REACT & REDUX

I. Installation:

- Install NodeJS and npm(node package manager) tools:

https://nodejs.org/en/download/

II. Big words:

React:

As a javascript library that is used to produce HTML that is shown to user in a web browser. When we write React code, we are writing individual components or views

Component:

React works with only 1 component, or only 1 component to store all other nested components. This big component like a background of a web to store other elements. This characteristic is similar with AngularJS – working with 1 object variable.

All nested components (.js files) are put in “src\components” folder

Is a collection of Javascript functions that produce HTML

Each component treat as <div>. React uses nested components like container in HTML

Component can be reusable as a function or method

Dependencies:

React use npm to inject dependencies to the project that includes react which in "node\_modules" which declared in "package.json" (similar as Gemfile in ROR)

III. Basic:

Clone the project from github. It already has package.json file, so we run “npm install” to install dependencies which declared in package.json. Then “npm start” to start local server at port 8080.

*import React from ‘react’;*

*import ReactDOM from ‘react-dom’;*

*import SearchBar from ‘./components/search\_bar’;*

*const API\_KEY = ‘…’;*

*const App = () => {*

*return (*

*<div>*

*<SearchBar /> // use component/JS file as a custom tag type*

*</div>*

*);*

*}*

*ReactDOM.render(<App />, document.querySelector(‘.container’));* // Render to DOM, need to import ReactDOM from ‘react-dom’. Passing <App /> as instance, if we pass “App” only, it will be passing a class and can’t render in DOM.

“App” is the name of a component (js function)

import React from ‘react’; //We need to import React so that we can use JSX format in JS (<div /> instead of React.createElement(“div”, null))

Create API key and install youtube api via npm:

* Go to google APIs to create an api with key
* npm install --save youtube-api-search //--save mean save info to package.json too

We can create components (.js files) and import them, but we need “export default method\_name” at the end of that file. To import that file, we need to show exactly path, not just the name as dependencies which injected by npm

import SearchBar from ‘./components/sear\_bar’;

“./” same lvl as the file importing

“search\_bar” is search\_bar.js file that content “export default method\_name”

Path: src\components\search\_bar.js

Functional component:

*import React from ‘react’;*

*const SearchBar = () => { // this is feature in ES6: function() ~ () =>*

*return <input />;*

*};*

*export default SearchBar*

Class-Based component:

Declare component as a class in Java // rewrite below

*import React from ‘react’; // import React, {Component} from ‘react’;*

*class SearchBar extends React.Component { // …extends Component*

*render() {*

*return <input />;*

*}*

*}*

*export default SearchBar;*

Handling Event in React has 2 steps:

* Declare event handler
* Pass handler to the element that we want to monitor for the event

*render() {*

*return <input onChange={this.onInputChange} />*

*}*

*onInputChange(event) {*

*console.log(event.target.value);*

*}*

Equivalent:

*render() {*

*return (*

*<input*

*value = {this.state.term} //attribute to capture state for each state cycle to make “controlled component”*

*onChange={event => console.log(event.target.value)} />*

*);*

*}*

Arrow function:

() => blablalba function() {return blablabla} // function() {blablabla}

a => blablabla function(a) {return blablabla} // function(a) {blablab}

State:

State is the plain JS object that use to record and react to user’s events. Each class-based component that we defined has it own state object. Whenever state of component changed, the component immediately re-render and also for all of its children to be re-render as well.

Before we can use state in a component, we need to initialize the state object.

Function component does not have state, only class-based component has.

*class App extends Component {*

*constructor(props) { //initalize the state*

*super(props); //calling parent class (Component)*

*this.state = { term: ‘’ }; //can use any word for ‘term’*

*}*

*}*

To set/update the state, we use “this.setState” in other place of the components except constructor (where we use this.state)

To use JS variable inside JSX (HTML language) we put in inside {…}

Controlled component:

Component that the value of the state be updated and store for each re-render

IV. Ajax request with React:

*class App extends Component {*

*constructor(props) {*

*super(props);*

*this.state = {videos : []};*

*YTSearch({key : API\_KEY, term : 'surfboards'}, videos => this.setState({videos : videos}))*

*}*

*render() {*

*return ( // use parancy for multiple lines*

*<div>*

*<SearchBar />*

*<VideoList videos = {this.state.videos}/> passing value from parent to child*

*</div>*

*);*

*}*

*}*

In videos\_list.js

*const VideoList = (props) => {*

*return (*

*<ul className = "col-md-4 list-group">*

*{props.videos.length}*

*</ul>*

*);*

*}*

Passing value from parent (App) to child (VideoList) we use normal JSX syntax and put javascript variable into {}. All properties in VideoList tag (videos) called “props”. Those will store into the parameter of function component VideoList where it is created.

ES6: {video, book} ~ {video : video, book : book}

JSX syntax: <VideoList videos = {…} /> is the same as calling functional component VideoList (from other JS file which importted) and passing object {videos : {…}} to that function so that we can call it by: {props.videos.length}.

If we don’t have *videos = {this.state.videos}* in <VideoList…> tag, we don’t need to have props parameter in VideoList function

Loop array in JS: Use map is better:

a = array.map(function(arg) {do s.t with arg}) -> go through each element in array and pass it to arg of function then do s.t with it (does not change array element)

IV. Reducer:

A reducer is a function that returns a piece of the application state. Because our application can have many different pieces of state, we can have many different reducers. A reducers returns a values of the state in a pair of {key of state : value of state}.

Note: Noname function and how to call:

Filename: reducer\_books.js

export default function() {

return […]

}

In other file, import this function, we can use any name for it.

Filename: index.js

Import { combineReducers } from ‘redux’;

import BookReducer from ‘./reducer\_books’;

const rootReducer = combineReducers({

books : BookReducer,

});

export default rootReducer;

V. Class-based Component:

Use for internal record keeping, something change inside and need to re-render.

Use in:

* Input
* Search

Structure:

*import React, { Component } from ‘react’;*

*class SearchBar extends Component {*

*render() {*

*return (*

*JSX {JavaScript} // JSX exist in return only, JavaScript is everywhere*

*);*

*}*

*}*

*export default SearchBar*

*---------------------*

*import React from ‘react’;*

*class SearchBar extends React.Component {*

*}*

*--------------------*

*export default class SearchBar extends Component {…}*

1. State:

* Use in class component only.
* Initialize in constructor method

*class …*

*constructor(props) {*

*super(props);*

*this.state = { key:value } ;// { term: ‘ ’}*

*}*

*render() {*

*return (…this.setState({term: othervalue}) //setState is the method in Component use via super(*

*}*

*}*

* When the state changed, all class component will be re-render and sub component of the class also be re-render
* Parents component pass state to child component via child’s props and it pass new state value (this.state.key\_of\_state)

1. Controlled Component:

* Use property “value” in a component and bind to its state to control

VI. Props:

Using component: <SearchBar a={…} b={…} c={…} > ABC </SearchBar>

a, b, c are in “props” object of SearchBar

ABC is “children” of SearchBar

Define component: const SearchBar = (props) => { // const SearchBar = function(props) {…}

props.a.key\_of\_object\_a

props.b.key\_of\_object\_b

props.c.key\_of\_object\_c

console.log(props.children);

}

VII. HTTP Request

VII. Events

In React we pass event handler to a function, not a function and events use camelcase. In HTML we pass a string to events:

HTML: <button onclick=”activateLaser()”> Activate </button>

React: <button onClick={activateLaser}> Activate </button>

Your event handlers will be passed instances of SyntheticEvent as parameter, so the event handler mus have 1 parameter in declaration.

SyntheticEvent object has those attributes:

* bubles : boolean
* cancelable: boolean
* currentTarget : DOMEventTarget
* defaultPrevented: boolean
* eventPhase: number
* isTrusted: number
* nativeEvent: DOMEvent
* preventDefault() : void
* isDefaultPrevented(): boolean
* stopPropagation(): void
* isPropagationStopped(): boolean
* target: DOMEventTarget
* timeStamp: number
* type : string

B. Redux:

I. Reducer:

Is a function returns a piece of state. An app may have many states so it may have many reducer function

Ex: Application state: have 2 states: books and active Book, each state is returned by a reducer

{

books: [{title: ‘Harry Potter’}, {title: ‘Javascript’}],

activeBook: {titile: ‘Javascript: The Good Parts’}

}

Create “reducer\_books”: (reducer 1) : this reducer returns an array of object

export default function() {

return [

{ title: ‘Javascript: The Good Parts’ },

{ title: ‘Harry Potter’ },

{ title: ‘The Dark Tower’ },

{title: ‘Eloquent Ruby’ }

]

}

Use reducer in “index.js” file at the same lvl with reducer\_books

import { combineReducers } from ‘redux’;

import BooksReducer from ‘./reducer\_books’;

const rootReducer = combineReducers({

books: BooksReducer

});

export default rootReducer;

LifeCycle – Order in execution:

1. Initial
2. GetDefaultProps
3. GetInitialState
4. ComponentWillMount
5. Updating State
6. Updating Props
7. ComponentWillReceiveProps
8. ShouldComponentUpdate
9. ComponentWillUpdate
10. Render
11. ComponentDidUpdate
12. ComponentDidMount
13. Unmounting
14. ComponentWillUnmount