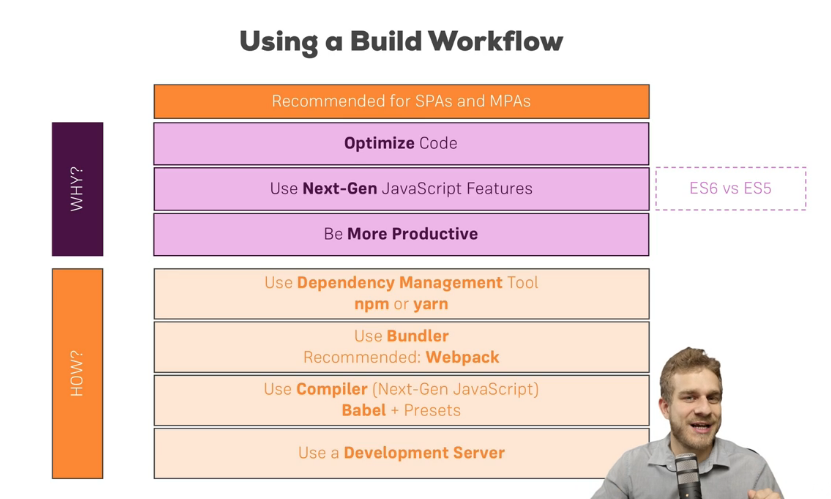
**Building our first React Application**

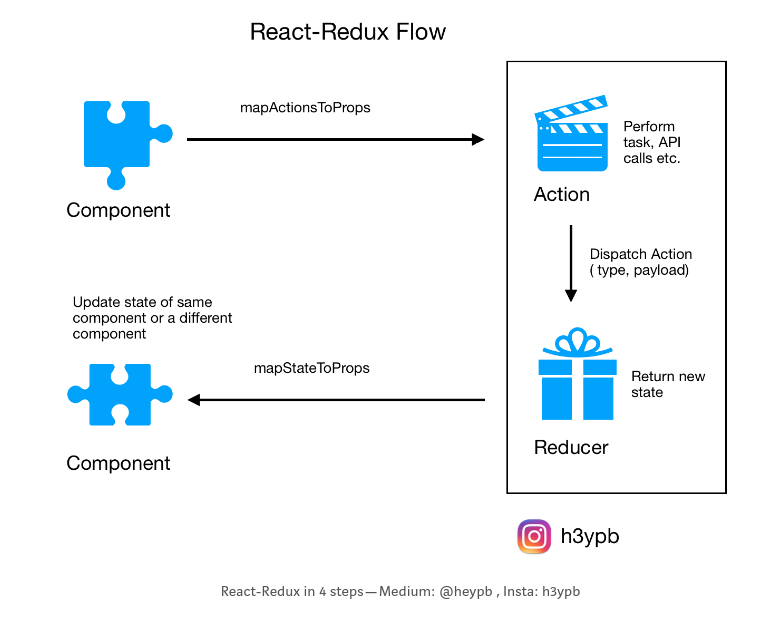
* The build workflow – what all do we use to build a react application
* Using Create React App
* Understanding the React Folder Structure
* Understanding basics of components
* Understanding JSX
* JSX Restrictions
* Creating a functional component
* Working with Components and re-using them
* Outputting Dynamic Content
* Working with Props
* Understanding the Children property
* Understanding and using state
* Props & State
* Handling events with Methods
* To Which events can you listen?
* Manipulating the State
* Function Components Naming
* Using the useState() hook for State Manipulation
* Stateless vs Statefull Components
* Passing method references between components
* Adding Two Way Binding
* Adding Styling with Stylesheets
* Working with inline Styles
* Assignment 1
* Useful Resources and Links

**The Build Workflow**

We need multiple technologies to bring together a React application. The technologies and the overall workflow is described in the following page.

* React application build workflow : <https://medium.com/front-end-weekly/what-are-npm-yarn-babel-and-webpack-and-how-to-properly-use-them-d835a758f987>
* React – Redux Application architecture : <https://hackernoon.com/https-medium-com-heypb-react-redux-workflow-in-4-steps-beginner-friendly-guide-4aea9d56f5bd>





We use the following technologies to bring together our react application.

* NPM and Yarn
* Babel
* Webpack

**NPM and Yarn**

NPM: [https://www.npmjs.com](https://www.npmjs.com/)

Yarn: [https://yarnpkg.com](https://yarnpkg.com/)

Yarn is a superset of NPM that solves many problems that NPM has.

NPM stands for node package manager. It is a package manager for node based environments. It keeps track of all the packages and allows the developer to easily update or remove these dependencies. All of these external dependencies are being inside a file called **package.json.**

The initial file can be created using CLI **npm init.** When you install a package using NPM, the packages get downloaded from a dedicated registry. There are a lot of features of NPM like publishing. Every language we use has a official package manager, either official or a 3rd party one.

* PHP has composer
* Python has PyPi
* Java has Gradle

Yarn is a package manager that uses NPM registry as its backend. Yarn has 2 main advantages over NPM.

* Firstly, Yarn creates a **yarn.lock** file. This file stores the exact versions of dependencies to the last digit. When installing, yarn checks the lock file for the versions, then checks the package.json file. NPM has a **shrinkwrap** command that does exactly this. However, Yarn creates and updates its lock files automatically.
* Yarn is faster than NPM. Yarn installs packages in parallel, while npm does it sequentially.

**Babel** [https://babeljs.io](https://babeljs.io/)

As any language, Javascript also has versions named ECMAScript (short for ES). Currently, most browsers support ES5. ES5 used to be good even though it was painful to code in it. Remember, this not reading from inside callback functions? The new version of Javascript, ES6, also known as ES2015 (specs of the language were finalized in June 2015) makes Javascript great again. If you want to learn about ES6, check out the links at the end of this article. All the great features of ES6 come with one big problem — majority of browsers do not fully support them. That’s when Babel comes to play. Babel is a JS transpiler that converts new JS code into old ones. It is a very flexible tool in terms of transpiling. One can easily add presets such as es2015, es2016, es2017, so that Babel compiles them to ES5.

**Webpack** [https://webpack.js.org](https://webpack.js.org/)

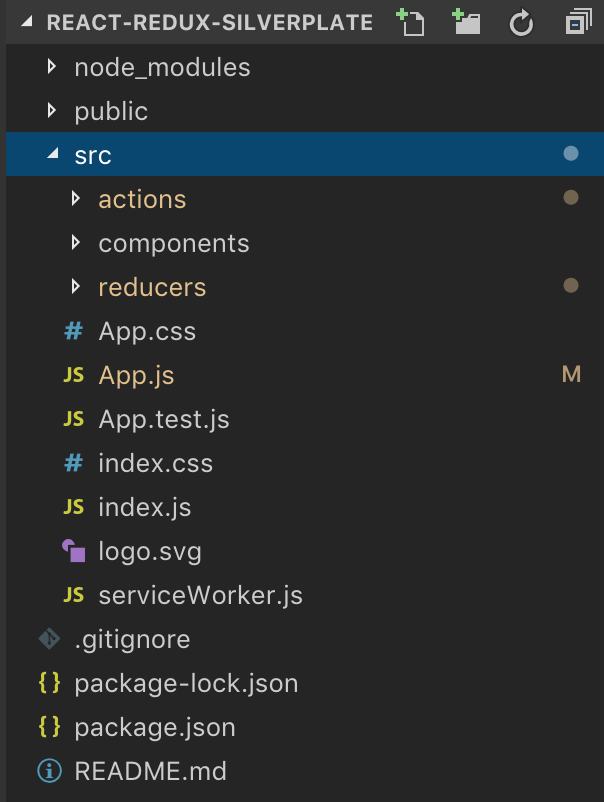
Now that we know what Babel and ES6/7 are, we would like to use that. We would also like to use SASS for our styles, PostCSS for autoprefixing. Plus, we would like to minify and uglify both our CSS and Javascript code. Webpack solves all of these problems using one config file (named webpack.config.js) and one CLI command webpack.

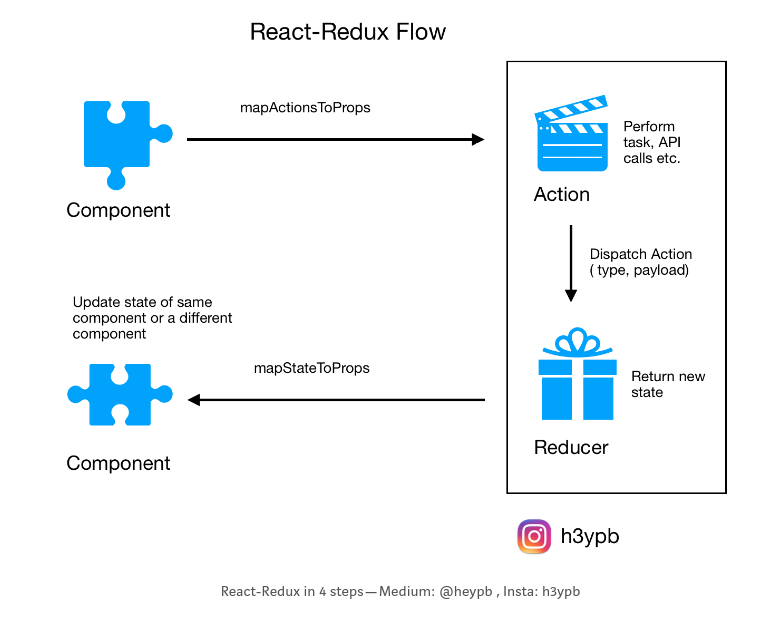
Webpack is a modular build tool that has two sets of functionality — Loaders and Plugins.

* **Loaders** transform the source code of a module. For example, style-loader adds CSS to DOM using style tags. sass-loader compiles SASS files to CSS. babel-loader transpiles JS code given the presets.
* **Plugins** are the core of Webpack. They can do things that loaders can’t. For example, there is a plugin called UglifyJS that minifies and uglifies the output of webpack.

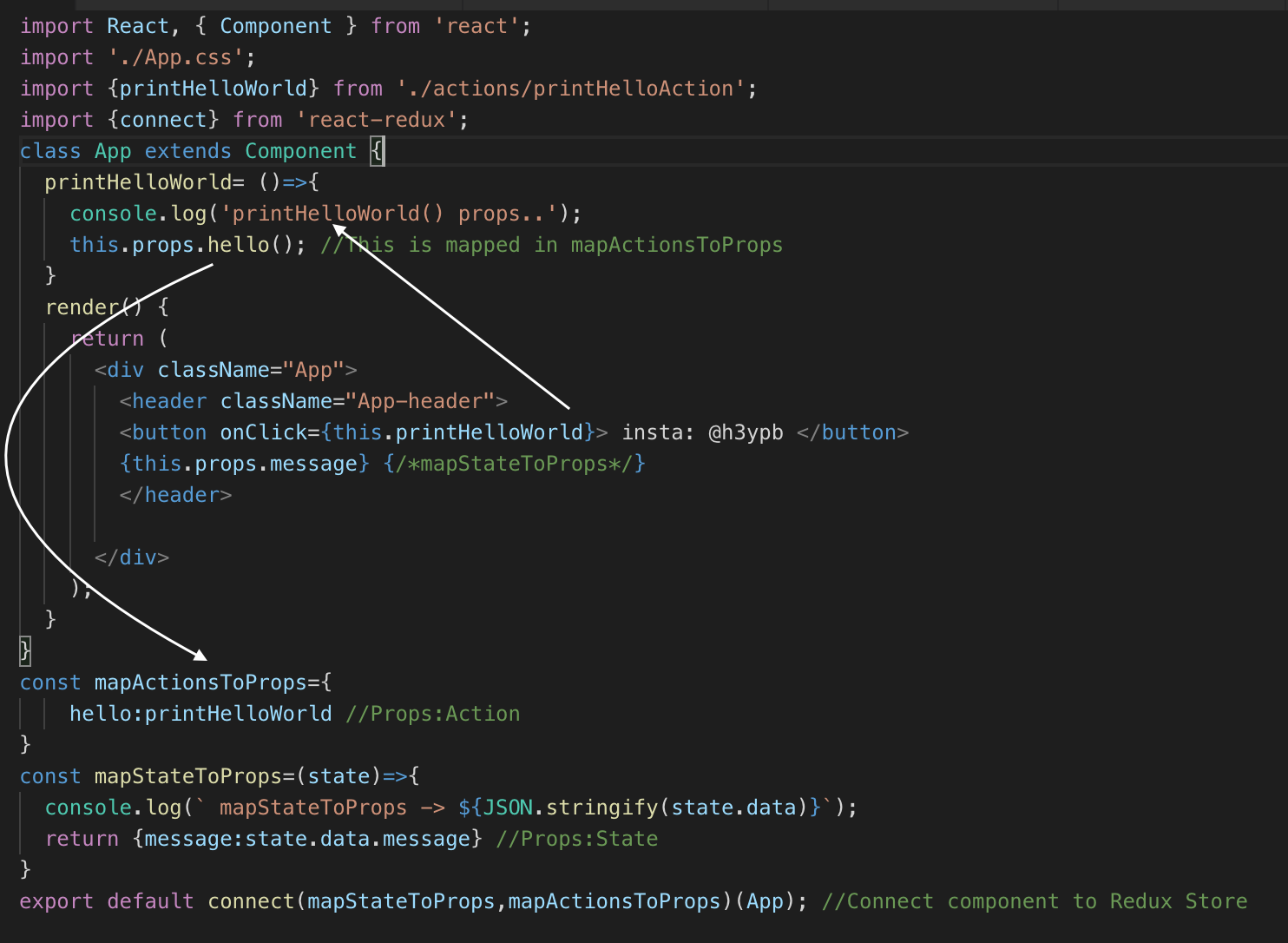
**React – Redux Architecture**

Here’s what a normal react-redux app’s folder structure looks like :

****

****

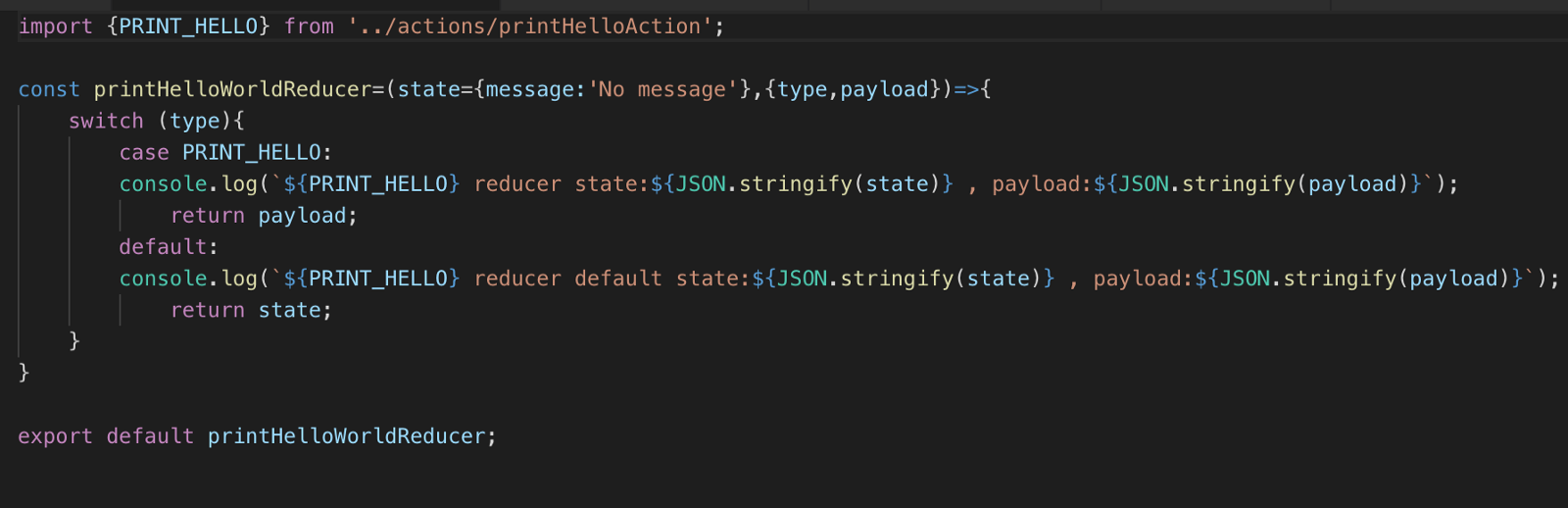
**1.**  A user interacts with a frontend component. A prop is used to call a function which initiates an action. Actions are initiated using props as they are mapped to props in an object called mapActionsToProps ( or it can be called anything ), which tells the react application which prop to be used to initiate an action.

****

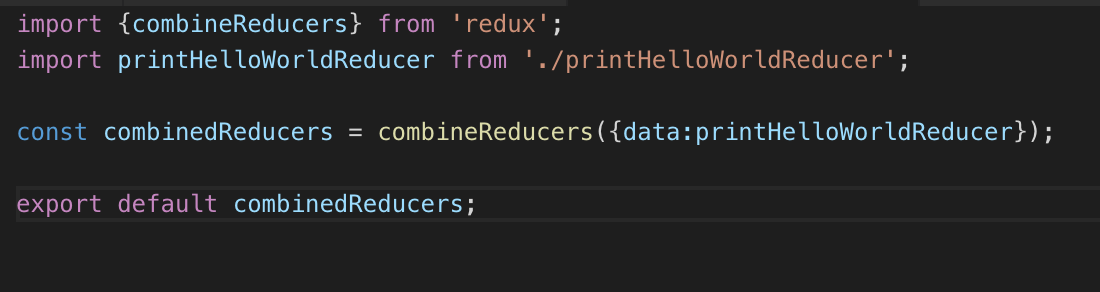
**2.** An action usually contains a type ( or identifier ) and payload ( or data ) . It can also perform tasks such as fetching data from APIs. The code below shows what an action might look like :

****

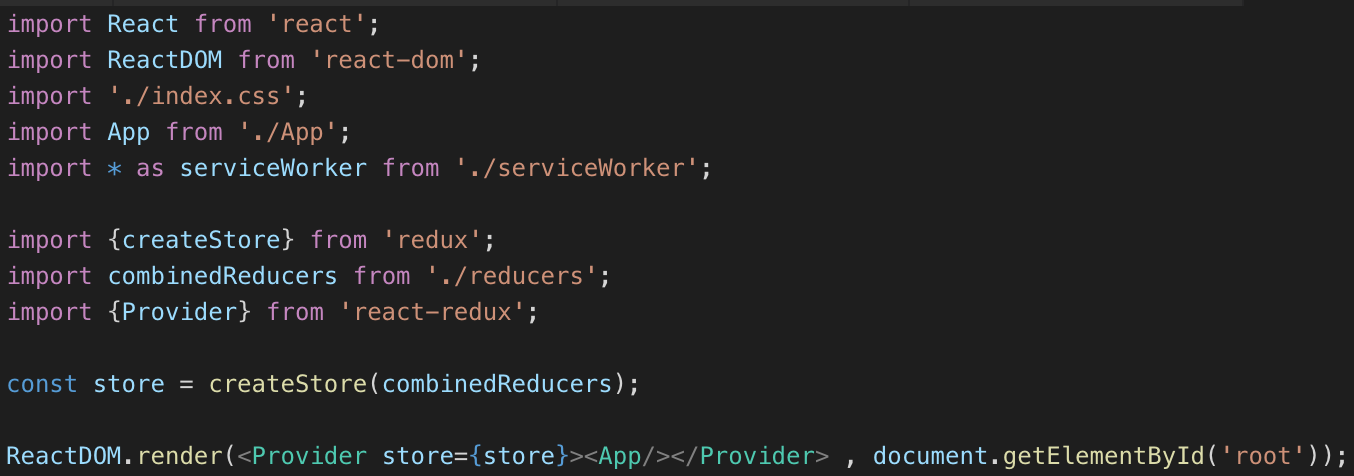
**3.** Once an action is dispatched, it is received by a reducer. A reducer’s job is to return the changed state. Depending on the type of of the action, a reducer may return an altered state of the component.

****

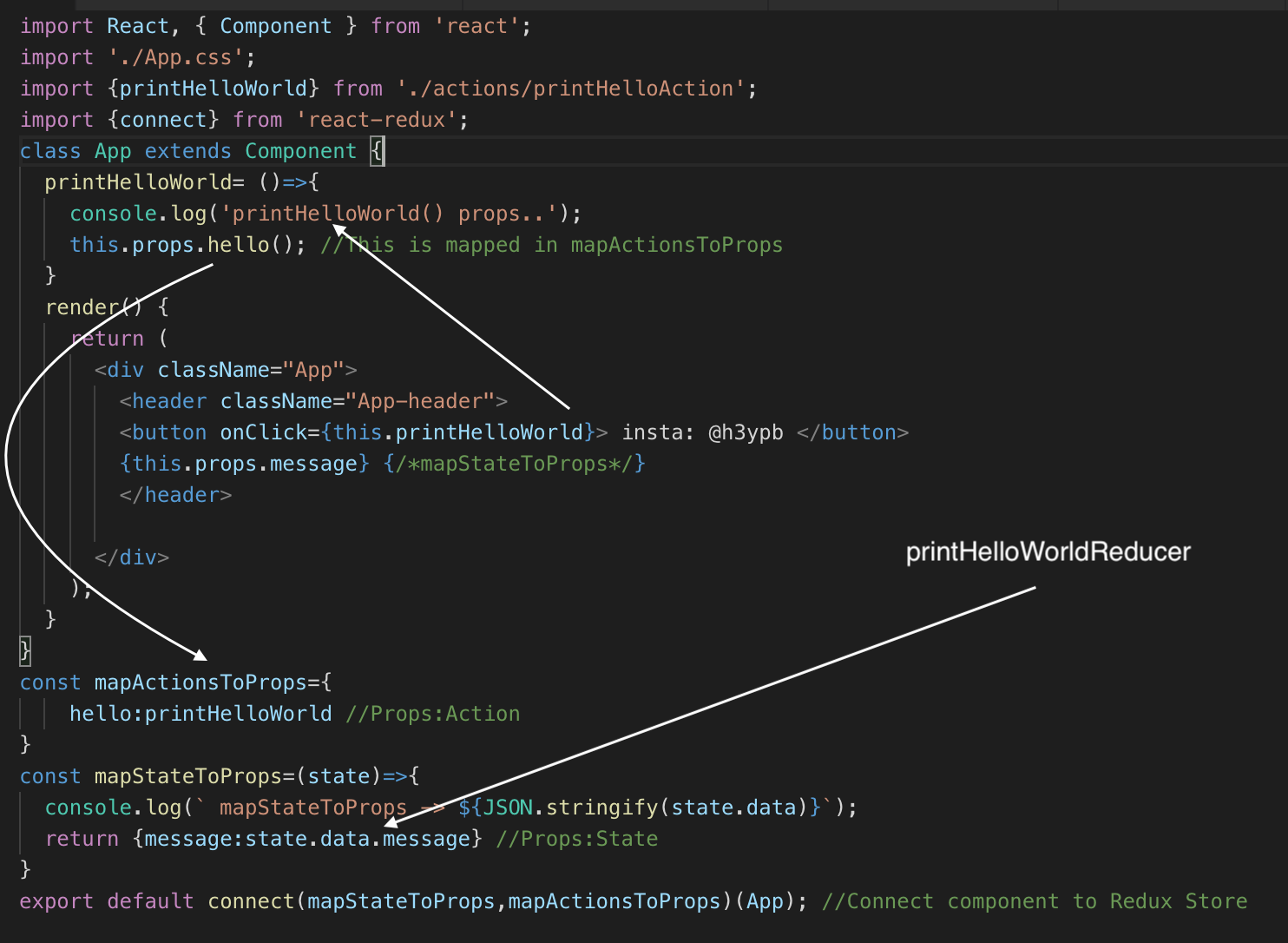
All reducers of a react app are combined into one single reducer which is passed as an argument to the redux store.

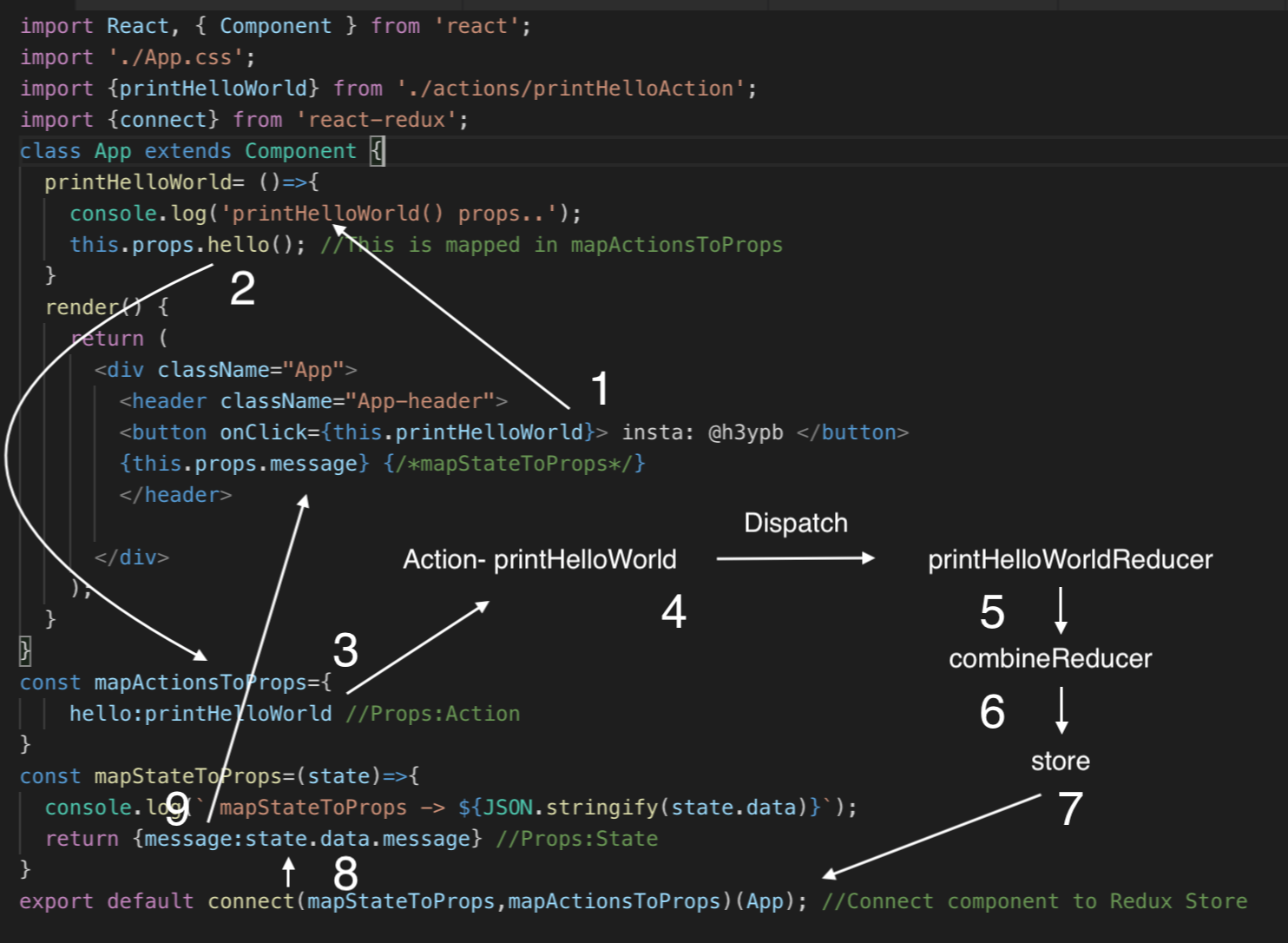


The redux store is provided to the react application in the root component.



**4.** The change in state is used to show the updated component. The states are mapped to the props of the component, in the mapStateToProps function. The defined props in this function can be used accordingly to update the components.

****

****

**Notes :**

1. Reducers must be pure functions. Given any input, output must always be the same.

2. The state of your whole application is stored in an object tree within a single store.

3. State is read only. The only way to change the state is to emit an action, an object describing what happened.

4. Changes in state are made with pure functions (reducers).

**Using Create React App**

* <https://facebook.github.io/create-react-app/docs/getting-started>
* [https://github.com/facebook/create-react-app#creating-an-app](https://github.com/facebook/create-react-app" \l "creating-an-app)

Create React App is an officially supported way to create single page react applications. It offers a modern build setup with no configuration.

**Quick Start**

npx create-react-app my-app

cd my-app

npm start

When you are ready to deploy it to production, create a minified bundle with npm run build.

**Advantages**

You don’t need to install or configure tools like Webpack or Babel. They are preconfigured and hidden so that you can focus on the code.

Just create the project and you’re good to go.

**npx**

npx create-react-app my-app

*npx comes with npm 5.2+ and higher*

**npm**

npm init react-app my-app

**yarn**

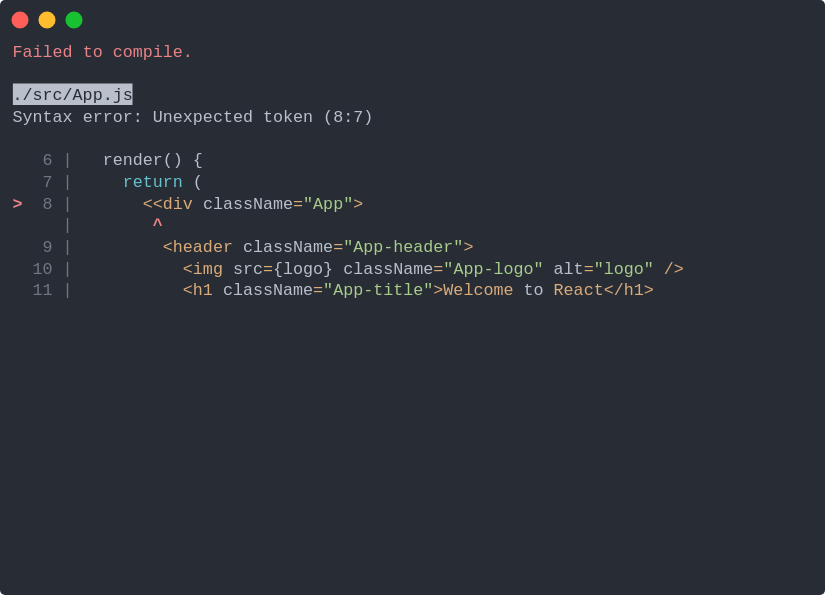
yarn create react-app my-app

**Scripts**

Inside the newly created project, you can run some built-in commands.

npm start OR yarn start

The page will automatically reload if you make changes to the code. You will see the build errors and lint warnings in the console.



**npm test OR yarn test**

Runs the test watcher in an interactive mode. By default, run tests related to the files changed since the last commit.

**npm run build OR yarn build**

Builds the app for production to the build folder. It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes. ( for checking if any changes occured in the file )

Your application is ready to be deployed.

For this course :

1. We first install our node for npm and package management. Installing node also helps us in setting up the development server for our application.

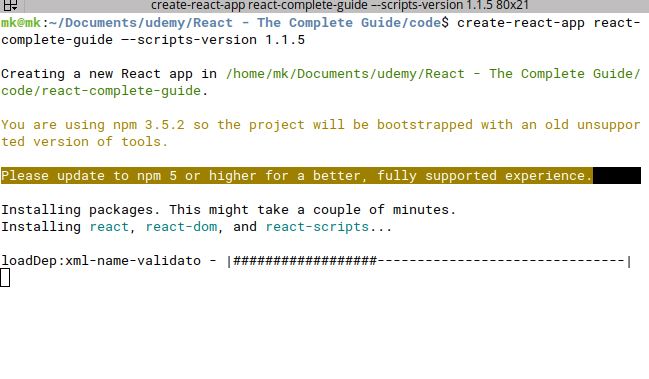
sudo npm install create-react-app -g

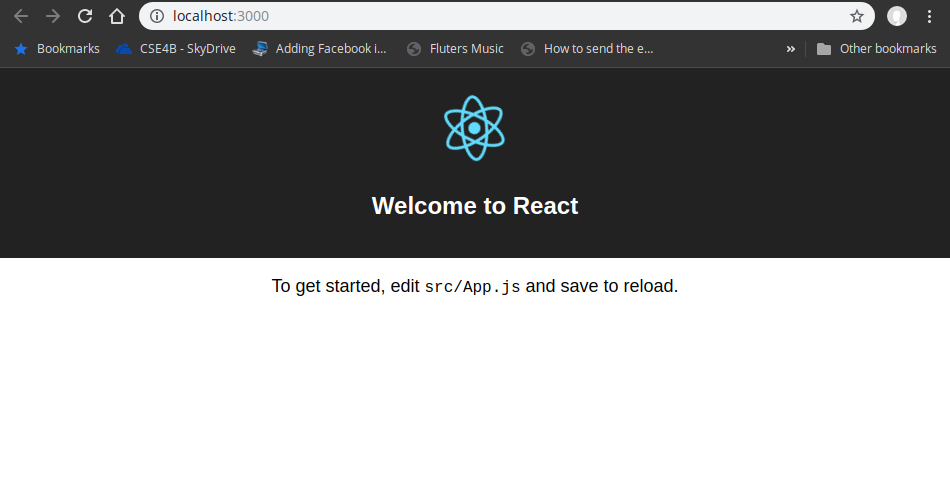
create-react-app is the CLI for create a new react application. We are installing it globally so that we can setup our react application from any location.

Installing node and npm : [https://linuxize.com/post/how-to-install-node-js-on-ubuntu-18.04/#install-node-js-and-npm-from-the-ubuntu-repository](https://linuxize.com/post/how-to-install-node-js-on-ubuntu-18.04/" \l "install-node-js-and-npm-from-the-ubuntu-repository)

create-react-app react-complete-guide –-scripts-version 1.1.5

// the –scripts-version helps with the folder structure.

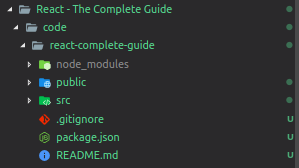




**Understanding the Folder Structure**

<https://facebook.github.io/create-react-app/docs/folder-structure>

After successfully setting up the react-app, following files will be there. Everything is create by create-react-app.



**Package.json**

{

"name": "react-complete-guide",

"version": "0.1.0",

"private": true,

"dependencies": {

"react": "^16.8.6",

"react-dom": "^16.8.6",

"react-scripts": "0.9.5"

},

"devDependencies": {},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test --env=jsdom",

"eject": "react-scripts eject"

}

}

For the project to build, the following files must exists with exact filenames :

* public/index.html is the page template
* src/index.js is the javascript entry point

You can delete or rename other files.

You may create subdirectories inside src. For faster rebuilds, only files inside src are processed by webpack. You need to put any JS and CSS files inside src, otherwise webpack won’t see them.

Only files inside public can be used from public/index.html.

You can, however, create more top-level directories. They will not be included in the production build so you can use them for things like documentation.

If you have GIT installed and your project is not part of a larger repository, then a new repository will be initialized resulting in an additional top-level .git directory.

**index.html**

This is the SPA template. If you want MPA to have multiple html templates then you will have to have multiple react apps or a different workflow. The root element in index.html is where the scripts insert the final DOM.

<body>

<div id="root"></div>

<!--

This HTML file is a template.

If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.

The build step will place the bundled scripts into the <body> tag.

To begin the development, run `npm start`.

To create a production bundle, use `npm run build`.

-->

</body>

**Understanding the component basics**

* Components let you split the UI into dependent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components.
* Conceptually, components are like javascript functions. They accept arbitrary inputs (called ‘props’) and return React elements describing what should appear on the screen.
* The simplest way to define a component is to write a javascript function. Such components are called function components.

**Understanding JSX**

**JSX Restrictions**

**Creating a functional component**

**Working with components & re-using them**

**Outputting dynamic content**

**Working with props**

**Understanding the children property**

**Understanding & using state**

**Props & State**

**Handling events with Methods**

**To which events can you listen?**

**Manipulating the state**

**Function Components Naming**

**Using the useState() hook for manipulation**

**Stateless vs Statefull Components**

**Passing method refereneces between components**

**Adding two way binding**

**Adding styling with stylesheets**

**Working with inline styles**

**Assignment 1**

**Useful resources & Links**