

The beginners

GUIDE TO REACT & REDUX

The Complete React Web
Development Course (with Redux)



Section 1

REACT 101

- Install Visual Studio Code
- Install Node.js
- Setting up a Web Server
- Setting Up Babel
- Exploring JSX & JSX Expressions
- ES6 Variables & Arrow Functions
- Manual Data Binding
- Forms & Input
- Arrays in JSX

In this section we will dive into the very basic fundamentals of React and setting up your own web developer environment to write and execute your own code. We will go through installing a very popular text editor release by Microsoft called Visual Studio Code to installing Node.js on your machine and then onto setting up your very own local web server and installing Babel.

Once we have setup your developer environment we will continue to learn the fundamentals of JavaScript (ES6) and JSX syntax to build our foundational skills on both languages and technologies on which the React Library is built upon.

After learning the basics we will move onto the next sections where we will start learning more about the React component architecture and will build our own applications.

It will soon become apparent why React is considered a very popular frontend JavaScript library as you are not required to learn a new language but rather learn an application architecture based on components.

INSTALLING VISUAL STUDIO CODE



There are many free and paid text editors out there available to download and use to write your own code, popular editors include Atom, Sublime Text, Notepad++ and Visual Studio Code to name a few. We will be using Visual Studio code as our text editor of choice to write our code as it allows us to install some useful packages to help write our code.

To install Visual Studio code simply visit the following website and download and install the application onto your machine: <https://code.visualstudio.com/>

Visual Studio Code has some useful extensions which you can install:

- Bracket Pair Colorizer
- ES7 React/Redux/GraphQL/React-Native Snippet
- Liver Server
- Prettier - Code Formatter
- Babel ES6/ES7

INSTALLING NODE.JS

Node is JavaScript on the server. You can visit the following website to download node onto your machine: <https://nodejs.org/en/>

Download the latest version of node that is available on their website.

To check that you have node.js installed on your machine, simply open up your terminal and enter the following command:

```
$ node -v
```

This will allow us to double check that node was installed onto our machine as we now have this new command and it also shows us what version of node you have installed on your machine. When installing node we also got NPM (node package manager) which allows us to install various dependencies/packages such as React or Yarn and other libraries. NPM and Yarn aims to do the same job. To check that you have npm installed enter the following command in your terminal:

```
$ npm -v
```

To install yarn on your machine globally run the following command in your terminal:

```
$ npm install -g yarn
```

On windows machines you will need to restart your machine to complete the installation. To check that yarn has installed successfully, run the following code in your terminal:

```
$ yarn --version
```



SETTING UP A WEB SERVER

To setup a developer web server we can achieve this in two ways using live-server.

Firstly you will need to create a directory (folder) for your application. This folder will act as a place for all your project code. This folder can be called anything for example 'indecision-app'. In this example we will create a sub-folder called public and store our basic HTML file.

If we open VS Code and have installed live-server extension we can simply open up our html document and right-click to open the file with live-server. Every-time we update our project files in the folder, the live-server will refresh the browser which will update our application with the changes automatically.

Alternatively, we can use npm or yarn to install live-server globally onto our machines using either command in the terminal:

```
$ npm install -g live-server    or    $ yarn global add live-server
```

To check that we have installed live-server on our machine correctly we would run the command:

```
$ live-server -v
```

To run live server from the terminal, navigate to your file directory using cd and the file path. Note: you can use **cd** to change directory, **ls** (or **dir** on windows) to list all the files within the folder. You can use **cd ~** to navigate back to your user folder. Once you have navigated to your project directory run the following code:

```
$ live-server public
```

Note: you would run live-server followed by the folder name in the directory you wish to serve through the live web server (in our example we had a sub-folder called public which contained our HTML file). Any changes made in the folder will automatically update in the browser.

SETTING UP BABEL

Babel is a compiler and allows you to write for example JSX, ES6, ES7 code and have it compile down to regular ES5 code, allowing your code to work on browsers which only support ES5 syntax.

<https://babeljs.io/>

Babel on its own has no functionality. Babel is a compiler but its not going to compile anything by default. We have to add various plugins and presets in order to get any sort of change in our code (e.g. taking JSX code and converting it into ES5 createElement calls). A preset is just a group of plugins.

We are going to install babel locally on our machines so that when we write our code in JSX/ES6/ES7 it will compile locally to our ES5 code (i.e. we want to write our code locally on our machine and update without using Babel on the web).

The babel website has a docs page which provide documentation on plugins available to you to install. We will install two presets: react preset and env preset. These presets have all the necessary plugins we require and we would not need to install the plugins individually by ourselves (*as this could get out of hand very quickly*). The env preset will include ES2015, ES2016 and ES2017 plugins which will give us access to the new JavaScript features (e.g. arrow functions, const and let variables, spread operator etc).

In our local environment we are going to install three things:
Babel itself, env preset and react preset.

In your terminal run the following command to install babel @v6.24.1 globally on your machine.

```
$ npm install -g babel-cli@6.24.1 or $yarn global add babel-cli@6.24.1
```

This will give us a new command in our terminal which we can run while in our project directory. Run the following code to check if Babel has been installed successfully. This should print out the help output in your terminal and will indicate if Babel has been installed successfully on your local machine.

```
$ babel --help
```

To clear the terminal enter the command line:

```
$ clear
```

The react and env presets will be installed locally in our projects (i.e. these codes will live in our projects so that babel CLI can take advantage of these codes to transform our code down to ES5 syntax). Within the project directory enter the following command:

```
$ npm init    or    $ yarn init
```

This command will setup our project to use node/yarn and specify the local dependencies. This will walk us through series of questions such as the app name, version, description, entry point, repository, author and license (MIT).

All this does is, it generates a brand new file in our project called package.json (*note we could have created this file ourselves without the npm/yarn init command*). The whole point of package.json is to outline all the dependencies that the project needs in order to run. This will make it easy to install everything.



We are going to add our dependencies to this package.json file using the following commands in the terminal:

```
$ npm install babel-preset-react@6.24.1 babel-preset-env@1.5.2    or
```

```
$ yarn add babel-preset-react@6.24.1 babel-preset-env@1.5.2
```

This will install the two dependencies and will update the package.json file in our app folder which will now list our dependencies for our app. Notice that a new folder has been created called node_modules. This is where all the modules and dependencies will live.



The dependencies will have their own package.json file which will list the dependencies they require to run. This will install all the sub-dependencies of their own in the node_modules folder. The package.json file makes it easy to install all the dependencies and sub-dependencies your application will need.

We will never need to go into the node_modules folder to change any of the code. This folder is an auto-generated folder from the package.json file and we can always delete and reinstall this folder/modules using the dependencies information from the package.json file (we would use the command `$ npm install` or `$ yarn install` to install the node_modules folder again).

This command has also generated a package-lock.json file (if you used yarn this file will be called yarn.lock) in our project folder. We will not need to edit/manually change this file. This file lists out all the dependencies in the node_module folder, the version used and where exactly it got that package from. This helps npm/yarn behind the scenes.

We are now able to use Babel locally on our machines within our project directory to compile our React JSX code into regular ES2015 code. In the terminal we will need to run the following command with a few arguments:

```
$ babel src/app.js --out-file=public/scripts/app.js --presets=env, react --watch
```

The first argument specifies the path to our code we want to compile (in our example above it lives in the src folder and the file is called app.js).

The second argument specifies the output file (in our example above it lives in the public/scripts folder and is also called app.js). This file will always be overridden by Babel.

The third argument specifies the presets we would like to use (this is a comma separated list of the presets we wish to use). Finally, the last argument will watch for any changes in the first specified file to update (compile) the second specified file automatically.

Babel will compile the JSX/ES6/ES7 code in one file into regular ES2015 code the browser will understand in the other file.

EXPLORING JSX

In JSX you can only have a single root element. If you want to add more elements side by side we will need to wrap it within a single root element for example:

```
var template = <div><h1 id="someid">Header Text</h1><p>Paragraph Text</p></div>;
```

```
var appRoot = document.getElementById('app');
```

```
ReactDOM.render(template, appRoot);
```

When we are creating JSX expressions, we can get really complex expressions to include a lot of information/nested elements - however, we must have a single root element.

We can make our JSX expression more readable by formatting the elements on separate lines i.e. format as we would do in our html code for example:

```
var template = (  
  <div>  
    <h1 id="someid">Header Text</h1>  
    <p>Paragraph Text</p>  
  </div>  
);
```

The above would still be seen as valid JSX expression and table will successfully render the expression down to ES2015.

The above is more readable to the eye and easily understood. We can make the expression even more complex and as long as there is a single parent root element (the `<div>` wrapper) then this will be seen by Babel as valid JSX. Below is another example of a complex JSX expression with nested elements.

```
var template = (  
  <div>  
    <h1 id="someid">Header Text</h1>  
    <p>Paragraph Text</p>  
    <ol>  
      <li>Item One</li>  
      <li>Item Two</li>  
    </ol>  
  </div>  
);
```

All this is doing is creating nested `React.createElement()` functions calls to create each element in the ES2015 syntax.

```
React.createElement('ol', null,  
  React.createElement('li', null, 'Item One'),  
  React.createElement('li', null, 'Item Two')  
)
```

This is why we do not write our React code in `React.createElement` calls as it is difficult to read and write, instead we use JSX.