# Guide — The Multi CRA Lerna Monorepo

Alistair MacDonald Follow
Oct 27, 2018 · 13 min read

> *In this guide you will learn how to scaffold a Monorepo to manage multiple Create React Apps that share common components, with a Storybook.*

## UPDATE: 2019/07/17

Please use *React Workspaces Playground*, instead of this guide. **React Workspaces Playground** is a supported codebase where you can test drive everything in this guide with the latest version of React and Storybook. This guide demonstrates how to achieve similar effects via monkey-patching - a bad practice that is difficult to maintain. Please review my *React Workspaces slides* for more information.

· · ·

**Featuring:**

- 🎸 **Babel-Loader-Lerna-CRA** — Auto-transpile, hot-reload Lerna modules!
- 🐉 **Lerna** — The Monorepo manager
- ⚛️ **Create-React-App-2** — React 16 App Scaffolding (unejected)
- 📖 **Storybook-4-React** — Component Storybook
- 🃏 **Jest** — Unit/Snapshot Testing

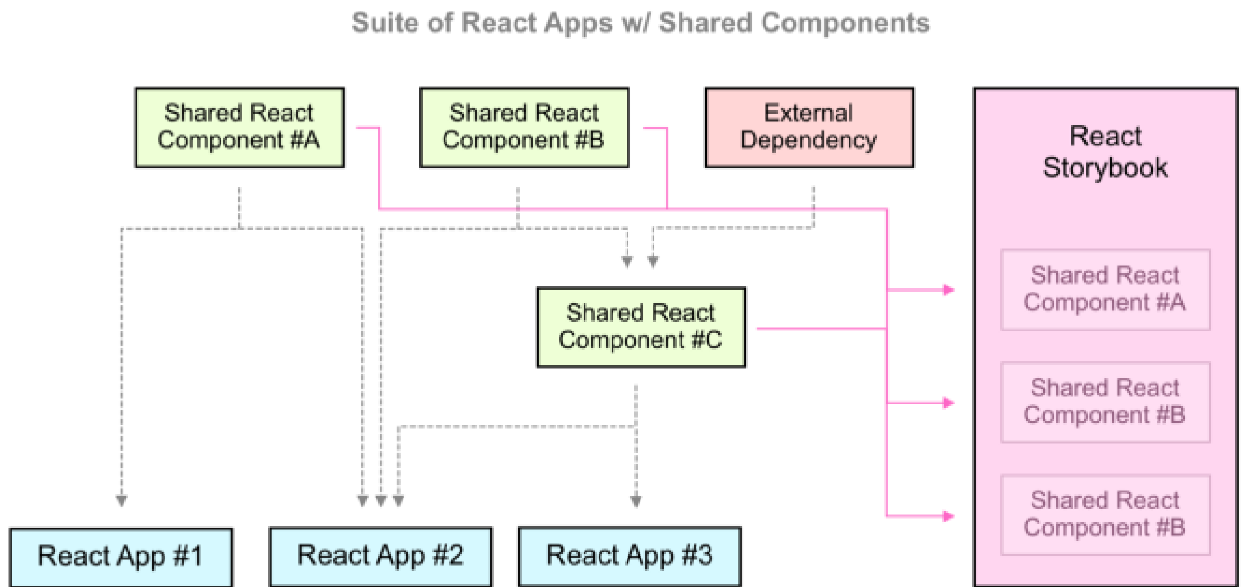
## TL;DR — Give Me The Code!

> Ok… f1lt3r/monorepo-react.

> ⚠️ ***IMPORTANT STEPS***
>
> *If you are checking out this code to test without using this guide, please remember to follow these important steps:*
>
> 1. `lerna booststrap`
> 2. `npx babel-loader-lerna-cra`

## The Case For Monorepos

Imagine a scenario where you are building a suite of three React apps that share the same architecture, design patterns, components, and styles. Now imagine making an update to a low-level component like a Button that is used in all three apps, as well as one sub-component.


Suite of React Apps w/ Shared Components
Shared React Component #A
Shared React Component #B
External Dependency
React Storybook
Shared React Component #C
Shared React Component #A
Shared React Component #B
Shared React Component #B
React App #1
React App #2
React App #3

In this scenario, you would be forced into a process like this:

1. Update the Button code in the Button’s git repository.
   *(Component B in the diagram above)*
2. Create a first Pull Request *(#1)* in the Component B repo and review the new code into master.
3. Publish Component B’s Button code on a public or private NPM service.
4. Go into the React repo that uses the Button and update the package.json dependencies.
5. Create a second Pull Request *(#2)* in the Component C repo and get that new code reviewed into master.
6. Publish the component to the NPM repo.

### Go into `React App #1`

1. Update the dependencies.
2. Republish the package on npm service.
3. Submit a new PR. *(#3)*
4. Deploy

### Go into `React App #2`

1. Update the dependencies.
2. Republish the package on npm service.
3. Submit a new PR. *(#4)*
4. Deploy

### Go into `React App #3`

1. Update the dependencies.
2. Republish the package on npm service.

3. Submit a new PR. *(#5)*

4. Deploy

**That is five pull requests for a change to one button component!**

Clearly, this is less than ideal.

### A Simpler Solution

Now imagine using a single repo for the same update. If we use a Monorepo tool like Lerna, the update process will look more like this:

1. Update the Button code in the Button's git directory. (`Component #B` in the diagram above)

2. Run `lerna bootstrap` to crosslink the Button `Component #B` into all the sub-dependencies.

3. Run `lerna publish` to update the packages in your private NPM service.

4. Create a Pull Request in the `Monorepo` repo and get the new code into `master`.

5. Re-deploy the apps with the updated `package.json` version numbers.

Now everything is done in one Pull Request.

This is why large organizations like Facebook and Google make good use of Monorepos. This process can be simplified to use a single shared repo for all the dependencies and apps. The Monorepo scales up without losing as much engineering velocity and reduces human error lost from switching contextual focus.

The following guide will show you how to set up a such Monorepo for a React project.

## Prerequisites

```
$ npm i -g lerna

$ npm i -g create-react-app
```

Create a directory for your Monorepo project.

```
$ cd ~/repos
$ mkdir monorepo-react
$ cd monorepo-react
```

## Setup Lerna

Create and initialize your Lerna monorepo:

```
$ lerna init
```

Your `package.json` should now look like this:

```
{
  "name": "root",
  "private": true,
  "devDependencies": {
    "lerna": "^3.4.3"
  }
}
```

## Install Common Dependencies

Installing these common dependencies will allow you to:

- Run Storybook for the root of your project.
- To have Storybook auto-install the right modules for your React project.
- Have Babel transpile correctly for code, testing and Storybook.

```
$ npm i -D react react-dom @babel/core@^7.0.0-0 @babel/cli babel-
plugin-transform-es2015-modules-commonjs babel-jest enzyme enzyme-
adapter-react-16 jest react-test-renderer babel-core@7.0.0-bridge.0
@babel/preset-env @babel/preset-react
```

Your `package.json` should now look like this:

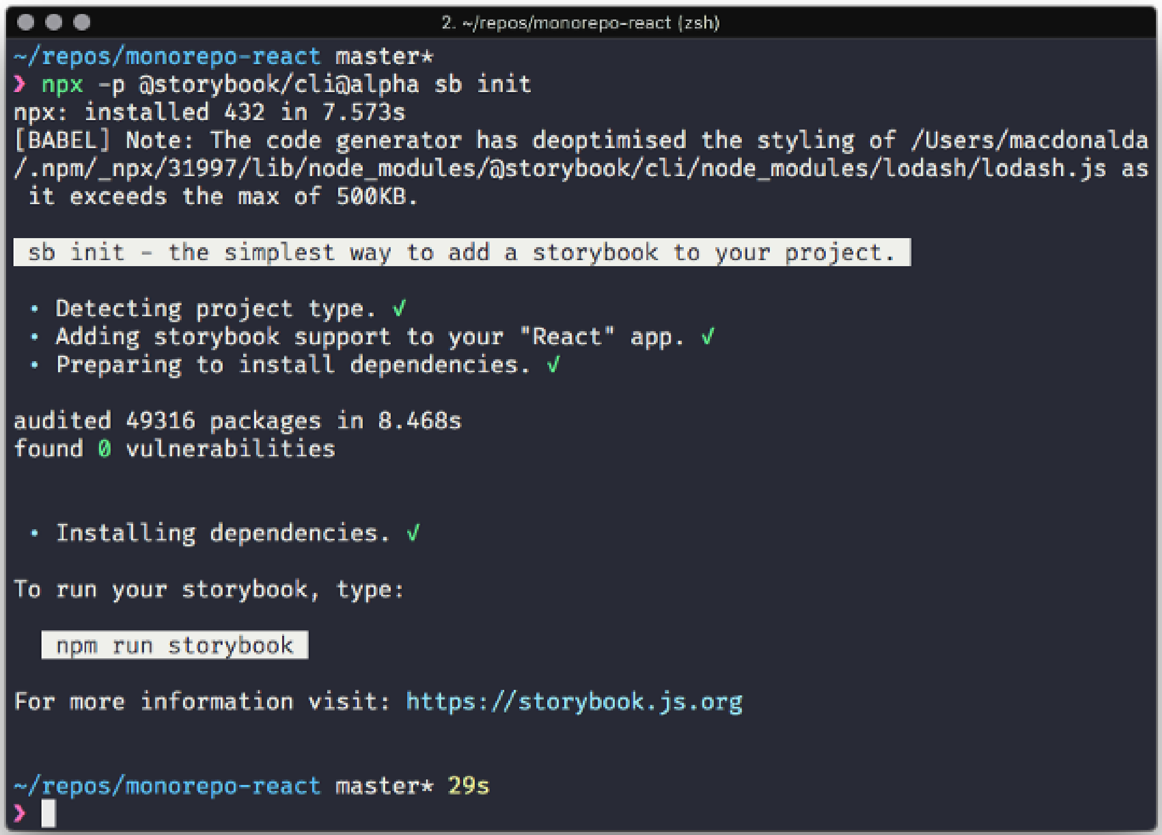
```
{
  "name": "root",
  "private": true,
  "devDependencies": {
    "@babel/cli": "^7.1.2",
    "@babel/core": "^7.1.2",
    "@babel/preset-env": "^7.1.0",
    "@babel/preset-react": "^7.0.0",
    "babel-core": "^7.0.0-bridge.0",
    "babel-jest": "^23.6.0",
    "babel-plugin-transform-es2015-modules-commonjs": "^6.26.2",
    "enzyme": "^3.7.0",
    "enzyme-adapter-react-16": "^1.6.0",
    "jest": "^23.6.0",
    "lerna": "^3.4.3",
    "react": "^16.6.0",
    "react-dom": "^16.6.0",
    "react-test-renderer": "^16.6.0"
  }
}
```

## Install Storybook React

Now we will install and initialize Storybook version 4.

```
$ npx -p @storybook/cli@alpha sb init
```

Note: Installing the `@alpha` version (currently `@4.0.0-rc.6`), will allow us to set our Babel configuration inside of our `package.json` files which will make configuration easier for sub-packages.



```
2. ~/repos/monorepo-react (zsh)
~/repos/monorepo-react master*
❯ npx -p @storybook/cli@alpha sb init
npx: installed 432 in 7.573s
[BABEL] Note: The code generator has deoptimised the styling of /Users/macdonalda
/.npm/_npx/31997/lib/node_modules/@storybook/cli/node_modules/lodash/lodash.js as
 it exceeds the max of 500KB.

 sb init - the simplest way to add a storybook to your project. 

 • Detecting project type. ✓
 • Adding storybook support to your "React" app. ✓
 • Preparing to install dependencies. ✓

audited 49316 packages in 8.468s
found 0 vulnerabilities


 • Installing dependencies. ✓

To run your storybook, type:

   npm run storybook 

For more information visit: https://storybook.js.org


~/repos/monorepo-react master* 29s
❯ 
```

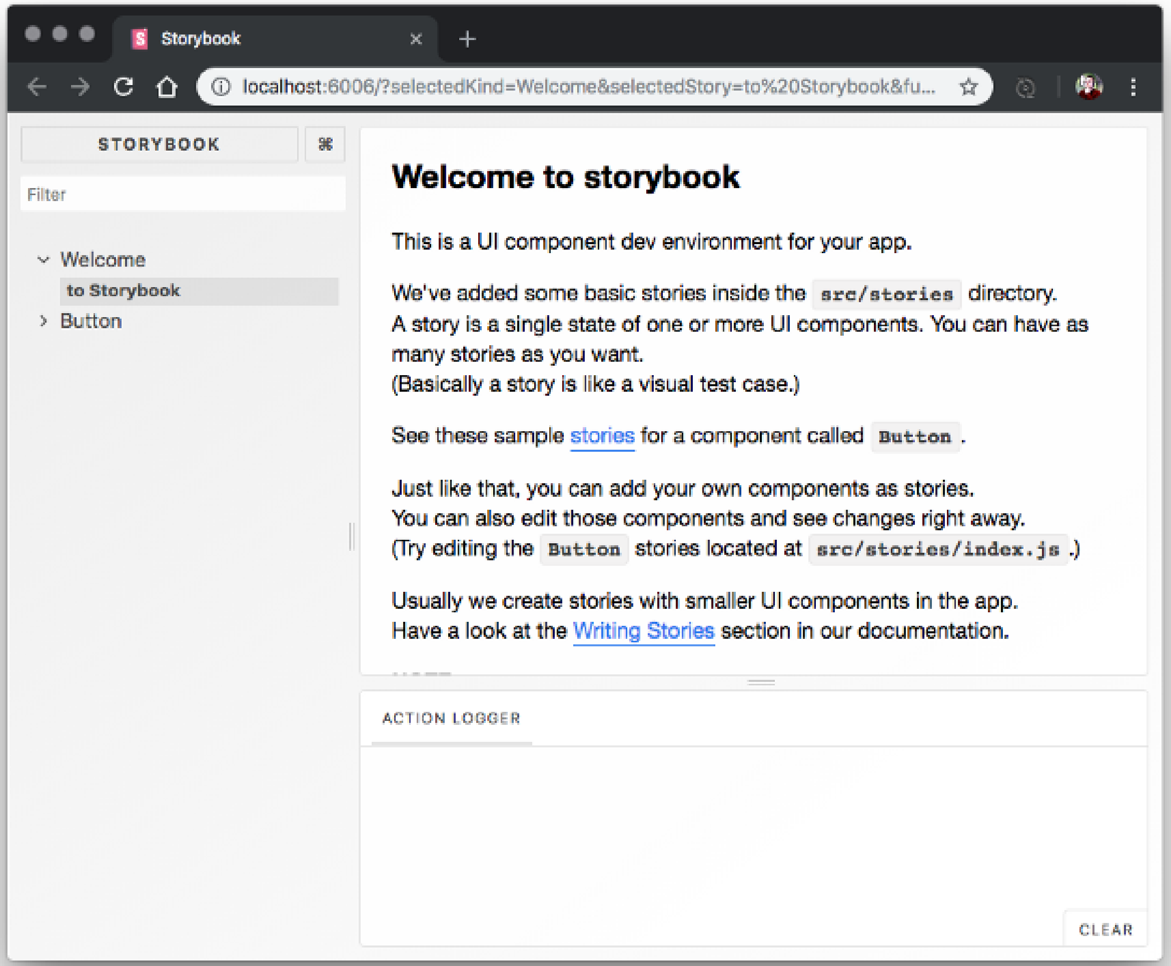
Your root `package.json` file should now look like this:

```
{
  "name": "root",
  "private": true,
  "devDependencies": {
    "@babel/cli": "^7.1.2",
    "@babel/core": "^7.1.2",
    "@babel/preset-env": "^7.1.0",
    "@babel/preset-react": "^7.0.0",
    "babel-core": "^7.0.0-bridge.0",
    "babel-jest": "^23.6.0",
    "babel-plugin-transform-es2015-modules-commonjs": "^6.26.2",
    "enzyme": "^3.7.0",
    "enzyme-adapter-react-16": "^1.6.0",
    "jest": "^23.6.0",
    "lerna": "^3.4.3",
    "react": "^16.6.0",
    "react-dom": "^16.6.0",
    "react-test-renderer": "^16.6.0",
    "@storybook/react": "^4.0.0-alpha.25",
    "@storybook/addon-actions": "^4.0.0-alpha.25",
    "@storybook/addon-links": "^4.0.0-alpha.25",
    "@storybook/addons": "^4.0.0-alpha.25",
    "babel-loader": "^8.0.4"
  },
  "dependencies": {},
  "scripts": {
    "storybook": "start-storybook -p 6006",
    "build-storybook": "build-storybook"
  }
}
```

Now you can test that Storybook runs on your machine.

```
$ npm run storybook
```

Storybook should now launch in your web browser automatically.


Storybook
localhost:6006/?selectedKind=Welcome&selectedStory=to%20Storybook&fu...
STORYBOOK
⌘
Filter
Welcome
to Storybook
Button
ACTION LOGGER
CLEAR

Welcome to storybook

This is a UI component dev environment for your app.

We've added some basic stories inside the `src/stories` directory.
A story is a single state of one or more UI components. You can have as many stories as you want.
(Basically a story is like a visual test case.)

See these sample stories for a component called `Button`.

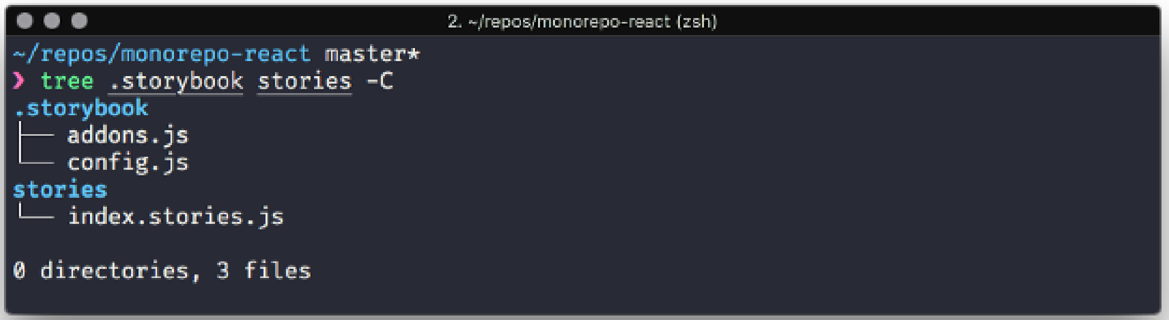
Just like that, you can add your own components as stories.
You can also edit those components and see changes right away.
(Try editing the `Button` stories located at `src/stories/index.js`.)

Usually we create stories with smaller UI components in the app.
Have a look at the Writing Stories section in our documentation.

List the storybook files:

```
$ tree -C .storybook stories
```

- Your `.storybook/` directory contains your Storybook configuration.
- Your `stories/` directory is where your global Storybook stories live.



```
~/repos/monorepo-react master*
❯ tree .storybook stories -C
.storybook
├── addons.js
└── config.js
stories
└── index.stories.js

0 directories, 3 files
```

> *Note: To install tree: [ `brew` / `apt-get` / `yum` / `pkg` ] `install tree`*

## Create Your React App

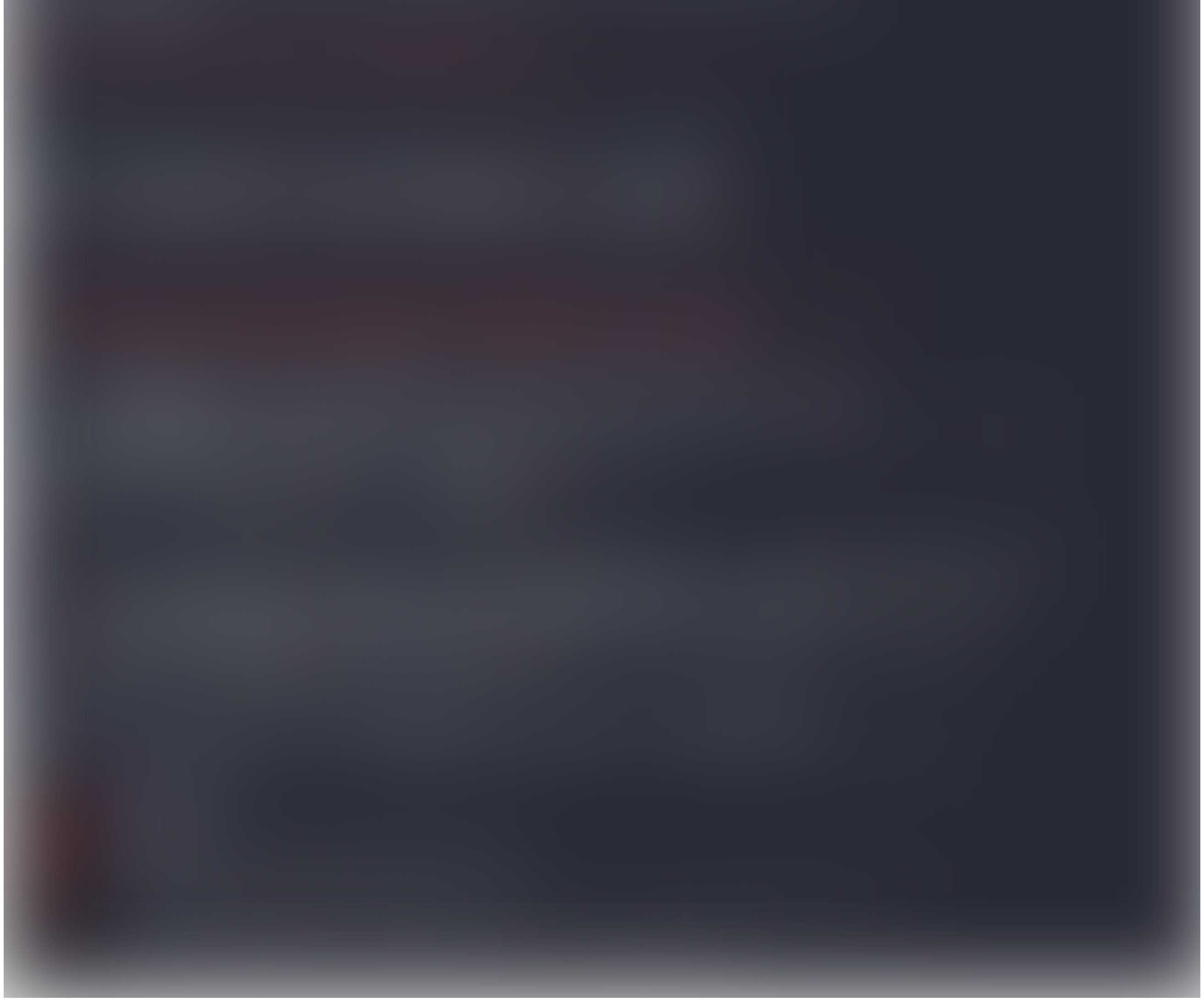
Create a home in `packages/my-react-app` for your React App.

```
$ cd ~/repos/monorepo-react/packages/
$ create-react-app my-react-app
```

Run your React app to test things worked.

```
$ cd my-react-app
$ npm run start
```

You should now see an error message about Webpack like this one:



We will work around this by setting the `SKIP_PREFLIGHT_CHECK=true` in the `.env` file, as suggested.

```
echo "SKIP_PREFLIGHT_CHECK=true" > .env
```

You should now be able to run your React app, and your browser should launch automatically.

```
$ npm run start
```

### Create an External React Component

Lets create our first external React component. We will do this inside our `./packages` directory provided by Lerna.

```
$ cd ~/repos/monorepo-react/packages/
$ mkdir comp-button
$ cd comp-button
```

Create a `packages/comp-button/package.json` file like this:

```
{
  "name": "@project/comp-button",
  "version": "0.1.0",
  "description": "A simple button component",
  "main": "dist/index.js",
  "module": "src/index.js",
  "scripts": {
    "transpile": "babel src -d dist --ignore
'**/*.spec.js,**/*.stories.js'",
    "jest": "jest --coverage --verbose --color"
  },
  "babel": {
    "presets": [
      "@babel/preset-env",
      "@babel/preset-react"
    ],
    "env": {
      "test": {
        "plugins": [
          "transform-es2015-modules-commonjs"
        ]
      }
      }
    }
  }
```

What is going on in the `package.json` file:

- `name` : The organizational namespace for your component when installing via NPM or cross-linked Lerna.
- `main` : The compiled code that will be shipped with the build of your React app.
- `module` : The pre-compiled code that will be imported as a local run-time dependency while developing the app or running tests.
- `transpile` : An NPM script start the transpile of your code with Babel. **Note:** We are not using `build` because we want to reserve this word later to build our React apps with `lerna run build` .
- `babel` : This setup configures our component to transpile with Babel 7 for React.

> *Note: Because we installed components like* `react` *,* `react-dom` *,* `@babel/core@^7.0.0-0` *in our root* `package.json` *we do not have to install them again in this package.*

Make a source directory for your React component.

```
$ mkdir src
$ cd src
```

Create your React component in `packages/comp-button/index.js` :

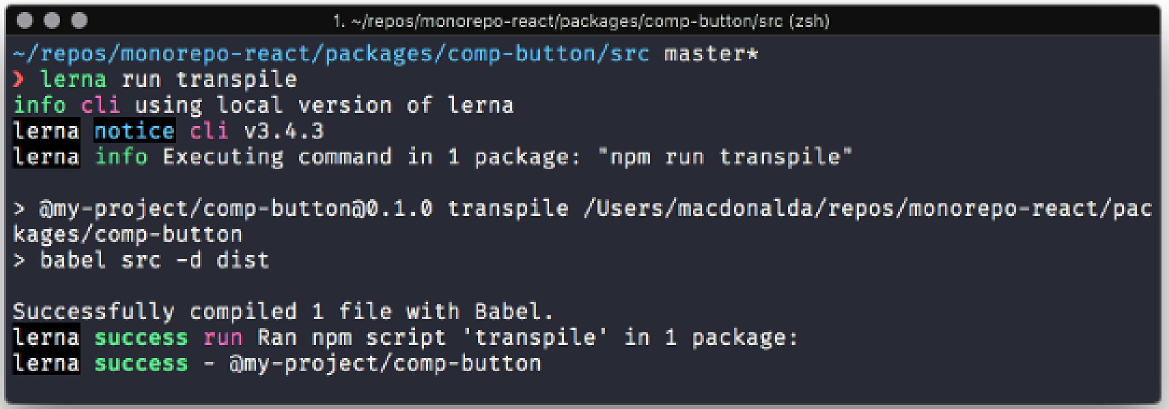
```
import React from 'react'

const Button = ({ type = 'button', children, onClick }) => (
    <div>
      <button type={type} className="button" onClick={onClick}>
        {children}
      </button>
  </div>
)

export default Button
```

## Transpile Your Component

Now let's try to transpile your React code to ECMAScript 2015 (JavaScript with support for older browsers).

```
$ lerna run transpile
```

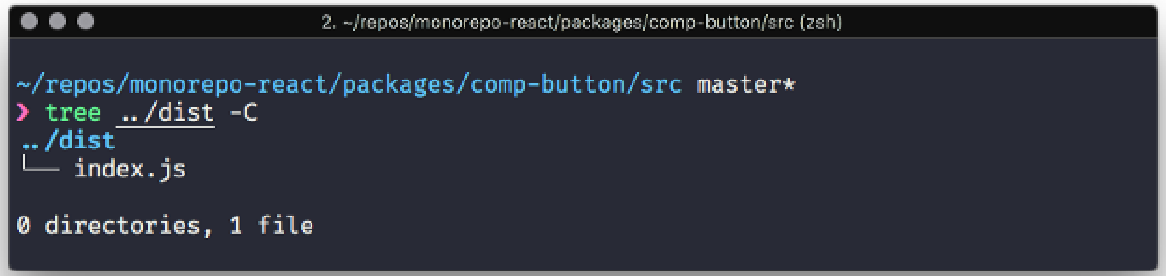
You should see the following output:



```
~/repos/monorepo-react/packages/comp-button/src master*
❯ lerna run transpile
info cli using local version of lerna
lerna notice cli v3.4.3
lerna info Executing command in 1 package: "npm run transpile"

> @my-project/comp-button@0.1.0 transpile /Users/macdonalda/repos/monorepo-react/pac
kages/comp-button
> babel src -d dist

Successfully compiled 1 file with Babel.
lerna success run Ran npm script 'transpile' in 1 package:
lerna success - @my-project/comp-button
```

Your `./dist/` directory should now contain the transpiled `index.js` file:

```
$ tree -C ../dist
```



```
~/repos/monorepo-react/packages/comp-button/src master*
❯ tree ../dist -C
../dist
└── index.js

0 directories, 1 file
```

The `./dist/index.js` file should contain your transpiled code, like this:

```
"use strict";

Object.defineProperty(exports, "__esModule", {
  value: true
});
exports.default = void 0;

var _react = _interopRequireDefault(require("react"));

function _interopRequireDefault(obj) { return obj && obj.__esModule ?
obj : { default: obj }; }

var Button = function Button(_ref) {
  var _ref$type = _ref.type,
      type = _ref$type === void 0 ? 'button' : _ref$type,
      children = _ref.children,
      onClick = _ref.onClick;
  return _react.default.createElement("div", null,
_react.default.createElement("button", {
    type: type,
    className: "button",
    onClick: onClick
  }, children));
};

var _default = Button;
exports.default = _default;
```

## Test Your Component

While we are here, lets create a Jest spec for your component in `packages/comp-button/src/index.spec.js`:

```
import React from 'react';
import {mount} from 'enzyme';
import Button from '.';

describe('Button Component', function() {
  it('renders without props', function() {
    const wrapper = mount(<Button />);
    const button = wrapper.find('.button');
    expect(button.length).toBe(1);
  })

  it('renders without props', function() {
    const wrapper = mount(<Button />);
    const button = wrapper.find('.button');
    expect(button.length).toBe(1);
  })

  it('renders children when passed in', () => {
    const wrapper = mount(
      <Button>
        <p className="child">Some Child</p>
      </Button>
    );

    const child = wrapper.find('.child')
    expect(child.length).toBe(1)
  })

  it('handles onClick events', () => {
    const onClick = jest.fn()
    const wrapper = mount(
      <Button onClick={onClick} />
    )

    wrapper.find('button').simulate('click')

    expect(onClick.mock.calls.length).toBe(1)
  })
})
```

> *Note: we installed `babel-core@7.0.0-bridge.0` and `babel-jest` earlier to make Babel 7 code compatible with Jest. (See: "Install Common Dependencies" above.)*

Add the following "jest" section to your root `package.json`:

```
"jest": {
    "setupFiles": [
      "../../setupTests"
    ]
  }
```

Your `packages/comp-button/package.json` should now look like this:

```
{
    "name": "@my-project/comp-button",
    "version": "0.1.0",
    "description": "A simple button component",
    "main": "dist/index.js",
    "module": "src/index.js",
    "scripts": {
    "transpile": "babel src -d dist --ignore
'**/*.spec.js,**/*.stories.js'",
      "jest": "jest --coverage --verbose --color"
    },
    "babel": {
      "presets": [
        "@babel/preset-env",
        "@babel/preset-react"
      ],
      "env": {
        "test": {
          "plugins": [
            "transform-es2015-modules-commonjs"
          ]
        }
      }
    },
    "jest": {
      "setupFiles": [
        "../../setupTests"
      ]
    }
  }
```

When Jest runs, `../../setupTests` file will reference `setupTests.js` in your Monorepo root.

Let’s add this `setupTests.js` file with some Enzyme helpers:

```
const enzyme = require('enzyme');
const Adapter = require('enzyme-adapter-react-16');
enzyme.configure({ adapter: new Adapter() });
```

> *Note: we deliberately use the older require syntax here, so that `setupTests.js` is loadable without additional babel configuration.*

Now let's run Jest to see the spec working:

```
lerna run jest
```

> *Note: We are using `jest` and not `test` to reserve the word "test" for running all tests, including End to End, linting, etc.*

## Add a Story for Your React Component

Now let's create a Storybook story for our new Button component:

Add the following code to `index.stories.js`:

```
import React from 'react'
import { storiesOf } from '@storybook/react'
import { action } from '@storybook/addon-actions'

import Button from '.'

storiesOf('Button', module)

  .add('with text', () => (
      <Button onClick={action('clicked')}>Button</Button>
  ))

  .add('with some emoji', () => (
      <Button onClick={action('clicked')}>😀 😎 👍 💯</Button>
  ))

  .add('with a theme provider', () => (
    <Button onClick={action('clicked')}>Button</Button>
  ))
```

### Reconfigure Storybook

We will now need to configure Storybook to load stories from all the `packages/**` directories, instead of loading the `stories/` directory from your Monorepo root.

Edit your Storybook configuration in `~/repos/monorepo-react/.storybook/config.js`, so it look like this:

```
import { configure } from '@storybook/react';

// automatically import all files ending in *.stories.js
const req = require.context('../packages', true, /.stories.js$/);
function loadStories() {
  req.keys().forEach(filename => req(filename));
}

configure(loadStories, module);
```

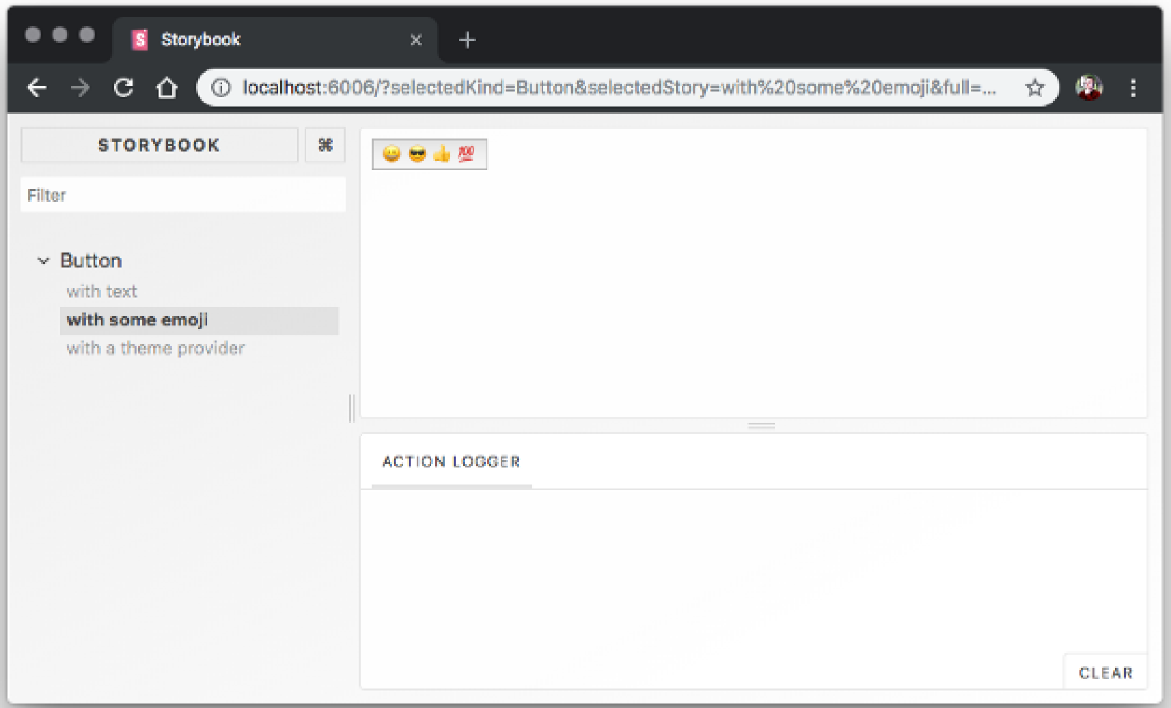
It’s now safe to delete the `stories/` directory at the Monorepo root.

```
$ cd ~/repos/monorepo-react/
$ sudo rm -r stories
```

Let's check that the Storybook still loads with your `comp-button` Story:

```
$ npm run storybook
```

You should now be able to see your button component Story which was built from your `packages/comp-button` directory:



## Crosslink Your Dependencies with Lerna

Add the following dependency to your `packages/my-react-app/package.json`:

```
{
  "dependencies": {
    "@my-project/comp-button": "*"
  }
}
```

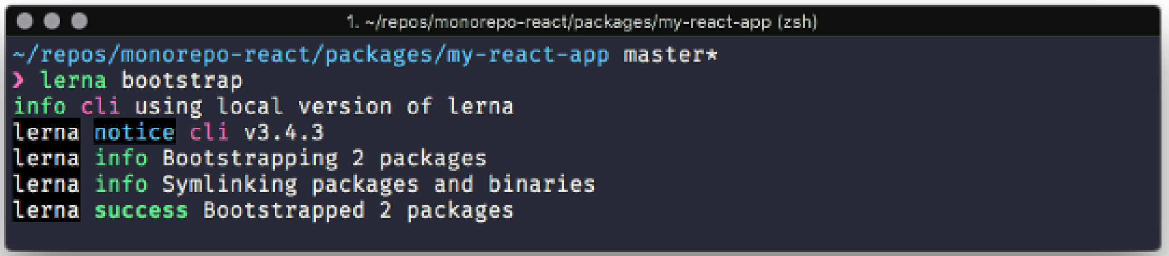
Your `packages/my-react-app/package.json` should now look like this:

```
{
  "name": "@my-project/my-react-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^16.6.0",
    "react-dom": "^16.6.0",
    "react-scripts": "2.0.5",
    "@my-project/comp-button": "*"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": [
    ">0.2%",
    "not dead",
    "not ie <= 11",
    "not op_mini all"
  ]
}
```

We can now crosslink our packages using `lerna bootstrap`.

```
$ lerna bootstrap
```

You should see the following success message:



```
~/repos/monorepo-react/packages/my-react-app master*
❯ lerna bootstrap
info cli using local version of lerna
lerna notice cli v3.4.3
lerna info Bootstrapping 2 packages
lerna info Symlinking packages and binaries
lerna success Bootstrapped 2 packages
```

## Use Your Component in The React App

Add the following lines to `packages/my-react-app/src/App.js`:

```
import CompButton from '@my-project/comp-button';
<CompButton>Foobar!</CompButton>
```

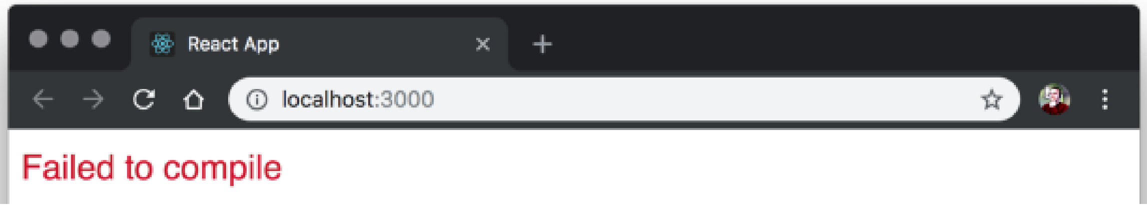
Your file will now look like this:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import CompButton from '@my-project/comp-button';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
          <CompButton>Foobar!</CompButton>
        </header>
      </div>
    );
  }
}

export default App;
```

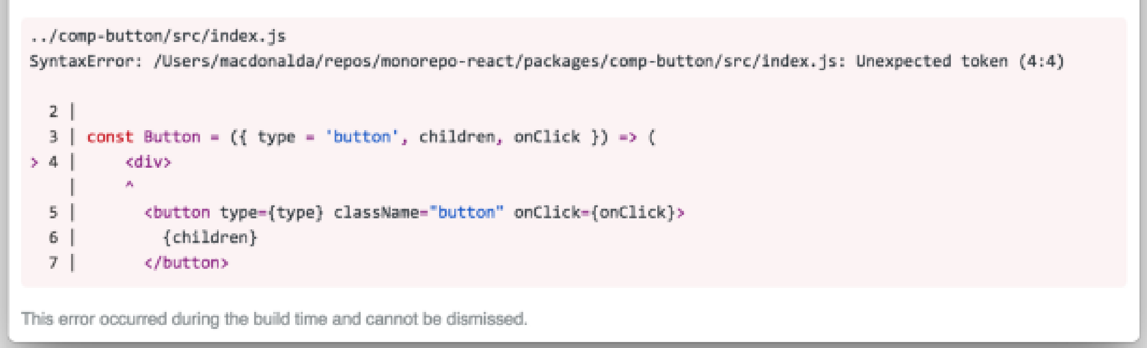
Now start your app:

```
$ npm run start
```

You should see the following error:



Failed to compile



```
../comp-button/src/index.js
SyntaxError: /Users/macdonalda/repos/monorepo-react/packages/comp-button/src/index.js: Unexpected token (4:4)

  2 |
  3 | const Button = ({ type = 'button', children, onClick }) => (
> 4 |     <div>
    |     ^
  5 |       <button type={type} className="button" onClick={onClick}>
  6 |         {children}
  7 |       </button>
```

This error occurred during the build time and cannot be dismissed.

The React App is failing to compile because Create-React-App's Webpack config is unaware of any external modules. This means Webpack cannot tell Babel-Loader about your component directories, and so the sources do not get transpiled.

It seems like this will problem may go away with future versions of Create-React-App, although this may require Yarn Workspaces. So make sure you check the GitHub Issue Create-React-App-Lerna-Support to see if this feature os landed before using the following workaround.

## Rewire Your React App for Lerna

I created a small workaround Node Module to override Create-React-App Webpack configs inside Lerna projects, called Babel-Loader-Lerna-CRA. It's pretty simple. It just updates the Webpack paths for Babel-Loader.

You can install this package using NPM:

```
npm i -D babel-loader-lerna-cra
```

Now let's update the `package.json` in our Lerna root with glob patterns that describe the relationship between our components and our app.

```
"babel-loader-lerna-cra": {
  "imports": "packages/comp-*/src",
  "apps":  "packages/*react-app*"
}
```

Your `package.json` should now look like this:

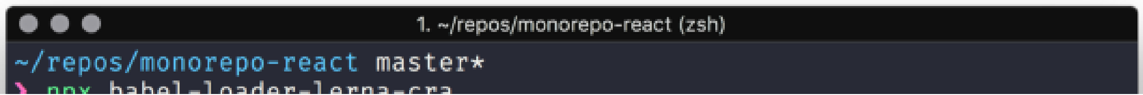
```
{
  "name": "root",
  "private": true,
  "devDependencies": {
    "@babel/cli": "^7.1.2",
    "@babel/core": "^7.1.2",
    "@babel/preset-env": "^7.1.0",
    "@babel/preset-react": "^7.0.0",
    "@storybook/addon-actions": "^4.0.0-alpha.25",
    "@storybook/addon-links": "^4.0.0-alpha.25",
    "@storybook/addons": "^4.0.0-alpha.25",
    "@storybook/react": "^4.0.0-alpha.25",
    "babel-core": "^7.0.0-bridge.0",
    "babel-jest": "^23.6.0",
    "babel-loader": "^8.0.4",
    "babel-loader-lerna-cra": "^0.1.2",
    "babel-plugin-transform-es2015-modules-commonjs": "^6.26.2",
    "enzyme": "^3.7.0",
    "enzyme-adapter-react-16": "^1.6.0",
    "jest": "^23.6.0",
    "lerna": "^3.4.3",
    "react": "^16.6.0",
    "react-dom": "^16.6.0",
    "react-test-renderer": "^16.6.0"
  },
  "dependencies": {},
  "scripts": {
    "storybook": "start-storybook -p 6006",
    "build-storybook": "build-storybook"
  },
  "babel-loader-lerna-cra": {
    "imports": "packages/comp-*/src",
    "apps":  "packages/*react-app*"
  }
}
```

- The **`imports`** refer to components that the React app will need to transpile.
- The **`apps`** inform `babel-loader-lerna-cra` where the Webpack overrides will need to happen.

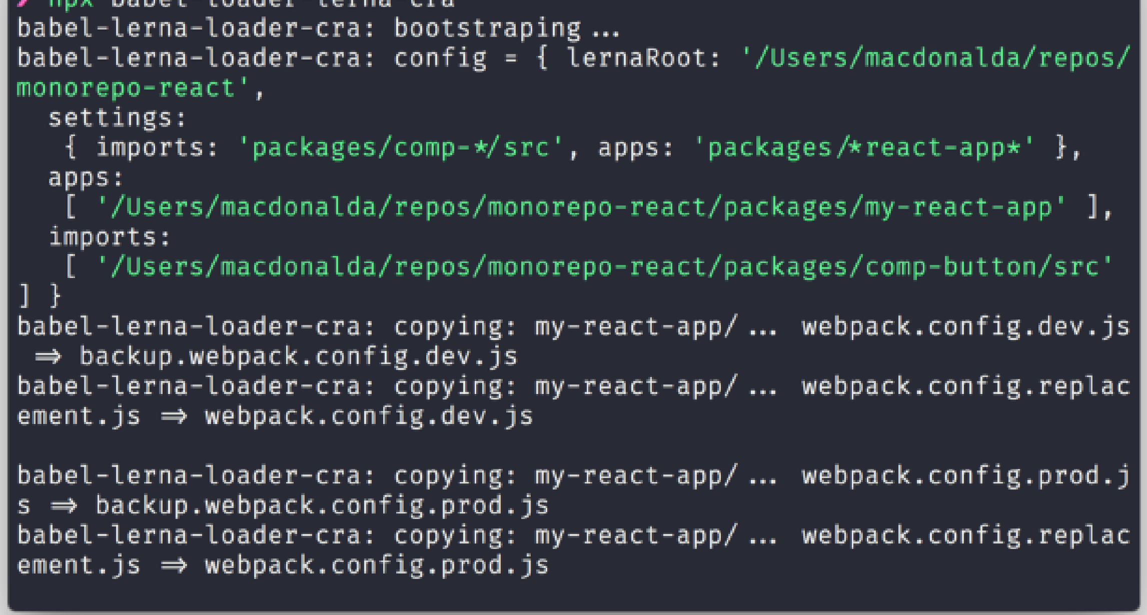
Now lets bootstrap the Webpack configs in our React app with `babel-loader-lerna-cra`:

```
$ npx babel-loader-lerna-cra
```

You should see the following output:



```
~/repos/monorepo-react master*
```

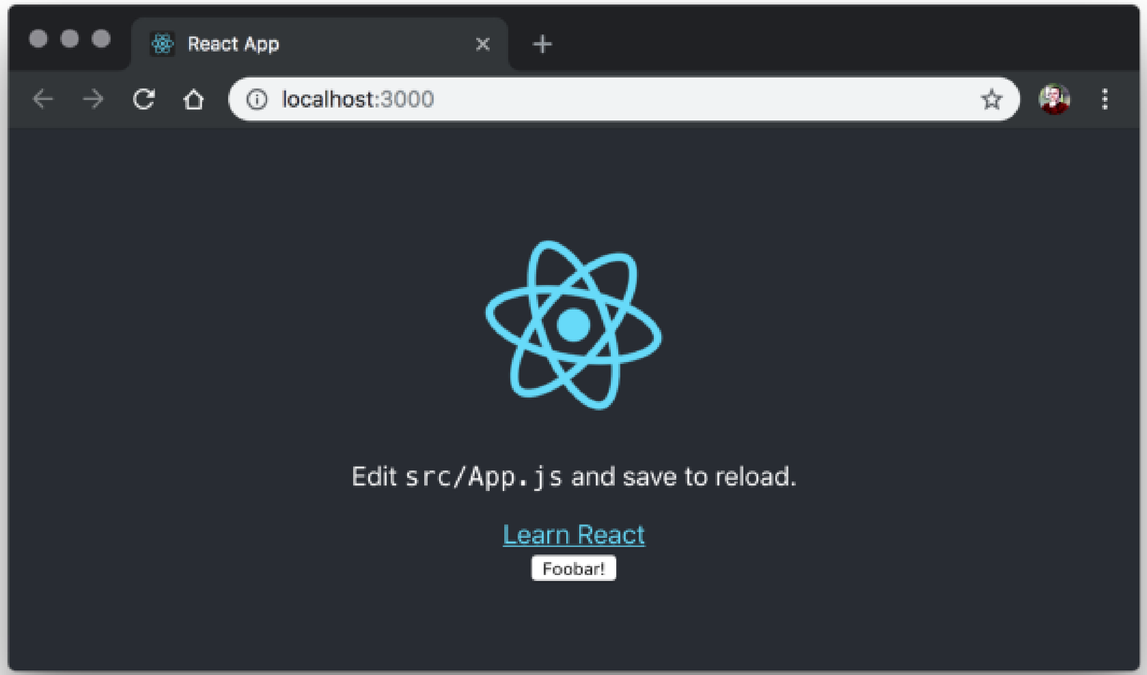


```
❯ npx babel-loader-lerna-cra
babel-lerna-loader-cra: bootstraping ...
babel-lerna-loader-cra: config = { lernaRoot: '/Users/macdonalda/repos/
monorepo-react',
  settings:
   { imports: 'packages/comp-*/src', apps: 'packages/*react-app*' },
  apps:
   [ '/Users/macdonalda/repos/monorepo-react/packages/my-react-app' ],
  imports:
   [ '/Users/macdonalda/repos/monorepo-react/packages/comp-button/src'
] }
babel-lerna-loader-cra: copying: my-react-app/ ... webpack.config.dev.js
 => backup.webpack.config.dev.js
babel-lerna-loader-cra: copying: my-react-app/ ... webpack.config.replac
ement.js => webpack.config.dev.js

babel-lerna-loader-cra: copying: my-react-app/ ... webpack.config.prod.j
s => backup.webpack.config.prod.js
babel-lerna-loader-cra: copying: my-react-app/ ... webpack.config.replac
ement.js => webpack.config.prod.js
```

Now let's try running your React App again:

```
$ cd ~/repos/monorepo-react/packages/my-react-app
$ npm run start
```

You should now see the React App launch in a browser with your `CompButton` component rendering with the text "Foobar!"



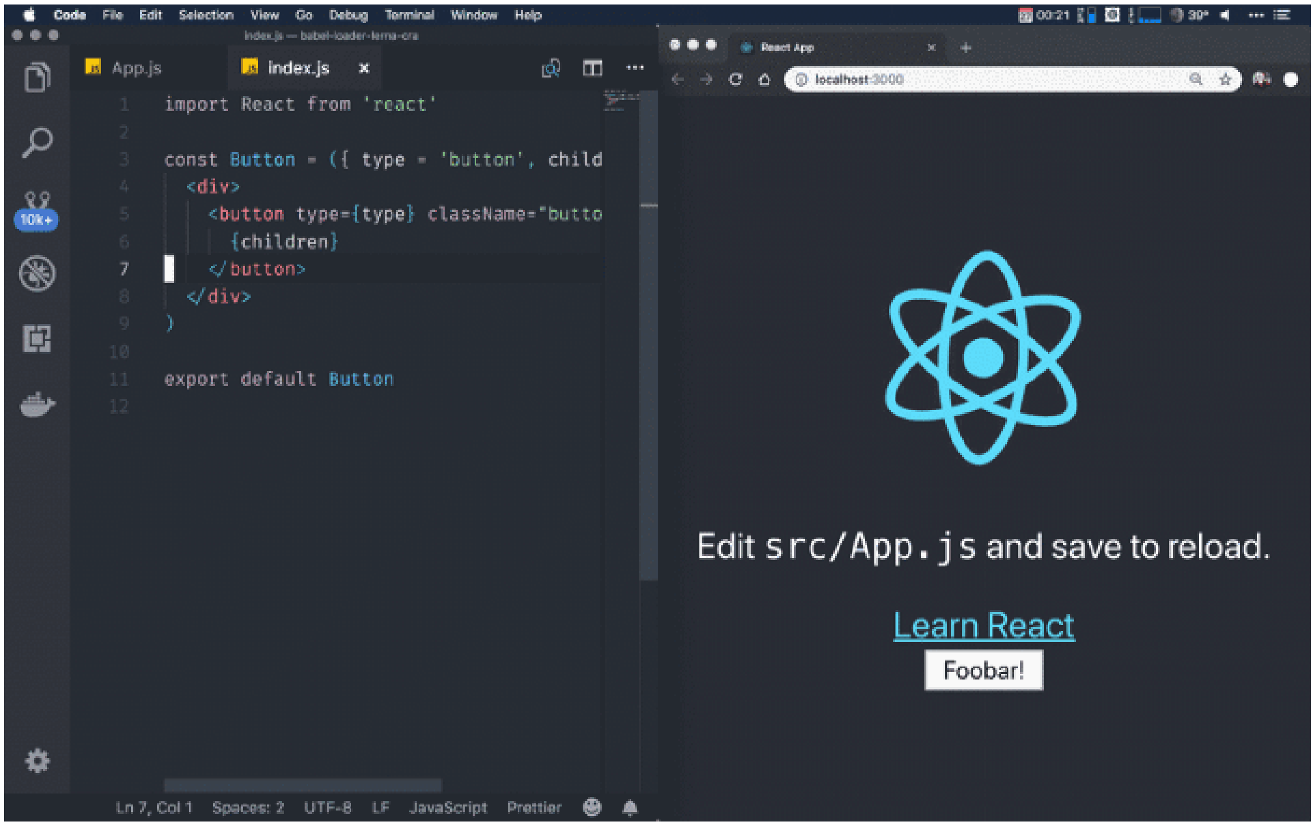
## So what did we get out of this workaround?

### Auto-Transpilation of Lerna Siblings

- Our React App can now import sibling Lerna dependencies and transpile then when needed.

### React App Hot Reloading

- When we change our React component file, will hot-update the app without having to add any global watchers to the Lerna project to kick of a transpile.
- Here is our `CompButton` component being Hot-Reloaded as it is being updated:

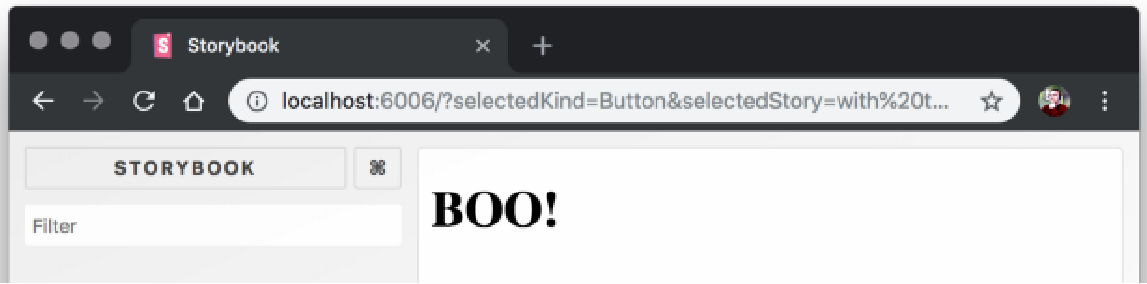


```
import React from 'react'

const Button = ({ type = 'button', child
  <div>
    <button type={type} className="butto
      {children}
    </button>
  </div>
)

export default Button
```

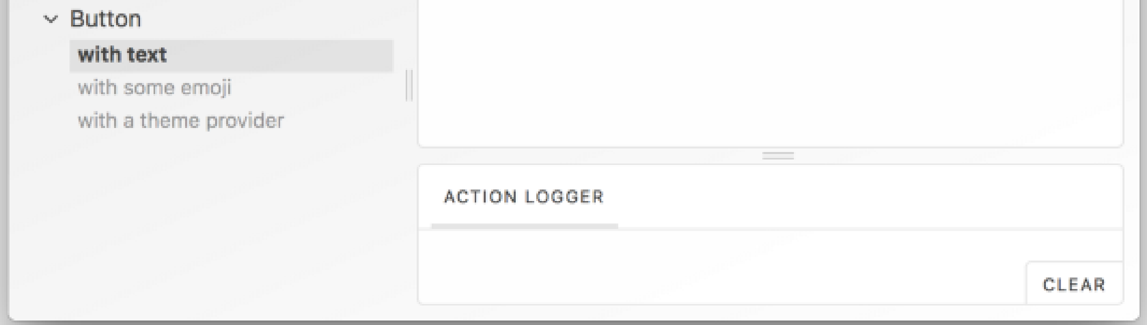
App.js
index.js
Ln 7, Col 1  Spaces: 2  UTF-8  LF  JavaScript  Prettier
React App
localhost:3000
Edit src/App.js and save to reload.
Learn React
Foobar!

### Storybook Hot Reloading

- Nothing special here, but it's worth noting that our Storybook will hot-reload too:



Storybook
localhost:6006/?selectedKind=Button&selectedStory=with%20t...
STORYBOOK
Filter
BOO!



## Conclusion

I think this is as far as I would like to take this in a single article. I hope someone else finds this setup useful. If people are interested I will follow up with Part 2 on how to setup CI to ship multiple React Apps from this Monorepo setup.

Comments, feedback, suggestions welcome.

— Alistair MacDonald

## Interesting Articles on This Topic:

- https://medium.com/@luisvieira_gmr/building-large-scale-react-applications-in-a-monorepo-91cd4637c131
- https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext
- https://github.com/facebook/create-react-app/issues/1333
- https://github.com/facebook/create-react-app/pull/3741
- https://github.com/jamiebuilds/react-loadable
- https://daveceddia.com/customize-create-react-app-webpack-without-ejecting/

Monorepo Lerna React Storybook Create React App