**map**

var arr = [1, 2, 3, 4, 5];

var arr2 = arr.map(value => value \* 2);

**reduce**

* Same as map but accept function with previous and current element as parameter

var number = [1, 2, 3, 4, 5, 6, 7, 8, 9];

var total = number.reduce(function(previous, current) {

return previous + current;

});

var array\_of\_arrays = [[1, 2], [3, 4], [5, 6]];

var concatenated = array\_of\_arrays.reduce(function(previous, current) {

return previous.concat(current);

});

var numbers = [65, 44, 12, 4];

function getSum(total, num) {

return total + num;

}

function myFunction(item) {

document.getElementById("demo").innerHTML = numbers.reduce(getSum);

}

**Filters**

--------

var tasks =[

{

'name' : 'Task 1',

'duration' : 120

},

{

'name' : 'Task 2',

'duration' : 100

},

{

'name' : 'Task 3',

'duration' : 99

},

];

var taskst = tasks.filter(x => x.duration >100 || x.duration ==100);

**Benefits of React**

* Simpler :
  + Component base architecture with pure Javascript
  + Declarative style not Imperative
  + Developer friendly DOM abstraction
* Fast UIs
  + Virtual DOM - updated only changed element (JQuery – Manual, Angular – All automated all the time)
* Less code to write – Component reuse

Component based Architecture

* Separation of concern
* Loose coupling
* Code reuse

React library has two packages (Version 0.14)

1. React Core (**react** package on npm)
2. ReactDOM (**react-dom** package on npm)

Because of this split rendering syntax got changes

* Before 0.14 (in 0.13)
  + React.render()
* After 0.14
  + ReactDOM.render()

React is almost used with JSX – a tiny language that let developers write React UIs more eloquently. We can transpile JSX into regular Javascript by using **Babel** or a similar tool.

Project Setup

* Create a folder to the project
* Install local Dev Server on project folder from command line
  + http-server or node-static
  + http-server : npm install http-server
* Create folder **js** with project folder and download **react.js** and **react-dom.js**
* Create a html file with project folder and refer above two js files
* Create React part in the html file or as separate js file
* Type http-server to start the server
* Browse the url display in the http-server command

**React.createElement('h1', null, 'Hey') equal to React.DOM.h1(null, ‘Hey’);**

**JSX and Babel**

Jsx – Javascript extension

Babel – Tranpiler for JSX which compile Jsx to React like syntext

render() {

let link

if (this.props.user.session)

link = <a href='/logout'>Logout</a>

else

link = <a href='/login'>Login</a>

return <div>{link}</div>

}

// Approach 2: Expression

render() {

let link = (sessionFlag) => {

if (sessionFlag)

return <a href='/logout'>Logout</a>

else

return <a href='/login'>Login</a>

}

return <div>{link(this.props.user.session)}</div>

}

// Approach 3: Ternary operator

render() {

return <div>

{(this.props.user.session) ? <a href='/logout'>Logout</a> **:**

➥ <a href='/login'>Login</a>}

</div>

}

//Approach : 4 : Using Immediate invoke

render() {

return <div>{

(sessionFlag) => {

if (sessionFlag)

return <a href='/logout'>Logout</a>

else

return <a href='/login'>Login</a>

}(this.props.user.session)

}</div>

}

With JSX :

let h1 = React.createElement("h1", null, "Hello World");

class HelloWorld extends React.Component{

render() {

return React.createElement('div', null, h1, h1)

}

}

ReactDOM.render(

<HelloWorld />,

document.getElementById('content'));

Without JSX :

let h1 = React.createElement("h1", null, "Hello World");

class HelloWorld extends React.Component{

render() {

return React.createElement('div', null, h1, h1)

}

}

ReactDOM.render(

React.createElement(HelloWorld, null),

document.getElementById('content'));

In ES5 if you want to send array as parameters to a function we need to use **apply**

Ex:

Function test (arg1, arg2, arg3)

{

}

Var param = [‘1’,’2’,’3’];

Test.apply(null, param);

Also we can use **spread operator** to call function

**function request(url, options, callback) {**

**// ...**

**}**

**var requestArgs = ['http://azat.co', {...}, function(){...}]**

**request(...requestArgs)**

let msg = "";

if(this.props.taskId == "1")

{

msg = 'You Enter Task Id :' + this.props.taskId;

}

else

{

msg = "You didn't Enter Task Id";

}

let msg1 = (id) =>

{

if(id == "1")

{

return 'You Enter Task Id :' + id;

}

else

{

return "You didn't Enter Task Id";

}

}

We can use msg variable or call msg1 function within return OR

We can run immediate run function with in return

<div>

{

(id1) => {

if(id1 == "1")

return <h3>You Enter Task Id : {id1} </h3>

else

return "You didn't Enter Task Id"

}(this.props.taskId);

}

</div>

**Setting up a JSX transpiler with Babel**

We can use

* Babel
* Node.js
* Grunt, Gulp or Webpack - Popular approach

To convert JSX to javascript but all are using Babel in one way or other.

Install Babel react preset locally

1. Create a new folder, such as ch03/babel-jsx-test.
2. Create a package.json file in the new folder and enter an empty object {} in it,

or use npm init to generate the file.

1. Define your Babel presets in package.json (used in this book and explained in

the next section) or .babelrc (not used in this book).

1. Optionally, fill package.json with information such as the project name, license,

GitHub repository, and so on.

1. Install the Babel CLI and React preset *locally*, using npm i babel-cli@6.9.0

babel-preset-react@6.5.0 --save-dev to save these dependencies in dev-

Dependencies in package.json.

1. Optionally, create an npm script with one of the Babel commands described

shortly.

If you use old browser we can also install Babel-preset to transpil to older version by

**npm i babel-preset-es2015 --save-dev**

and add preset in package.json like

"presets": ["react", "es2015"]

To convert JSX to JS we can run this command

**./node\_modules/.bin/babel js/script.jsx -o js/script.js**

**Also wecan add this in the package.json so that we can run this by**

**Npm run build**

Package.json like below

{

"name": "sample2",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"build": "./node\_modules/.bin/babel js/script.jsx -o js/script.js"

},

"author": "",

"license": "ISC",

"devDependencies": {

"babel-cli": "^6.9.0",

"babel-preset-react": "^6.5.0"

},

"babel" : {

"presets" : ["react"]

}

}

If we run this command to enable watch on Babel when ever you change the JSX file it will be converted to JS file

**./node\_modules/.bin/babel js/script.jsx -o js/script.js -w**

**Build all files JSX file in js to separate js files in build folder we can run ./node\_modules/.bin/babel js –d build**

**Build all files JSX file in js to single js file in build folder we can run**

**./node-modules/.bin/babel js -o script-compiled.js**

Steps

1. Npm init
2. Npm install babel-cli babel-preset-react –save-dev
3. Add

"babel" : {

"presets" : ["react"]

}

So the entire package.json looks like,

{

"name": "babel-setup",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC",

"description": "",

"devDependencies": {

"babel-cli": "^6.26.0",

"babel-preset-react": "^6.24.1"

},

"babel" : {

"presets" : ["react"]

}

}

1. Check Babel version by running the command

./node\_modules/.bin/babel - - version

1. Issue a command to process your js/script.jsx JSX into js/script.js

./node\_modules/.bin/babel js/script.jsx -o js/script.js

But this command is too long to type every time, so we can include this command in package.json like,

“build” : “./node\_modules/.bin/babel js/script.jsx -o js/script.js”

So now you can run by

npm run build

We can also automate this command with the watch option ( -w or -watch)

./node\_modules/.bin/babel js/script.jsx -o js/script.js -w

When our jsx files grow In larger number we can compile all the jsx into separate js files of into a single js file

Separate files :

**./node\_modules/.bin/babel sourcedir - - d builddir**

Single file :

**./node\_modules/.bin/babel sourcedir -o singlefile.js**

1. Install React

npm install react react-dom –save-dev

**Simply Setup React Dev environment (need node version >=6.9.0 and < 7.0.0 or >= 8.9.0)**

1. Npm install -g create-react-app
2. Create-react-app <AppName>
3. Cd <App-Name>
4. Npm start / yarn start (yarn build / yarn test / ) -- this will start your react app in http://localhost:3000
5. Code . – open app in VS code

Props vs state

* State are mutable meaning we can change the data on state but Props are immutable, we can’t change the value of a property
* State are define in the component itself but Props are coming from the parent component

We can write stateless components which does not use any state but can use Properties. Stateless component are predictable means easy to understand, maintain and debug. We can write stateless component as Constant. For example,

Class HelloWorld extends React.Component{

Render(){

Return <h1 {…this.props}> Hello {this.props.frameworkName} world!!!</h1>

}

}

Const HelloWorld = function(props)

{

Return <h1 {…this.props}> Hello {this.props.frameworkName} World!!! </h1>

}

const HelloWorld = (props) => <a href = {prps.href} target = “\_blank” className=”btn btn-primary”> {props.text} </a>

const HelloWorld = (props) => {

return (

<a href = {prps.href} target = “\_blank” className=”btn btn-primary”> {props.text} </a>

)

}

**Lifecycle of React Component**

Lifecycle Events

* Mounting Events - Happen when a React element is attached to a DOM node (Invokes only once)
* Updating Events – Happen when a React element is updated as a result of a new values of its properties or state (Invokes many times)
* Unmounting Events – Happedn when React elemement is detached from the DOM. (Invokes only once)

**Order of the Events**

1. Constructor – Happen when an element is created and let you set the default properties and initial state.
2. Mounting
   1. componentWillMount() - Before mounting to the DOM
   2. componentDidMount() – After mounting to the DOM
3. Updating
   1. componentWillReceiveProps(nextProps) – When the component about to receive properties
   2. shouldComponentUpdate(nextProps, nextState) 🡪 bool - Let you optimize the component’s rendering by determining when to update and when to not update
   3. componentWillUpdate(nextProps, nextState) – Right before the component is updated
   4. componentDidUpdate(prevProps, preState) – Right after the component updated
4. Unmounting
   1. componentWillUnmount() – Let you unbind and detach any event listener or do other cleanup work

We missed rerender event here but rerender will not happen part of the life cycle but we need to force to rerender the update, for this we can use **this.forceUpdate()**  for example if the data changes in render block it is not a part of state or properties so we need to manually trigger the update. BUT this.forceUpdate is not recommended by react team because it will make component impure.

React Events

<button onClick = {(function(event){

console.log(this, event) }).bind(this)

}>

Save

<button>

* here bind is needed so that event-handler function, you get a reference to the instance of the class.
* If you don’t use this in the function you don’t want to bind
* If you use fat arrow function you don’t need bind

We can create a control with event as separate component

Class SaveButton extends React.Component{

handleSave(event){

console.log(this,event)

}

Render(){

Return <button onClick= {this.handleSave.bind(this)}> Save </button>

}

}

Or

Class SaveButton extends React.Component{

Constructor(props){

Super(props)

This. handleSave = this. handleSave.bind(this);

}

handleSave (event){

console.log(this,event)

}

Render(){

Return <button onClick= {this.handleSave}> Save </button>

}

}