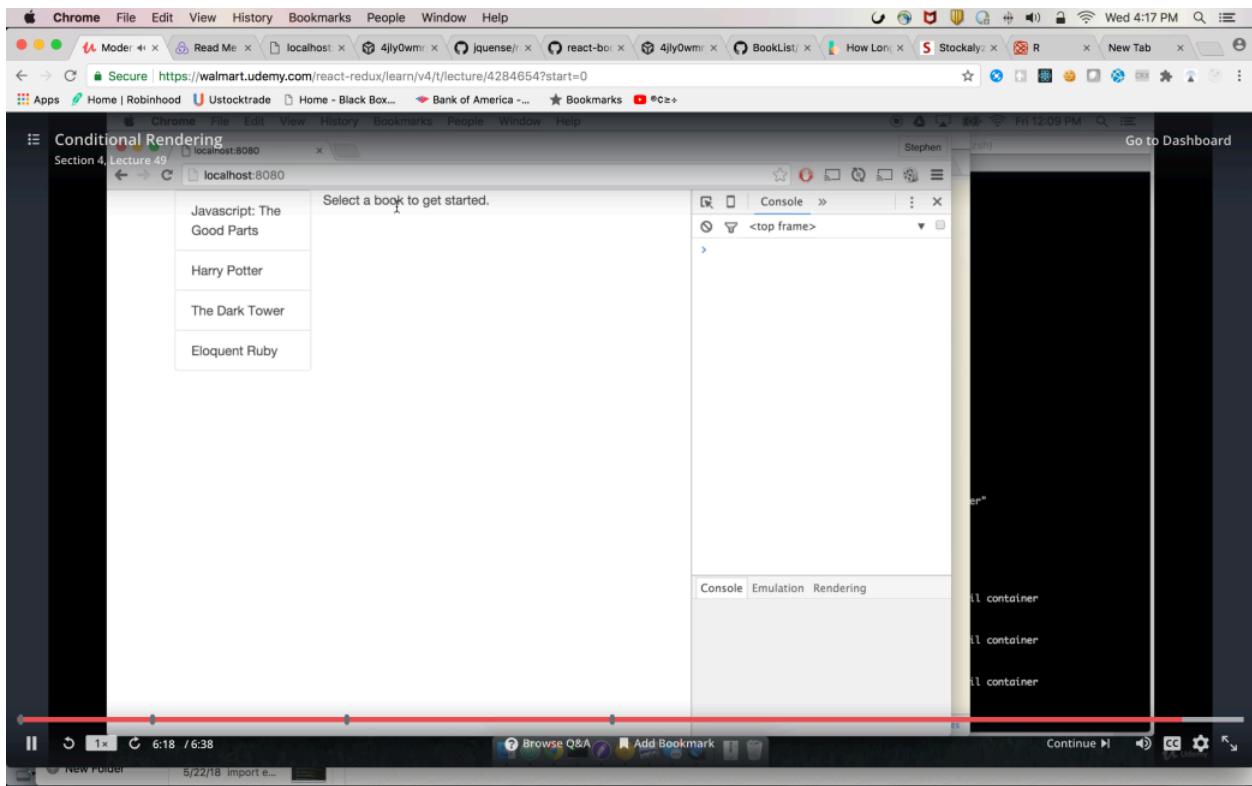


number of pages in bookDetail component

The screenshot shows a browser window with multiple tabs open. The active tab is a code editor for a file named `book-detail.js`. The code is as follows:

```
1 import React, { Component } from 'react';
2 import { connect } from 'react-redux';
3
4 class BookDetail extends Component {
5   render() {
6     if (!this.props.book) {
7       return <div>Select a book to get started.</div>;
8     }
9
10    return (
11      <div>
12        <h3>Details for:</h3>
13        <div>Title: {this.props.book.title}</div>
14        <div>Pages: {this.props.book.pages}</div>
15      </div>
16    );
17  }
18}
19
20 function mapStateToProps(state) {
21   return {
22     book: state.activeBook
23   };
24}
```

The code uses conditional rendering to check if a book is selected. If no book is selected, it displays a message. Otherwise, it shows the title and the number of pages.



click in li will result in

