Current Tech Notes

**Revisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **User** | **Modifications** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# MOOC Courses

## CourseRa: Specialization: Full stack React

**1: Front-End Web UI Frameworks and Tools: Bootstrap 4**

**--------------------------------------- Resource Links:**

* Bootstrap4: <https://getbootstrap.com/docs/4.3/getting-started/introduction/>
* Review the basic of CSS: <https://www.w3schools.com/css/css_syntax.asp>

**--------------------------------------- Progress:**

* <https://www.coursera.org/learn/bootstrap-4/supplement/kXErq/exercise-instructions-getting-started-with-bootstrap> (stop)

**2: Front-End Web Development with React**

**3: MobileApp-ReactNative**

-----------------------------------[Multiplatform Mobile App Development with React Native](https://www.coursera.org/learn/react-native/home/welcome) from CourseRa:

\*\*react native web site: <http://facebook.github.io/react-native/>

\*\*ES2015 features: <https://babeljs.io/docs/en/learn/>

Stop:

<https://facebook.github.io/react-native/docs/tutorial>

[https://babeljs.io/docs/en/learn/#arrows-and-lexical-this](https://babeljs.io/docs/en/learn/" \l "arrows-and-lexical-this)

\*\*React Native App Development using npm

* npm install create-react-native-app -g
* Scaffolding an App: create-react-native-app <app name>
* create-react-native-app gives access to Expo SDK.

\*\*Use Yarn:

* Yarn is installed
* Yarn command: yarn global add create-react-native-app@1.0.0 (done)
* create-react-native-app --help

\*\*demo app:

* Cd c:\dev\courseRa\spe\_react\react\_native
* create-react-native-app confusion
* yarn command: yarn start; yarn run ios (Mac only require ios); yarn run android; yarn test, yarn run eject
* Installed the expo client ios app in my phone, refer to [https://expo.io/tools#client](https://expo.io/tools" \l "client);
* start the demo app: yarn start
  + Your app is now running at URL: exp://10.0.0.51:19000

› Press a to open Android device or emulator.

› Press s to send the app URL to your phone number or email address

› Press q to display QR code.

› Press r to restart packager, or R to restart packager and clear cache.

› Press d to toggle development mode. (current mode: development)

* Press s to send the app URL to my iPhone, but timed out.
* Commit the initial code to git (done)

**4: Node JS Express MongoDB**

-----------------------------------[Server-side Development with NodeJS, Express and MongoDB](https://www.coursera.org/learn/server-side-nodejs/home/welcome), courseRa:

\*\*Node modules follows the CommonJS module specification

\*\*Demo app:

* Demo app: C:\dev\courseRa\speReact\serverSite\NodeJS\node-examples
* Start the demo app by: 1) npm init; 2) update the package.json; 2) modify index.js, using rectangle.js as external module; 3) to run the app: npm start
* User callback for error handling and JS closure:
* Express: fast, unopinionated minimalist web framework for Node.js

\*A good intro: <http://evanhahn.com/understanding-express/>

\*Install express locally under the project folder: npm install express --save

\*Install morgan: npm install morgan --save

<https://github.com/expressjs/morgan>

\*npm install body-parser --save

<https://github.com/expressjs/body-parser>

\*Express router creates a mini-express application.

Stop @:

<https://www.coursera.org/learn/server-side-nodejs/supplement/iQ9cI/exercise-instructions-express-router>

<http://evanhahn.com/understanding-express/>

## CourseRa: Rice: Java Specialization

## Udemy classes

* **Udemy: Deploy Spring Boot Microservices to AWS - ECS & AWS Fargate**

From <<https://www.udemy.com/course/deploy-spring-microservices-to-aws-with-ecs-and-aws-fargate/?couponCode=FIRST_PROMO_UDEMY&utm_content=promo&utm_campaign=2019-08-26&utm_source=email-sendgrid&utm_term=13363166&utm_medium=1352468>>

## NEXT

# Java

## Modern Java

**------------------------------ Java Generics:**

* Links:

<https://howtodoinjava.com/java/generics/complete-java-generics-tutorial/>

* In layman's term, generics force type safety in java language, issues are caught during compile time, avoid "ClassCastException" in run time.

**------------------------------Java8 features:**

* **Links:**

<https://howtodoinjava.com/java-8-tutorial/>

<https://howtodoinjava.com/java8/java-streams-by-examples/> stop at

3. Convert streams to collections

* **Lambda Expression** (or function): is just an anonymous function.
* **Functional Interface:** The annotation, @FuncationalInterface, permits exactly one abstract method, you are free to add default methods.
* **Default methods** in interface.
* Java 8 **streams**: Java 8 Streams API, which provides a mechanism for processing a set of data in various ways that can include filtering, transformation, or any other way that may be useful to an application.
  + Java 8 streams are designed in such a way that most of the stream operations returns streams only, this help us creating chain of stream operations, also called pipe-lining.
* Java 8 Date/Time API changes

## NEXT

# JavaScript

## General

-------------------TODO:

<https://jsbin.com/help/getting-started/> : verify if JSBin is a good site to do JS testing

-------------------Links:

\*\*javascript tutorial: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript>

\*\*javascript-prototypes: <http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/>

\*\* <https://www.quora.com/What-are-some-of-the-advanced-topics-in-JavaScript>

------------------Tools:

\*\*Babel is javascript compiler

\*\*webpack: is module bundler for modern javascript applications.

------------------Important concepts:

\*\*JavaScript Operators:

* JS ES6 spread syntax: "…"

\*\*Two most noticeable features of JS:

* First-class function: a function can be treated the sme ways as any other variables.
* Closures:
  + A function defined inside another function has access to all the variables declared in the outer function (outer scope)
  + The inner function will continue to have access to the variables from the outer scope even after the outer function has returned (?).

\*\*JavaScript Objects( <http://javascriptissexy.com/javascript-objects-in-detail/> ):

* JS has one complex data type, the object data type, which is **mutable** and **save-as-reference**.
* JS has 5 primitive data types: Number, String, Boolean, Undefined and Null. They are **immutable** and **save-as-value**.
* An object is an unordered list of primitive data types (and sometimes reference data types) that is stored as a series of name-value pairs. Each item in the list is called a property (functions are called methods, functions are objects within objects?).
* JSON.stringify function to serialize your objects, use the JSON.parse to deserialize object

\*\*JavaScript Prototype (<http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language>) :

* Every JS function has a prototype property, which is used primarily for inheritance.
* Prototype attribute is the characteristic of the object, for example, it tells the object's parent.

\*\*JavaScript Closure( <http://javascriptissexy.com/understand-javascript-closures-with-ease/> ):

* A closure is an inner function that has access to the outer (enclosing) function’s variables—scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function’s variables, and it has access to the global variables.
* The inner function has access not only to the outer function’s variables, but also to the outer function’s parameters.
* The closure has access to the outer function’s variables by reference, not by value. you can use an Immediately Invoked Function Expression (IIFE) to avoid the value of the variable changing due to reference.
* Closures are used extensively in Node.js; they are workhorses in Node.js’ asynchronous, non-blocking architecture.

\*\*JavaScript: Higher order component (HOC) <https://medium.com/ovrsea/mapstatetoprops-and-why-you-may-not-need-mapdispatchtoprops-as-a-beginner-dd012a3da5e6>, <https://daveceddia.com/extract-state-with-higher-order-components/>:

* It is a function that takes a component as input and returns the same component with new props.
* Redux uses HOC to give you the possibility to access the data and also to modify your store, this data and methods are accessible through props, this is why we have to **mapStateToProps** and **mapDispatchToProps**.
* your store might be huge, and your component does not need to be aware of all the changes in it. **You want to choose which variables are accessed by which component.**This is where **mapStateToProps** comes into action !
* The only way to update data in your store is to **dispatch** actions. Well, something only a few knows: **dispatch is accessible as a props if you don’t provide a mapDispatchToProps function.**

\*\*JavaScript callback functions (high-order functions) ( <http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/> ):

* JS functions are first class objects: they are of type Object and they can be used in a first-class manner like any other objects (String, Array, Number etc), they can be stored in variables, passed as arguments to functions, created within functions, and returned from functions.
* Because functions are first-class objects, we can pass a functions as an argument in another function and later execute that passed-in function or even return it to be executed later, this is the essence of using callback functions in JS. Callback functions are probably the most widely used functional programming technique in JS.
* Callback function is a closure, closures has access to the containing function's scope and also the variables from the global scope.

\*\*JavaScript’s “this” (<http://javascriptissexy.com/understand-javascripts-this-with-clarity-and-master-it/> ) :

* "this" is not an assigned value until an object invokes the function where this is defined.
* All functions in JS has properties, just as objects have properties. And when a function executes, it gets the "this" property - **a variable with the value of the objects that invoke the function where "this" is used.**
* We can use bind(), apply() or call() method to specifically set the value of "this".
* It is important to take note that **closures** cannot access the outer function's "this" variable by using the "this" keyword because the "this" variable is accessible only by the function itself, not by inner functions.

\*\*JavaScript’s Apply, Call, and Bind Methods (<http://javascriptissexy.com/javascript-apply-call-and-bind-methods-are-essential-for-javascript-professionals/>):

* Apply and Call are nearly identical and are frequently used in JavaScript for borrowing methods and for setting the this value explicitly, we use Bind for setting the this value in methods and for currying functions.

Note: Apply and Call come with ECMAScript 3 (available on IE 6, 7, 8, and modern browsers), ECMAScript 5 (available on only modern browsers) added the Bind method

* ddd

\*\* "use strict": indicate that the code should be executed in "strict mode", for example, you cannot use undeclared variables

\*\*map():

The **map**() **method** creates a new **array** with the results of calling a **function** for every**array** element. The **map**() **method** calls the provided **function** once for each element in an **array**, in order.

## Angular

------------------------------------Links:

<https://angular.io/>

------------------------------------Tutorial:

\*\*https://www.hongkiat.com/blog/angularjs-tutorials-screencast/

\*\*https://codesjava.com/angularjs-directives-ng-app-init-model-repeat-controller-bind (text based, may be old)

------------------------------------Templates:

<https://startangular.com/product/sb-admin-bootstrap-4-angular-6/>

<http://angularexpo.com> - to get angular themes

-------------------------------------Notes:

You write Angular applications by composing HTML templates with Angularized markup, writing component classes to manage those templates, adding application logic in services, and boxing components and services in modules.

the you launch the app by bootstrapping the root module.

Angular apps are modular and Angular has its own modularity system called NgModules:

1) Every Angular app has at least one NgModule, the root module, ususally nambed, AppModule.

2) An NgModule is a class with an @NgModule decorator, which takes a single metadata object whose propterties describe the module:

\*declarations - the view classes that belong to this module: components, directives, pipels

\*exports

\*imports

\*providers - creators of services that this module contributes to the global collection of services.

\*bootstrap - the main application view, called the root component, that hosts all other app views. Only the root module should set this property.

----------RequiredJS--------------

RequiredJS is a java script library that helps in lazy loading JavaScript dependencies.

RequireJS implements Asynynchronous Modules specified by CommonJS.

The dependency injection system built into AngularJS deals with the objects needed in a component; while dependency management in RequireJS deals with the modules or, JavaScript files.

----------Angular JS--------------

angularjs sandbox

angular JS internal

<http://www.sitepoint.com/using-requirejs-angularjs-applications/>- about required JS and angular JS

angular JS and jQuery

<https://docs.angularjs.org/tutorial/step_08> - stop

<https://docs.angularjs.org/guide/providers>, answer below question

<https://docs.angularjs.org/api/ng/service/$q> -- promised object?

test page: [file:///C:/apptest/angularJS/t2/index.html](file:///C:\\apptest\\angularJS\\t2\\index.html)

<script src="Scripts/angular.js"></script>

what is the difference between provider and service?

Provider can only be injected into config functions, thus you could not inject $routerProvider into PhoneListCtrl.

what is function in angular JS?

<http://www.smashingmagazine.com/2015/01/22/angularjs-internals-in-depth/> - read one more time

<http://blog.xebia.com/2013/09/01/differences-between-providers-in-angularjs/> - read one more time

<http://www.dotnet-tricks.com/Tutorial/angularjs/4E7J080914-Understanding-AngularJS-$watch(),-$digest()-and-$apply().html>

bower?

-----------angularJS 2.0:

<http://angularjs.blogspot.com/2014/03/angular-20.html>

<http://onehungrymind.com/10-things-consider-keeping-level-head-angularjs-2-0/>

A lot of these ideas going in to Angular 2.0 (should it be Angular\*JS\* 2.0 as it will also be for Dart?) are already in AngularDart. After switching over to AngularDart the advantages of notations and killing the controller are easy to see. Code is cleaner, easier to understand, there is less abuse of the scope so templates are easier to grok and your IDE can now help you! This is definitely a step in the right direction.

------------

Games and GUI editors are examples of applications with intensive and tricky DOM manipulation. These kinds of apps are different from CRUD apps, and as a result are probably not a good fit for Angular. In these cases it may be better to use a library with a lower level of abstraction, such as jQuery.

HTML is a great declarative language for static documents,

Angular handles all of the DOM and AJAX glue code you once wrote by hand and puts it in a well-defined structure

Angular comes with: data-binding, basic templating directives (?), form validation, routing, deep-linking(?), reusable components and dependency injection (?), testing, seed application (?)

About declarative vs. imperative programming:

read this one:http://stackoverflow.com/questions/1784664/what-is-the-difference-between-declarative-and-imperative-programming

Declarative programming:

when you say what you want.

we do not specify any "implementation details"

One benefit of declarative programming is that it allows the compiler or framework to make decisions that might result in better code than what you might make by hand

imperative programming:

when you say how to get what you want.

Service: moves view-independent logic from the controller into a service

Within ANgular, the Dependency Injection (DI) container is called the injector.

In general, a controller shouldn't try to do too much. It should contain only the business logic needed for a single view.

Scope is an object that refers to the application model. It is an executtion context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.

For AngularJS, "compilation" means attaching event listeners to the HTML to make it interactibe.

Adirective is just a function which executes when the compiler encounters it in the DOM

Watchers are the single most important aspect of an AngularJS application's data-binding capabilities.

when an angular application starts with a given application module, angular creates a new instance of injector, which in turn creats a registry of recipes as a union of all recipes defined in the core "ng" module, application module and its dependencies. The injector then consults the recipe registry when it needs to create an object for your application.

----------jQuery --------------

STOP @ <http://www.w3schools.com/jquery/jquery_ajax_load.asp>

test file: C:\apptest\jQuery\test.html

jQuery is a lightweight JS library. It sinmplifies a lot of complicated things from JS, like AJAX calls and DOM manipulation:

\* HTML/DOM, CSS manipulation

\* HTML event methods

\* Effects and animations

\* AJAX

\* Utilities - example?

\* plugins for almost any task out there .... how to write plug in for jQuery?

With jQuery, you select (query) HTML elements and perform "actions" on them: $(selector).action()

jQuery is tailor-made to respond to events in an HTML page

<http://quickstart.web.att.com/index.html>

\*\*\*\*\*\*Angular JS, AJAX, JSON, RequireJS, jQuery, Angular JS, APIGEE(?) \*\*\*\*\*\*\*\*\*

\*Angular JS is java script MVC (MVVM) framework, a full-featured SPA framework.

\*Angular is a solid core that you can built on top of it without including many othe libraries and worry about version issues, easier to maintanin.

\*Angular includes two way data binding, MVC, routing (SPA and bring the mini pages to the shell page), testing, jqLite (Angular plays well with jQuery) for DOM manipulation, templates, history, factories (share code), directives (teach HTML new tricks?), dependency injection (dynamically injecting features).

\*$scope is the glue (ViewModel) between a controller and a view

Dan Wahln's you tube tutorial:

<http://tinyurl.com/AngularJSDemos>

<https://www.youtube.com/watch?v=i9MHigUZKEM>

stoped @ 53.29

\*\*\*\*\*\*MVVM - model-view-view model pattern:

The model is that it holds the information, but not behaviors or services that manipulate the information.

The view is the presentation of the data

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Rsponsive Web Design:

<http://www.w3schools.com/bootstrap/bootstrap_case_navigation.asp>

C:\Users\dl8899\Documents\projects\next\_step\Bootstrap\test.html

<http://startbootstrap.com/template-categories/all/>

downlaoded:

C:\Users\dl8899\Documents\projects\next\_step\Bootstrap\startbootstrap-agency-1.0.4\startbootstrap-agency-1.0.4

## React

---------------------------------------Current

------------React online Tutorial

<https://reactjs.org/>

----Following this tutorial:

<https://reactjs.org/tutorial/tutorial.html>

Getting error:

×ReferenceError: 'React' is undefined

Generate a new demoproject to compare

------------------------------------------------Tutorial:

\*\*Official: <https://reactjs.org>

\*\*Recommended tutorials (5):https://www.javascriptstuff.com/getting-started-tutorials/

\*\*Recommended tutorials: <https://hackr.io/tutorials/learn-react>

\*\*https://medium.mybridge.co/learn-react-js-from-top-45-tutorials-for-the-past-year-v-2018-28b7f4d4b2c4

\*\*Recommended: <https://hackr.io/tutorials/learn-react>

------------------------------------React:

-------------------------------------------------Templates:

\*\*https://startreact.com/

\*\*https://coreui.io/react/

\*\*https://themeforest.net/tags/react

--------------------------------------------------React Dev tools:

\*\*The React Devtools extension for Chrome and Firefox lets you inspect a React component tree in your browser devtools.

It exists both as a browser extension (for [Chrome](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi) and [Firefox](https://addons.mozilla.org/firefox/addon/react-devtools/)), and as a [standalone app](https://github.com/facebook/react-devtools/tree/master/packages/react-devtools) (works with other environments including Safari, IE, and React Native).

<https://github.com/facebook/react-devtools>

---------------------------------------Learn React:

---------Doc:

stop @Introducing JSX

---------tutorial:

Stop @ <https://reactjs.org/tutorial/tutorial.html>

----stopped @ Lifting State Up

work @ c:\dev\0\_react\my-app

npm install -g create-react-app

create-react-app my-app

**---------------React App Env set up:**

* Links:

<https://daveceddia.com/multiple-environments-with-react/>

* Use .env Files:

If you want to set environment-specific variables, put those in files named .env.development, .env.test, or .env.production for the environment you need:

REACT\_APP\_SPECIAL\_FEATURE=true

REACT\_APP\_API\_HOST=default-host.com

1. Create .env file, and add it to .gitignore
2. Follow instruction in: <https://blog.logrocket.com/create-react-app-a-quick-setup-guide-b812f0aad03c>

**---------------How to structure React Project:**

* Links:

<https://daveceddia.com/react-project-structure/>

* Suggestion:

Machine generated alternative text:
s rc 
api.js 
components 
Button. js 
Icon. js 
UserDetail. js 
UserList.js 
containers 
App.css 
App. js 
App. test. js 
HomePage. js 
UserDetailPage. js 
UserListPage. js 
images 
logo. svg 
index. js 
utils 
testUtils.js 

When you add Redux to your project, start off with something like this:

* src/redux/actions - Create a file for each set of related actions, like userActions.js, productActions.js, etc. I like to bundle action creators and the related action constants in the same file.
* src/redux/reducers - Create a file for each reducer, and an index.js in here to contain the “root” reducer.
* src/redux/configureStore.js - Create and configure the store here. You can just import rootReducer from './reducers'.

------------React Router:

* Documents: <https://reacttraining.com/react-router/web/guides/philosophy>
* React Router packages:
  + React-router: the core of react router.
  + React-router-dom: DOM bindings for React Router
    - import { BrowserRouter, Route, Switch, Link} from 'react-router-dom';
  + React-router-native: react native for react router
  + React-router-config: static route config helpers

------------JS v. JSX:

* Cleaner import for js, require('./MyComponent'); vs require('./MyComponent.jsx');
* Don't worry about it for now

---------------------React Info:

\*Conceptually components are like JavaScript functions. They accepts arbitrary inputs (called "props") and return React elements describing what should appear on the screen.

\*Component **Props** are read only: whether you declare a component as a functions or a class, it must never modify its own props. (immutable)

Note: functions are called [“pure”](https://en.wikipedia.org/wiki/Pure_function) because they do not attempt to change their inputs, and always return the same result for the same inputs. All React components must act like pure functions with respect to their props.

\***State** and **Lifecycle**:

-- **Component state** is similar to props, but it is **private** and fully controlled by the component, only component from class has state.

1. When <Clock /> is passed to **ReactDOM.render()**, React calls the **constructor** of the Clock component. Since Clock needs to display the current time, **it initializes this.state** with an object including the current time. We will later update this state.
2. **React then calls the Clock component’s render() method**. This is how React learns what should be displayed on the screen. React then updates the DOM to match the Clock’s render output.
3. When the Clock output is inserted in the DOM, React calls the componentDidMount()lifecycle hook. Inside it, the Clock component asks the browser to set up a timer to call the component’s tick() method once a second.
4. Every second the browser calls the tick() method. Inside it, the Clock component schedules a UI update by calling setState() with an object containing the current time. Thanks to **the setState() call, React knows the state has changed, and calls the render() method again** to learn what should be on the screen. This time, this.state.date in the render() method will be different, and so the render output will include the updated time. React updates the DOM accordingly.
5. **If the Clock component is ever removed from the DOM**, React calls the **componentWillUnmount**() lifecycle hook so the timer is stopped.

-- It is commonly called a “top-down” or “unidirectional” data flow. Any state is always owned by some specific component, and any data or UI derived from that state can only affect components “below” them in the tree.

\*A **higher-order function** is a function that takes a function as an argument and returns a function.

------------------------------------React and Angular:

Instead of two-way data binding, React emphasized unidirectional data flow. The React approach said that two-way data binding didn’t scale well and created applications that were difficult to understand. It also introduced things like JSX and the virtual DOM

-----------------------------------Use axios with react:

<https://alligator.io/react/axios-react/>

-------------------------------------React and Redux:

<https://daveceddia.com/how-does-redux-work/>

<https://www.sohamkamani.com/blog/2017/03/31/react-redux-connect-explained/>

\*The state managed by redux is application state, not the component state.

\*An "**action**" is a JS object that describes a change we want to make. It requires to have a "type" property, as a string.

\*To **change data**, we need to **dispatch** an **action**.

\*To **obtain data** we need to get the current **state** of the **store**.

\***Reducer** is function that takes **current** state and action as arguments, and returns the next state of the app. It tells redux what our state should look like.

\***Redux state**: State is read-only, and actions are the only way to modify it. Changes happen one way, and one way only: dispatch(action)--> reducer --> new State -->rerender. The reducer function must be "pure" - it cannot modify its arguments (state and action).

\***Provider** is a react component from"react-redux" library that serves one purpose: to provide the store to its' child components. only components within the Provider can be connected (In the below diagram, the connect is done through the Provider).

\*We only have access to the top (dispatch) and bottom (state) component of the redux flow. **Connect** map the stores **state** and **dispatch** to the **props of a component**.

\*The **dispatch**() function can be accessed directly from the store as **store.dispatch(),** but more likely you will access it using a helper like react-redux's **connect**(). You can use **bindActionCreators**() to automatically bind many action creators to a **dispatch**() function.

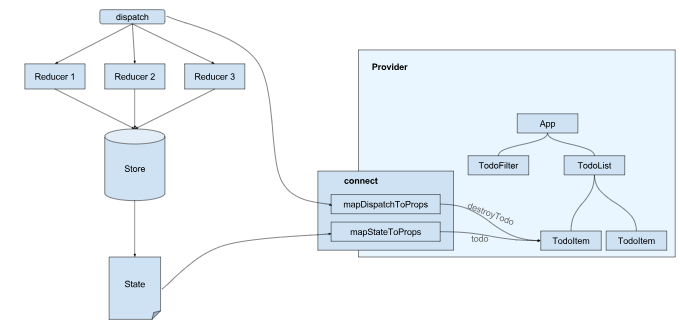
In addition to injecting the result of **mapStateToProps** as **props**, **connect** also inject **dispatch** function as a prop.

\***mapStateToProps** and **mapDispatchToProps** are both pure functions that are provided the stores “state” and “dispatch” respectively. Furthermore, both functions have to return an object, whose **keys** will then be passed on as the props of the component they are connected to.

\***connect** returns a customized function, and this function returns a "connected" version of the component supplied as argument. That explains the use of 2 groups of parentheses. The 1st one is for the arguments for the connect method itself, and the second one is the arguments of the customized function which has been returned by connect:

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

mapStateToProps: you are passing read-only properties for a component to consume.



-------------------------------------Immutability in React and Redux:

* <https://daveceddia.com/why-not-modify-react-state-directly/>
* In React, it is important to never mutate state or props; To modify state, always use this.setState; as for props, they are one ways thing.

-------------------------------------React Router:

React Router is React version of SPA:

A Web Page 
00 xo 
user the URL 
Content! 
His tory 
Hey History the user 
just the URL. 
heres the new one 
React 
Hey ReactRouter. 
heres a new URL. 
React •Router 
Updates the react 
shown on the scren. depending 
on the URL 
Hey React. here are the 
components you need to render 

-------------------------------REACT Component LifeCycle: <https://engineering.musefind.com/react-lifecycle-methods-how-and-when-to-use-them-2111a1b692b1>

* componentWillMount: 99% of your components should probably not use this, and it is not a good idea to start AJAX call here to load data for your component.

Most Common Use Case: App configuration in your root component.

Can call setState: Don’t. Use default state instead.

* componentDidMount: here is where you make the ajax call and load data for your component. Doing AJAX here will ganrantee that there is a component to update. Here is there you want to do all the setup you couldn't do without a DOM.

Most Common Use Case: Starting AJAX calls to load in data for your component.

Can call setState: Yes.

* componentWillReceiveProps: we have access to both the next props (via nextProps), and our current props (via this.props).

Most Common Use Case: Acting on particular prop changes to trigger state transitions.

Can call setState: Yes.

* shouldComponentUpdate: ShouldComponentUpdate allows us to say: only update if the props you care about change.

Most Common Use Case: Controlling exactly when your component will re-render.

Can call setState**:** No.

* componentWillUpdate: we never use componentWillUpdate. Functionally, it’s basically the same as componentWillReceiveProps, except you are not allowed to call this.setState.

Most Common Use Case: Used instead of componentWillReceiveProps on a component that also has shouldComponentUpdate (but no access to previous props).

Can call setState**:** No**.**

* componentDidUpdate:

Most Common Use Case: Updating the DOM in response to prop or state changes.

Can call setState**:** Yes.

* componentWillUnmount

Most Common Use Case: Cleaning up any leftover debris from your component.

Can call setState**:** No.

------------------------------------React Native:

\*\*official site: <https://facebook.github.io/react-native/docs/getting-started.html>

\*expo SDK: <https://expo.io/>

## NEXT

## NEXT

# Designing NEXT

## Array

123

## Requirement and Goal of the System

Functional and none functional requirement

## Design Consideration

## Capacity Estimation and Constrains

## High level System Design

## Database Schema

## Data Size Estimation

## Component Design

## Reliability and Redundancy

## Data Sharding

## Ranking and News Feed Generation

## News Feed with Sharded Data

## Cache and Load Balancing