Published in Bits and Pieces

Kannan Nagasamy Follow
Feb 16 · 6 min read · Listen

Save

Create React App without Create React App

This article talks about the process of creating react app without using any libraries or frameworks such as “create-react-app” , “NextJS” etc.

Prerequisites concepts to know

- **Webpack —** helps in bundling our code into one single file
- **Babel** — used to convert ECMAScript 2015+ (ES6+) code into a backwards compatible version of JavaScript that can be run by older JavaScript engines.
- **Node.js** — installing node, creating package.json and installing node modules

- How webpack and b
- The actual starting to ending flow of building the React app;
- How development and production build is being setup and its significance;
- Setting up the server details thats required;
- Writing webpack and babel config files and understanding the logic that exists there;
- How we can configure client-side and server-side rendering;
- Understanding how HMR is works in React.

Source code

- *Repository — https://github.com/kannanagasamy/react-app-without-cra*
- *Branch — main*

Step by Step

1. Make sure node is installed in your system

Install Node.js in your system and make sure its installed by typing `node -v` in your terminal.

2. Create project folder and package.json

Create a project folder of any name and navigate to the folder and then use `npm init` to create a **package.json** file inside the folder. Navigate to the folder.

3. Install webpack dependencies

```
npm i --save-dev webpack webpack-cli webpack-dev-server
```

- **webpack** — Will allow us to bundle all of our code into a final build

- **webpack-dev-serve** .js Express server.It uses a library called SockJS to emulate a web socket. Will enable us to create a localhost dev environment

4. Install the Babel dependencies

```
npm i --save-dev babel-loader @babel/preset-env @babel/core
@babel/plugin-transform-runtime
@babel/preset-react
babel-eslint
@babel/runtime
@babel/cli
```

- **babel-loader —** allows transpiling JavaScript files using babel and webpack. exposes a loader-builder utility that allows users to add custom handling of Babel's configuration for each file that it processes.
- **@babel/preset-env —** allows you to use the latest JavaScript without needing to micromanage which syntax transforms (and optionally, browser polyfills) are needed by your target environment(s). This both makes your life easier and JavaScript bundles smaller!
- **@babel/core —** core package
- **@babel/plugin-transform-runtime —** A plugin that enables the re-use of Babel's injected helper code to save on codesize.
- **@babel/preset-react —** use react preset when we are using Reactjs. Helps in converting html files to react based file
- **babel-eslint —** parser that allows ESLint to run on source code that is transformed by Babel
- **@babel/runtime —** package that contains a polyfill and many other things that Babel can reference.
- **@babel/cli —** command line interface to use babel

```
npm i --save-dev eslint eslint-config-airbnb-base
eslint-plugin-jest
eslint-config-prettier
path
```

6. Install react and react-dom

```
npm i react react-dom
```

7. Create index.html file

Create folder called “**public**” in the root of the project. Inside that, create **index.html**.

8. Create App.js file

Create src folder and ate a file called **App.js**. Add the following code to it:

```
import React from "react";

const App = () =>{
    return (
        <h1>
            Welcome to React App thats build using Webpack and Babel separately
        </h1>
    )
}

export default App
```

App.js hosted with ❤ by GitHub view raw

8. Create index.js file

Create an **index.js** file at the root of project or anywhere you wish to have. This will act as entry point for our webpack.

Add the following code to it:

```
reactDom.render(<App /
```

index.js hosted with ❤ by GitHub view raw

9. Create webpack.config.js file

Create a file called **webpack.config.js** at the root of project and add the following code. On a higher note, this file contains configs that takes care of bundling the files into one single file and setting up the dev server

The comments in the code helps us understand what each line does:

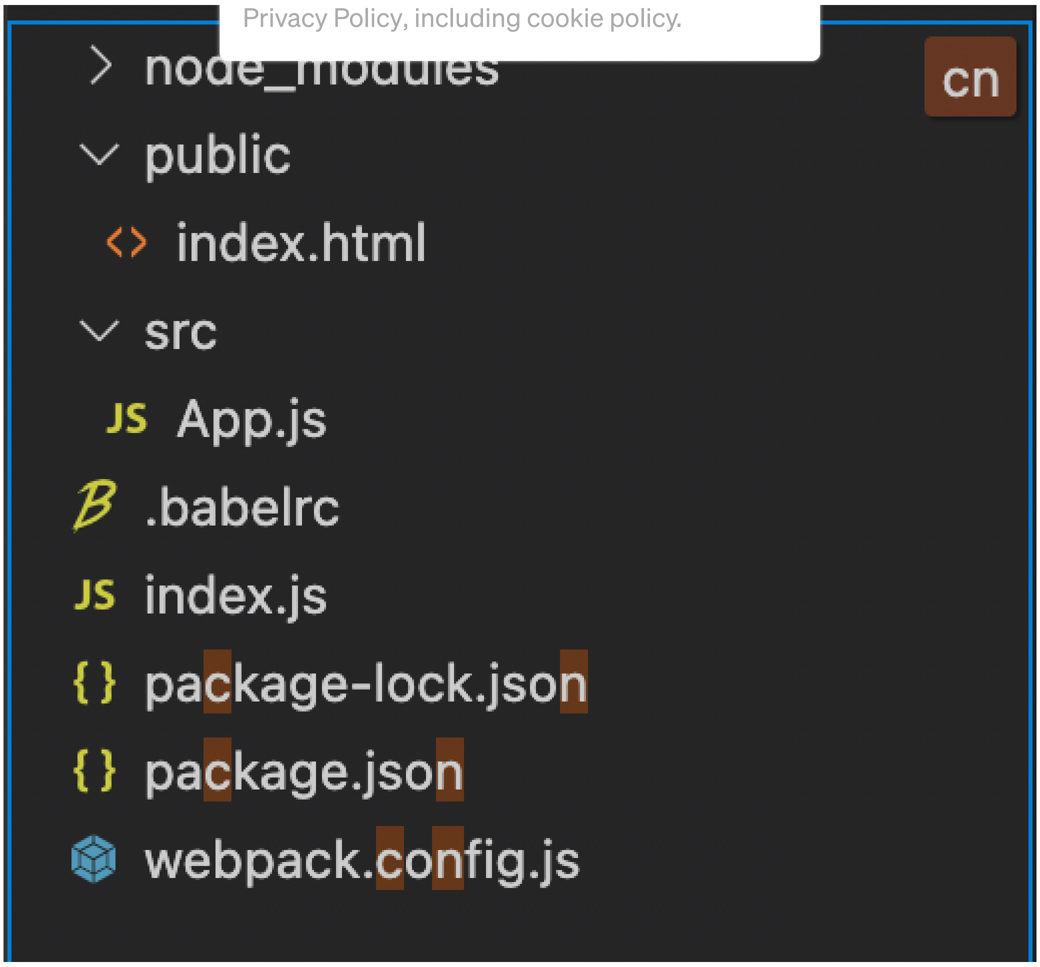
This is the configuration abel to use the plugin and presets defined inside.

11. Update package.json file

Add the start and build scripts to it — *line number 7 and 8.*

- The **start** script asks to run the webpack-dev-server in our current project at port 9500, from the public folder.

12. Final project folder structure has to be like this



```
> node_modules
v public
    index.html
v src
    App.js
  .babelrc
  index.js
  package-lock.json
  package.json
  webpack.config.js
```

13. Run "npm run build"

- Once added the above code, hit `npm run build` . it generates **main.js** file in our **public** folder. The file is actually **over 1 MB** in size. This is our development build.

14. Run "npm start"

Start the application by giving `npm start` from terminal. This will start our dev server.

The final code can be found in the repo link I shared above.

Other Key Findings

Changing the build to production

- Now we can try changing it to production build. For this you need to make the following change to **webpack.config.js** file.

- now running npm ru ut the size will be very less (<200kb).
- With the optimization from **1000 KB** to **200 KB**, we might want to use production build always. But, we should use development mode while doing development because the **hot reloading is faster in development mode.**

Hot Module Replacement

- HMR is handled by webpack-dev-server. We can use HMR without page load option too. Setting the required options helps greatly in performance aspect.
- Check the below code snippet for various scenarios:

```
//If you want to use HMR but no live reload then use the below
config in webpack.config.js
devServer: {
        hot: true ,
        liveReload:false
    }

//If you want don't want to use HMR but want to use live reload
then,
devServer: {
        hot: false ,
        liveReload: true
    },

//If you want don't want to use live reload then,
devServer: {
        hot: false , //this is mandatory to be set to false for this
        liveReload: false
    },
```

References

1. HMR with webpack — https://webpack.js.org/guides/hot-module-replacement/
2. Different ways to reduce the bundle size — https://blog.jakoblind.no/3-ways-to-reduce-webpack-bundle-size/
3. Understanding devserver and working in details —

5. How does productio
   https://webpack.js.org/guides/production/

6. Setting up perfect devpack server — https://linguinecode.com/post/how-to-setup-webpack-dev-server-react-babel

7. Loaders in details — https://webpack.js.org/concepts/loaders/

8. Understanding babel-preset-env in details — https://blog.jakoblind.no/babel-preset-env/

I hope you found this article useful. Will meet you soon with next one.

Build composable web applications

Don't build web monoliths. Use Bit to create and compose decoupled software components — in your favorite frameworks like React or Node. Build scalable and modular applications with a powerful and enjoyable dev experience.

Bring your team to Bit Cloud to host and collaborate on components together, and greatly speed up, scale, and standardize development as a team. Start with composable frontends like a Design System or Micro Frontends, or explore the composable backend. **Give it a try →**