

JavaScript History

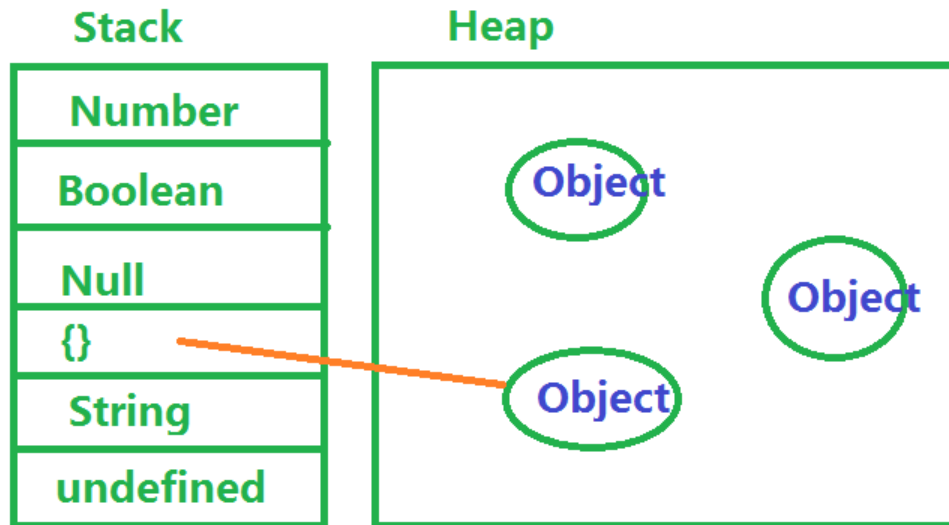
- Created in May 1995 by [Brendan Eich](#)
- ECMAScript 2 ---1998
- ECMAScript 3 ---1999
- **ECMAScript 5 ---2009**
- **ECMAScript 6 ---2015**
- ECMAScript 7

<https://github.com/tc39/ecma262>

<https://developer.mozilla.org/en-US/>

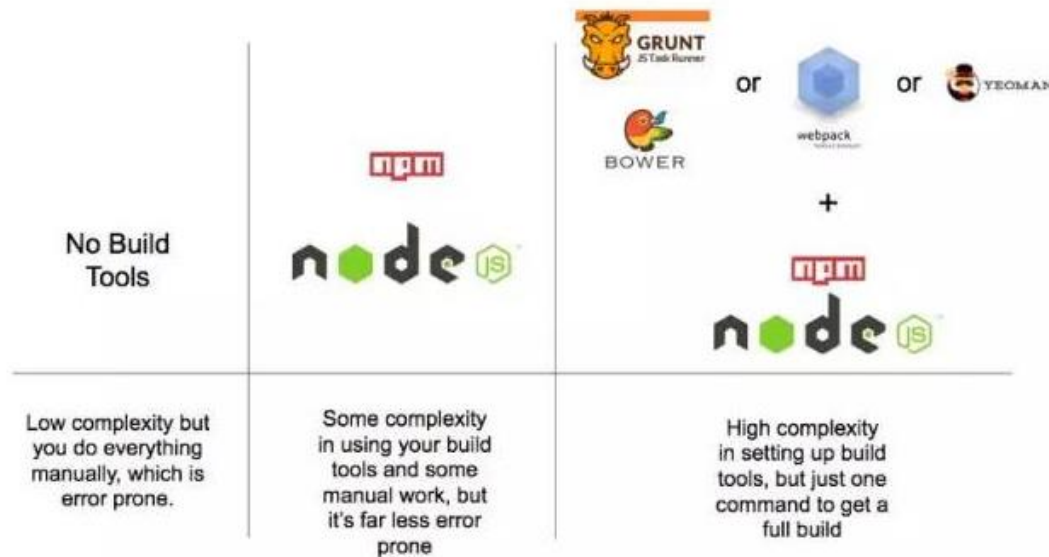
JS data types: By Val, By Ref

- Primary data types: Number, String, Null, Boolean, undefined
- Referenced Type: Array, Function, Object



Front end build tools

- Installation and help do work



ES6 features in React

- Arrow function
- const and let (no hoisting)
- Function default value
- Object.assign()
- Destructuring and Spread operator
- Import, Export, Class
- Backtick, especially in Angular

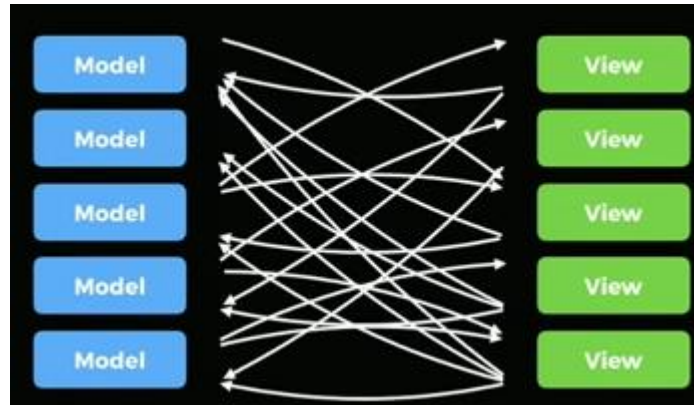
*remove global object definition

**demo some important code related to immutable application such as spread operation, destructure, deep copy of Object(Babel REPL)

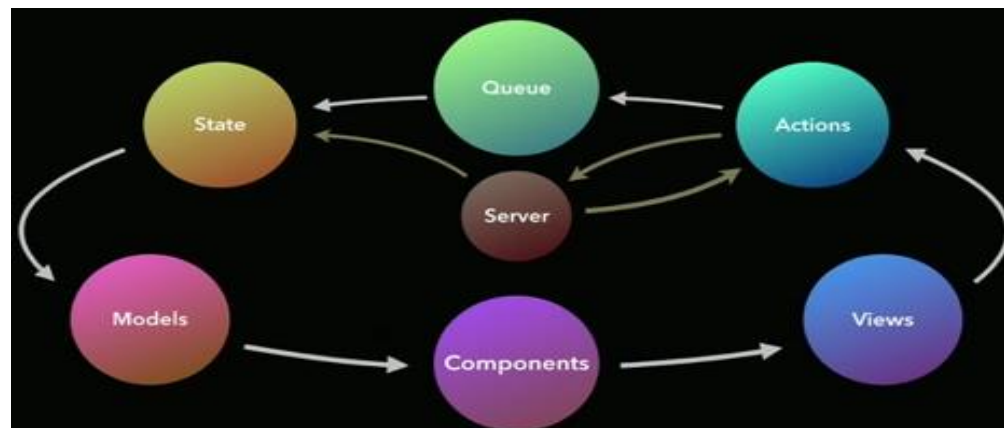
- React Syntax ---JSX (Babel REPL)

React App Architecture

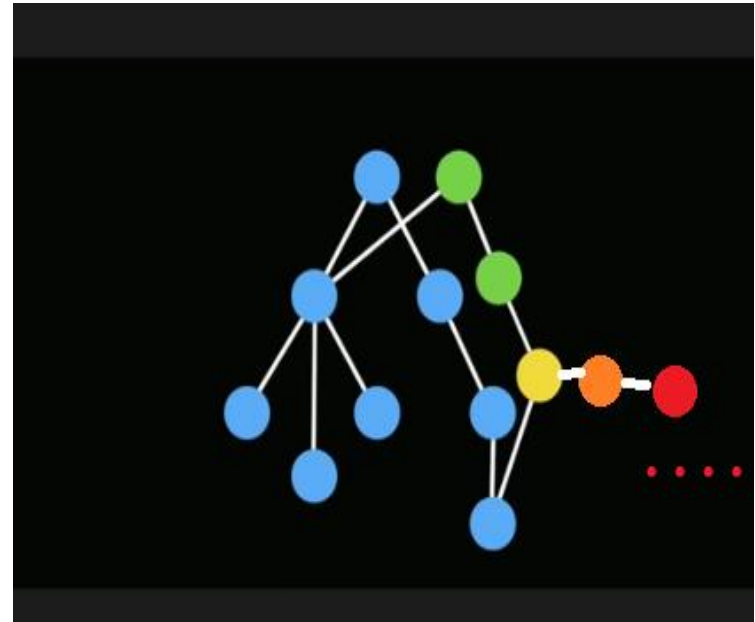
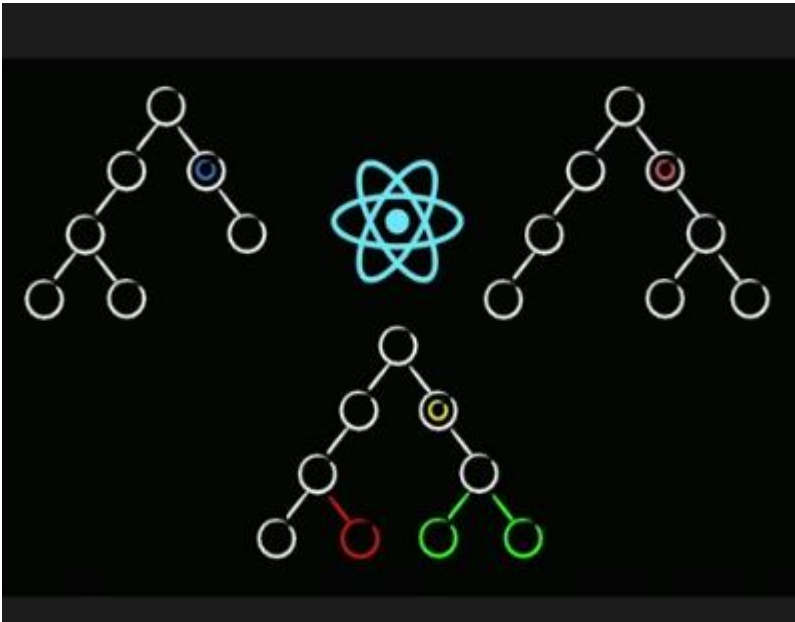
- MVC left with lots of challenge things not resolved



- Immutable App Architecture



React – Immutable Application

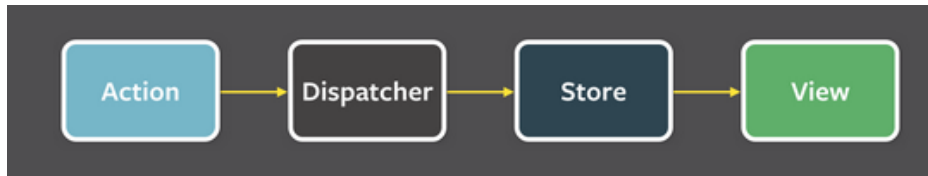


*** Lee Byron presentation on “Immutable User Interface”**

****Demo elements is being copied many many.....**

React App Architecture Implementation

- Facebook provide Flux solution:



- Most Popular Solutions:
 - MobX observable: state is mutable
 - Redux functional: state is immutable



React Introduction--Component

Search...

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

- <https://facebook.github.io/react/docs/thinking-in-react.html>

React Introduction—Key concepts

- Key concept: View is the output of State
- Essence: Using JS function to create the GUI
- Important features: `components(this.props, this.state, this.refs)`, virtual DOM, immutable app architecture
- Life Cycle: `componentWillMount()`, **`componentDidMount()`***, **`componentWillUpdate()`***, `componentDidUpdate()`, `componentWillUnmount()`, `componentWillReceiveProps()`
- JSX, Babel transpiler

* NEVER, NEVER USE `setState()`

React Component

```
import React,{Component} from 'react'  
import MessageComposer from './MessageComposer';  
import MessageListItem from './MessageListItem'
```

library and component import
statements

.....

```
export default class Chat extends Component {  
  render(){  
    <div className="main">  
      <header>{activeChannel}</header>  
      {PrivateMessageModal}  
      {MessageListItem}/>  
      {MessageComposer />  
    </div>  
  }  
}
```

Make component available for
import

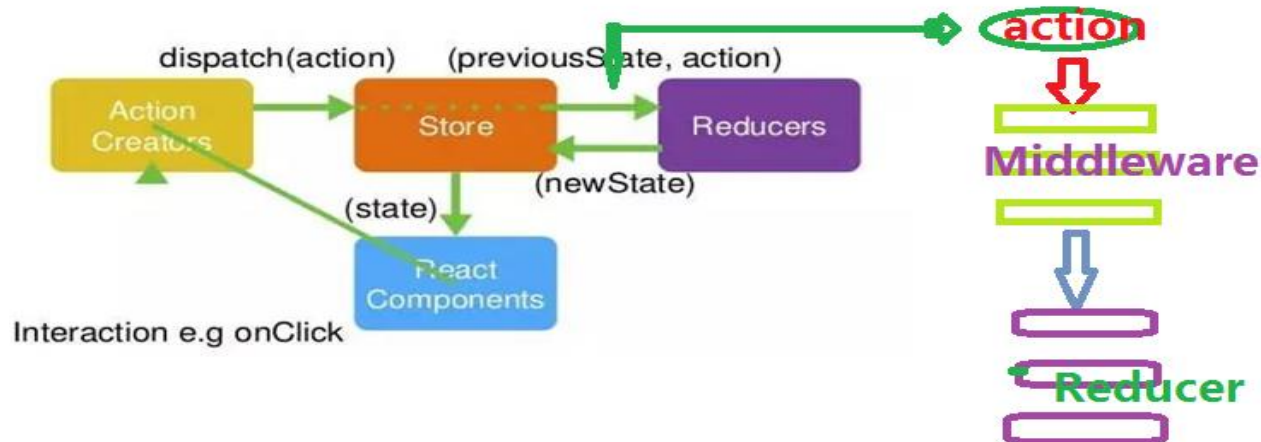
Component body

Redux---State Container

- Don't need Redux if your app is simple, no users' interaction, no server interaction
- Web app is about state, view and state one to one relationship, all states in store object.

```
const store=createStore(reducer)
```

Redux Flow



Redux---State Container

- View: 1:1 with state
- Action: carry the data
- Reducer : pure function, only for calculate state
- Middleware: the place for add functions,
example: `applyMiddleware(thunk, promise, logger);` thunk
middleware make `store.dispatch` can accept function as parameter

React-Redux Introduction

- Component separate into 2 parts: UI—a pure UI, container – data and logic

- `connect()`: create container from UI,

```
const TodoList=connect(  
  mapStateToProps,  
  mapDispatchToProps  
)  
(ToDoList)
```

- **Provider**: help pass state object to UI

```
<Provider store={store}>  
  <App/>  
</Provider>
```

- **React-Router**

```
const Root = ({ store }) => (  
  <Provider store={store}>  
    <Router>  
      <Route path="/" component={App} />  
    </Router>  
  </Provider>  
)
```

React Project Demo

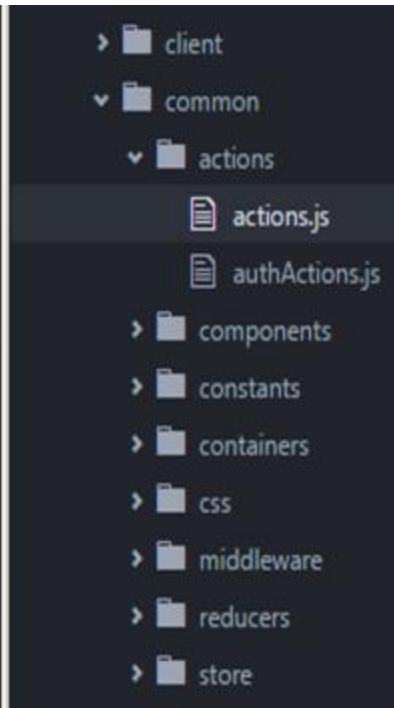
We are going to learn a pretty solid example to demo: React-Redux-SocketIO project--- An open source ChatRoom project using MongoDB as database:

<https://github.com/raineroviir/react-redux-socketio-chat>

- Project Overview and folder structure
- Send/Create message process
- Receive/Dispatch message process

CreateMessage I

```
6 // NOTE:Chat actions
7
8 function addMessage(message) {
9   return {
10     type: types.ADD_MESSAGE,
11     message
12   };
13 }
14
15 export function receiveRawMessage(message) {
16   return {
17     type: types.RECEIVE_MESSAGE,
18     message:message
19   };
20 }
21
22 export function receiveRawChannel(channel) {
23   return {
24     type: types.RECEIVE_CHANNEL,
```



```
155 }
156
157 export function createMessage(message) {
158   return dispatch => {
159     dispatch(addMessage(message))
160     return fetch('/api/newmessage', {
161       method: 'post',
162       headers: {
163         'Content-Type': 'application/json'
164       },
165       body: JSON.stringify(message))
166       .catch(error => {throw error});
167   }
168 }
```

- src/.../ actions/actions.js

CreateMessage II

The image displays a code editor with three panes illustrating the implementation of a chat application's message saving functionality.

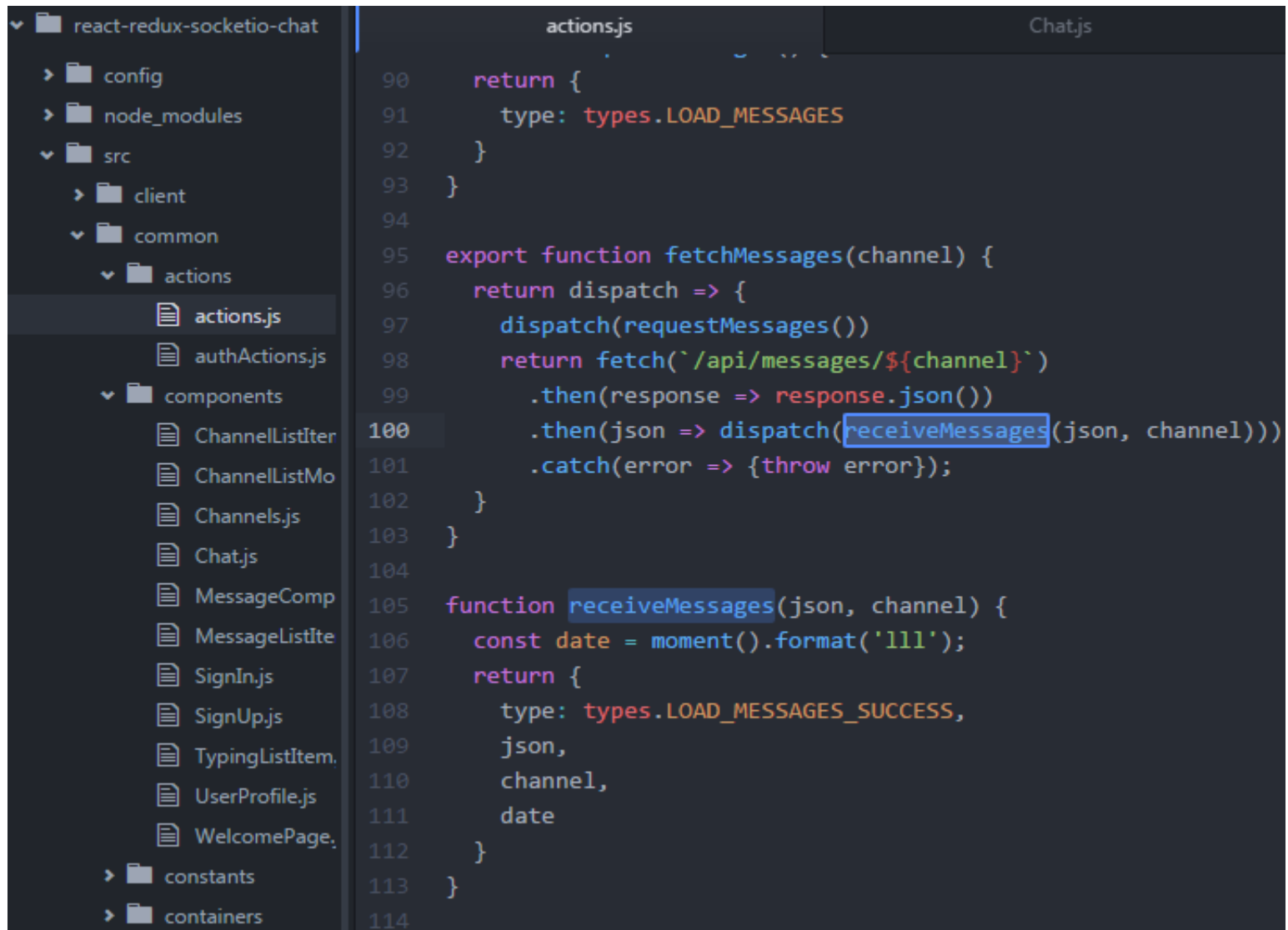
Left Pane (MessageComposer.js): Shows the `render()` method. A text input field is highlighted with a yellow box. The input field has the following props: `style={{ zIndex: '52', left: '21.1rem', right: '1rem', width: '100%', flexShrink: '0', order: '2', marginTop: '0.5em' }}`, `type="text"`, `name="message"`, `ref="messageComposer"`, `autoFocus="true"`, `placeholder="Type here to chat!"`, `value={this.state.text}`, `onChange={::this.handleChange}`, and `onKeyDown={::this.handleSubmit}`.

Middle Pane (Chat.js): Shows the `handleSave(newMessage)` function. The function calls `dispatch(actions.createMessage(newMessage))` to save the message. The function is highlighted with a yellow box.

Right Pane: Shows a search for `handleSave` in the codebase. The search results show that `handleSave` is defined in `MessageComposer.js` and is called from `Chat.js` via `onSave={::this.handleSave}`.

- `src/.../ components/Chats.js, MessageComposer.js`

Receive Message I



The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the directory structure of a project named 'react-redux-socketio-chat'. The 'actions' folder is expanded, showing 'actions.js' as the selected file. The code editor displays the content of 'actions.js', which includes a 'return' statement for 'types.LOAD_MESSAGES', an export function 'fetchMessages' that dispatches 'requestMessages' and fetches data from an API, and a function 'receiveMessages' that formats the date and returns the message data.

```
90   return {
91     type: types.LOAD_MESSAGES
92   }
93 }
94
95 export function fetchMessages(channel) {
96   return dispatch => {
97     dispatch(requestMessages())
98     return fetch(`/api/messages/${channel}`)
99       .then(response => response.json())
100      .then(json => dispatch(receiveMessages(json, channel)))
101      .catch(error => {throw error});
102   }
103 }
104
105 function receiveMessages(json, channel) {
106   const date = moment().format('lll');
107   return {
108     type: types.LOAD_MESSAGES_SUCCESS,
109     json,
110     channel,
111     date
112   }
113 }
114
```

- src/.../ actions/actions.js

Receive Message II

The image displays a code editor with three panes illustrating the implementation of a chat application.

Left Pane (Chat.js):

```
handleSignOut() {
  const { dispatch } = this.props;
  dispatch(authActions.signOut());
}

changeActiveChannel(channel) {
  const { socket, activeChannel, dispatch } = this.props;
  socket.emit('leave channel', activeChannel);
  socket.emit('join channel', channel);
  dispatch(actions.changeChannel(channel));
  dispatch(actions.fetchMessages(channel.name));
}

handleClickOnUser(user) {
  this.setState({ privateChannelModal: true, targetedUser: user });
}

closePrivateChannelModal(event) {
  event.preventDefault();
  this.setState({ privateChannelModal: false });
}

handleSendDirectMessage() {
  const { dispatch, socket, channels, user } = this.props;
  const doesPrivateChannelExist = channels.filter(item => item.name === `${this.state.targetedUser.username} - ${user.username}`);
  if (doesPrivateChannelExist.length > 0) {
    const newChannel = {
      name: `${this.state.targetedUser.username} - ${user.username}`,
      id: Date.now(),
      private: true,
      between: [this.state.targetedUser.username, user.username],
    };
    dispatch(actions.createChannel(newChannel));
    this.changeActiveChannel(newChannel);
    socket.emit('new private channel', this.state.targetedUser.username);
  } else {
    this.changeActiveChannel(doesPrivateChannelExist[0].name);
  }
  this.setState({ privateChannelModal: false, targetedUser: null });
}

render() {
  const { messages, socket, channels, activeChannel, filteredMessages } = this.props;
  const username = this.props.user.username;
  const dropdownMenu = (
    <div style={{ width: '21rem', top: '0', align: 'center', border: '1px solid #ccc', padding: '5px' }}>
      <DropdownButton key={1} style={{ width: '21rem' }}>
        <MenuItem style={{ width: '21rem' }} eventKey={1}>New Channel</MenuItem>
      </DropdownButton>
    </div>
  );
}
```

Middle Pane (File Structure):

- react-redux-socketio-chat
 - config
 - node_modules
 - src
 - client
 - common
 - actions
 - actions.js
 - authActions.js
 - components
 - ChannelListIter
 - ChannelListMo
 - Channels.js
 - Chat.js
 - MessageComp
 - MessageListIt
 - SignIn.js
 - SignUp.js
 - TypingListIt
 - UserProfile.js
 - WelcomePage
 - constants
 - containers
 - css
 - middleware
 - reducers
 - store
 - routes.js
 - server
 - static
 - .babelrc
 - .eslintignore

Right Pane (actions.js):

```
const mobileNav = (
  <Navbar fixedTop style={{ background: '#337ab7', color: 'white' }}>
    <span style={{ fontSize: '2em' }}>{username}</span>
    <Navbar.Toggle />
    <Navbar.Collapse style={{ maxHeight: '100%' }}>
      <Button bsStyle="primary" onSelect={this.handleSignOut}>Sign out</Button>
      <section style={{ order: '2', marginTop: '1.5em' }}>
        <Channels socket={socket} onClick={this.changeActiveChannel} channels={channels} messages={messages} dispatch={dispatch} />
      </section>
    </Navbar.Collapse>
  </Navbar>
);

const bigNav = (
  <div className="nav">
    <dropdownMenu>
      <section style={{ order: '2', marginTop: '1.5em' }}>
        <Channels socket={socket} onClick={this.changeActiveChannel} channels={channels} messages={messages} dispatch={dispatch} />
      </section>
    </div>
  );

return (
  <div style={{ margin: '0', padding: '0', height: '100%', width: '100%', display: 'flex', flex-direction: 'column' }}>
    {screenWidth < 500 ? mobileNav : bigNav}
    <div className="main">
      <header style={{ background: 'white', color: 'black', flexGrow: '0', order: '0', padding: '5px 0 0 0' }}>
        <div>
          {activeChannel}
        </div>
      </header>
    </div>
  </div>
);
```

Action-Middleware-Reducer

(use receive message as an example, slide 11)

- `///
function receiveMessages(json, channel){ }
export function fetchMessages(channel) { } <--- isomorphic-fetch
http://localhost:3000/api/messages/Lobby`
- `///
LOAD_MESSAGES_SUCCESS`
- `///
promiseMiddleware`
- `message_routes.js: http://localhost:3000/api/channels;
http://localhost:3000/api/messages/Lobby`
- `///
configureStore.dev.js: export configureStore
../src/client/index.js <--- entry point; root element named: "react"
Once "store" get new data, through "Provider", it will update GUI by "state"`

How to start React Project

- Method I

1. `npm install -g create-react-app`
2. `create-react-app name_of_project`
3. `cd name_of_project`
4. `npm start`

- Method II

1. Using a start boilerplate Projection
2. <https://github.com/StephenGrider/ReduxSimpleStarter>

Q & A

- ?