# React Environment Set Up

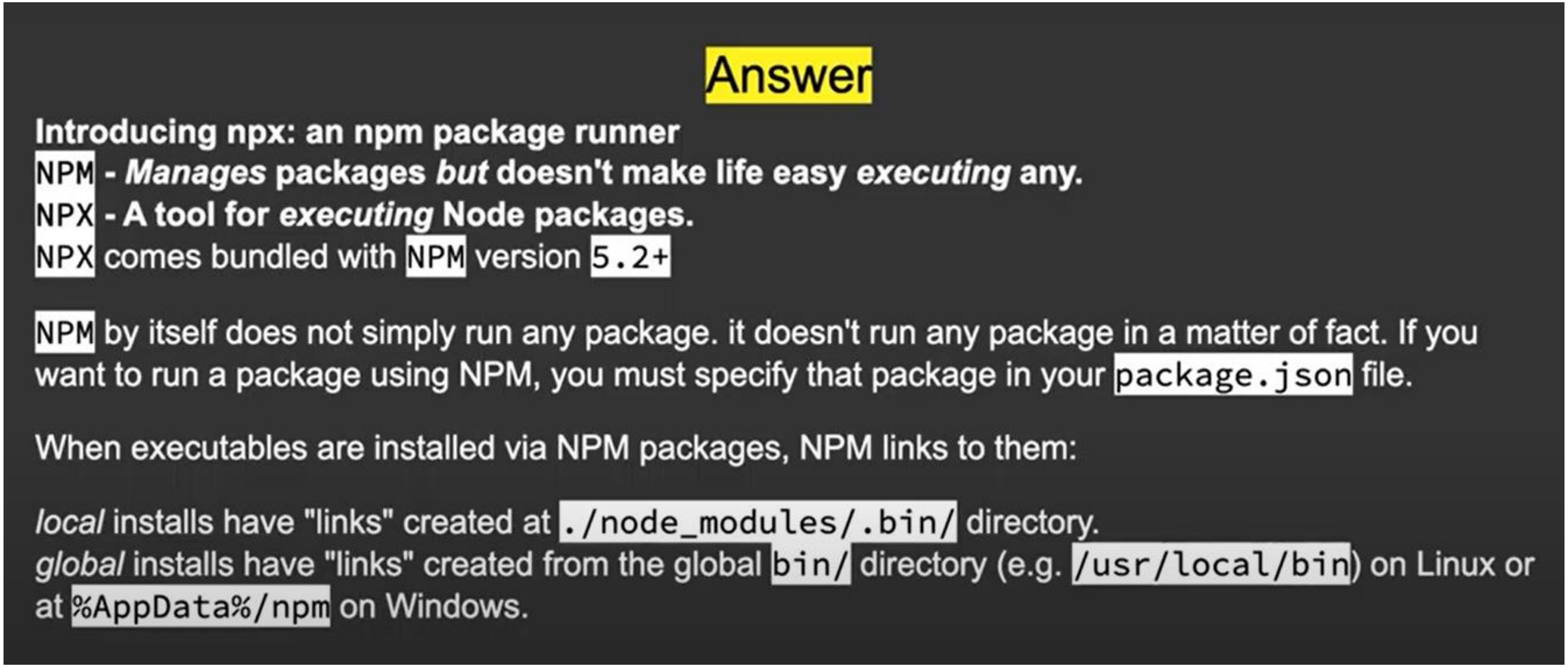
## Install Node JS, IDE and Npm

- Recommended Node JS and npm for using ReactJS in your system/ PC.
- npm - node package manager Open-source developers use npm to share software.
- npm is the world's largest Software Library with 8 Lakhs code packages.
- npm is installed with Node.js
- This means that you have to install Node.js to get npm installed on your computer.

## How to Create React App

- You must have Nodejs 8.10 above version and npm 5.6 above version in your system.
- After install it for creating a react app you command the below line in your command prompt.
- npx create-react-app my-app
- cd my-app
- npm start
- Editing your react code in visual studio code command the below line in your command prompt.
- cd
- my-app code.

## Why NPX ?



Answer

**Introducing npx: an npm package runner**
`NPM` **- *Manages* packages *but* doesn't make life easy *executing* any.**
`NPX` **- A tool for *executing* Node packages.**
`NPX` comes bundled with `NPM` version `5.2+`

`NPM` by itself does not simply run any package. it doesn't run any package in a matter of fact. If you want to run a package using NPM, you must specify that package in your `package.json` file.

When executables are installed via NPM packages, NPM links to them:

*local* installs have "links" created at `./node_modules/.bin/` directory.
*global* installs have "links" created from the global `bin/` directory (e.g. `/usr/local/bin`) on Linux or at `%AppData%/npm` on Windows.

**NPM:**
One might install a package locally on a certain project:

```
npm install some-package
```

Now let's say you want NodeJS to execute that package from the command line:

```
$ some-package
```

The above will **fail**. Only **globally installed** packages can be executed by typing their name *only*.

To fix this, and have it run, you must type the local path:

```
$ ./node_modules/.bin/some-package
```

You can technically run a locally installed package by editing your `packages.json` file and adding that package in the `scripts` section:

```
{
  "name": "whatever",
  "version": "1.0.0",
  "scripts": {
    "some-package": "some-package"
  }
}
```

Then run the script using `npm run-script` (or `npm run`):

```
npm run some-package
```

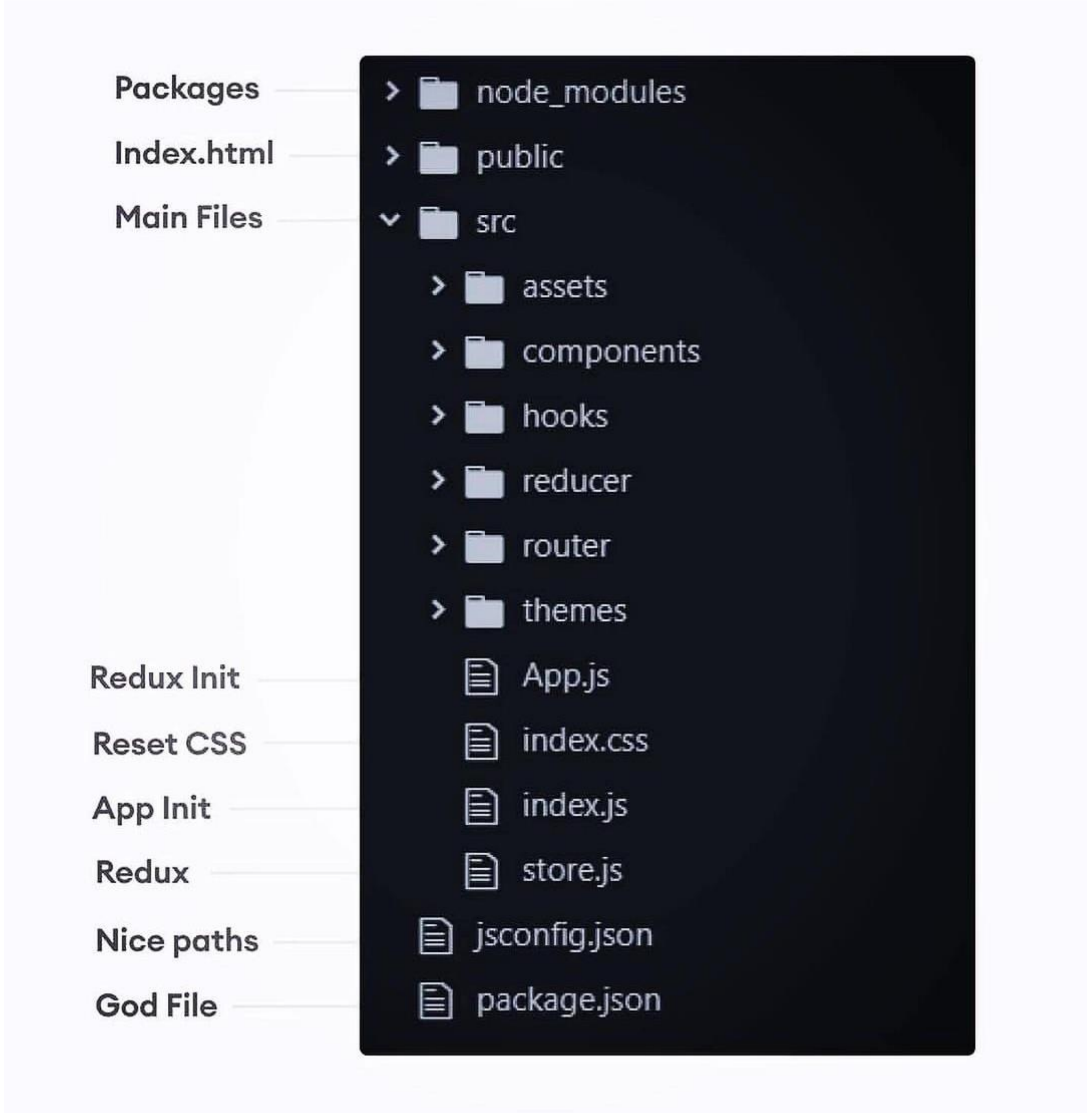
**NPX:**
`npx` will check whether `<command>` exists in `$PATH`, or in the local project binaries, and execute it. So, for the above example, if you wish to execute the locally-installed package `some-package` all you need to do is type:

```
npx some-package
```

Another **major** advantage of `npx` is the ability to execute a package which wasn't previously installed:

```
$ npx create-react-app my-app
```

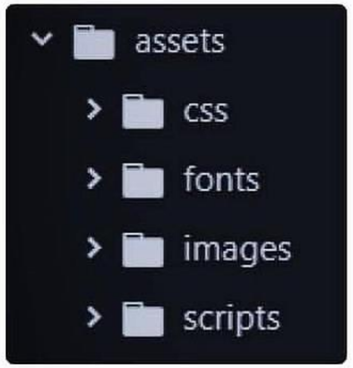
The above example will generate a `react` app boilerplate *within* the path the command had run in, and ensures that you always use the latest version of a generator or build tool without having to upgrade each time you're about to use it.



```
> node_modules
> public
v src
    > assets
    > components
    > hooks
    > reducer
    > router
    > themes
      App.js
      index.css
      index.js
      store.js
  jsconfig.json
  package.json
```

## Assets

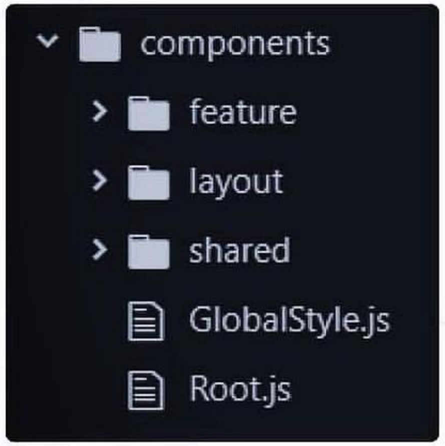
This is where we store extra bits like JS scripts and CSS. Note, I try to use this folder sparingly and instead prefer to CDN images and fonts



```
assets
  css
  fonts
  images
  scripts
```

## Components

This is where the majority of work happens. At the bottom we have our main Root component that will hold our application together

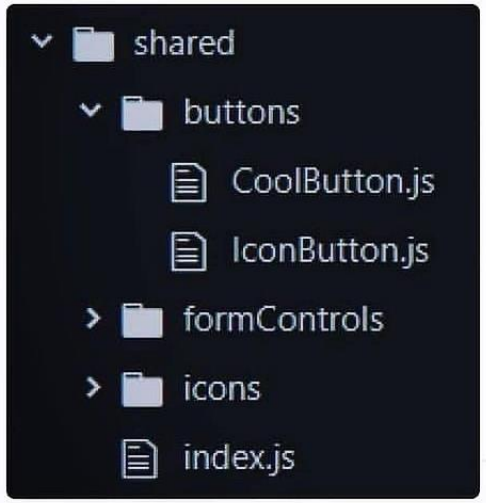


```
components
  feature
  layout
  shared
  GlobalStyle.js
  Root.js
```

body styles

## Shared Components

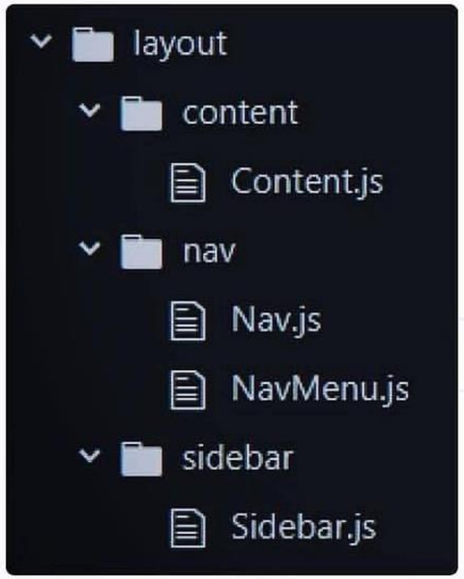
These folders include our essentially "dumb" components. Their composition relies on props being passed in but they can access the theme



```
shared
  buttons
    CoolButton.js
    IconButton.js
  formControls
  icons
  index.js
```

## Layout Components

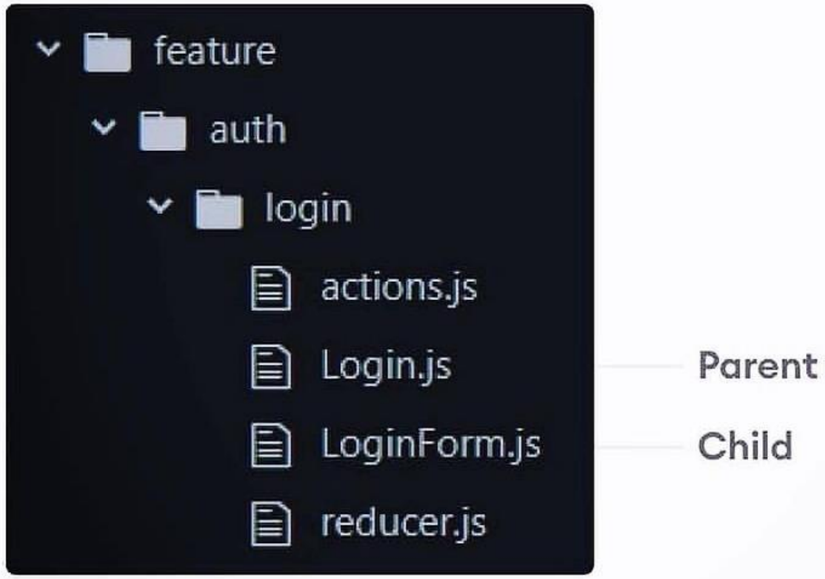
These folders include all of the components that make up the master layout of your app. Note, a loyout component can include a feature



```
layout
  content
    Content.js
  nav
    Nav.js
    NavMenu.js
  sidebar
    Sidebar.js
```

## Feature Components

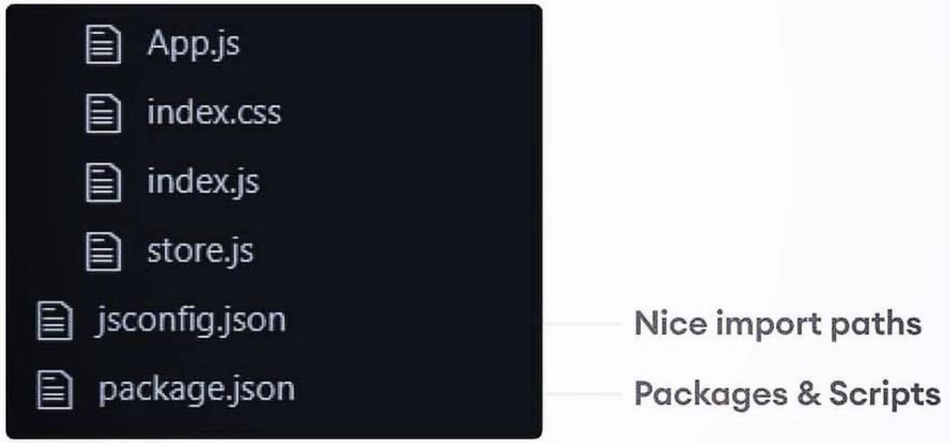
These folders include everything you need to make a page in your app. If I'm using redux, I like them to have their own actions and reducer



```
feature
  auth
    login
      actions.js
      Login.js
      LoginForm.js
      reducer.js
```

## Royalty 👑

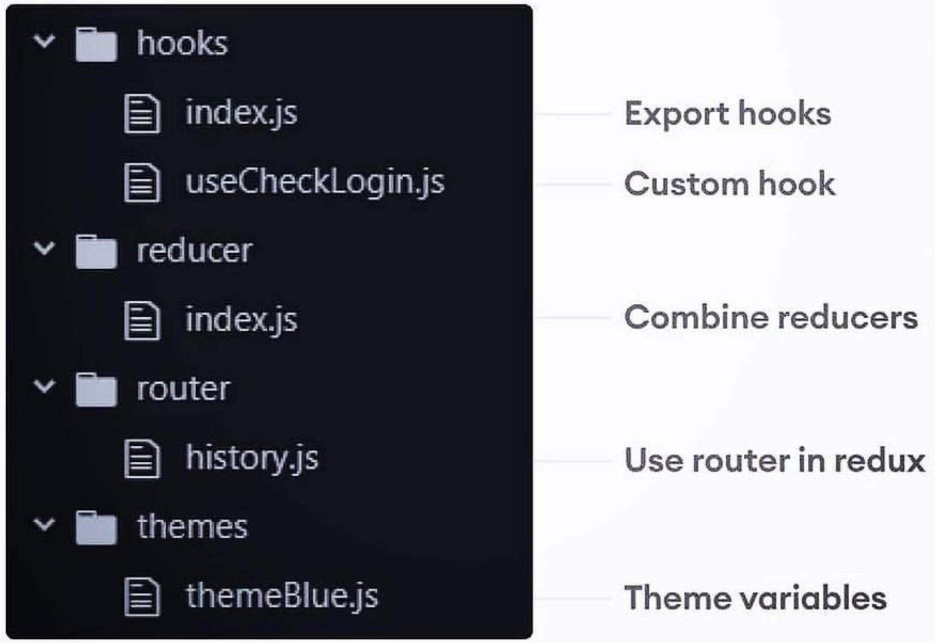
All essential files for our app that we will hardly ever need to touch. Package.json will can change though as we add packages, settings and scripts



```
    App.js
    index.css
    index.js
    store.js
jsconfig.json          Nice import paths
package.json           Packages & Scripts
```

## VIP's

Hooks are a great way to keep code clean, and the themes folder is our CSS in JS variables. The others are involved in reducer and router setup



```
hooks
    index.js           Export hooks
    useCheckLogin.js   Custom hook
reducer
    index.js           Combine reducers
router
    history.js         Use router in redux
themes
    themeBlue.js       Theme variables
```

## SUMMARY

- Environment set up
- Npm
- Npx
- Npx Advantage
- Folder structure
- First React Application

QUERIES