Don't use create-react-app: How you can set up your own reactjs boilerplate.

#react #javascript #tutorial #webdev

Nikhil Kumaran S Aug 1, 2020 · 7 min read

What is CRA?

Create React App is a toolchain built and maintained by developers at Facebook for bootstrapping React applications. You simply run one command and Create React App sets up the tools you need to start your React project.

Advantages of CRA

- Get started with a single command

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

- Less to Learn. You can just focus on React alone and don't have to worry about webpack, babel, and other such build dependencies.
- Only one build dependency `react-scripts`. This maintains all your build dependencies, so it's easy to maintain and upgrade with just one command.

```
npm install react-scripts@latest
```

then it overrides the **Only one build dependency** advantage. The other way is you can use packages like customize-cra or react-app-rewired but then they have limited capabilities.

- Abstracts everything. It's important to understand the things that need to run a React app. But due to it's **Only one build dependency** advantage, a beginner might think that `react-scripts` is the only dependency needed to run react apps and might not know that transpiler(babel), bundler(webpack) are the key dependencies which are used under the hood by `react-scripts`. This happened to me until I read this awesome article.
- CRA is bloated - IMO. For example, CRA comes with SASS support, if you are using `plain CSS` or `Less` it's an extra dependency that you will never use. Here is a package.json of an ejected CRA app.

The alternative for CRA is to set up your own boilerplate. The only advantage that we can take from CRA is **Get started with a single command** and we can eliminate all of its disadvantages by setting up dependencies and configs by ourselves. We cannot take the other two advantages because it introduces two disadvantages(Abstracts everything and Difficult to add custom build configs).

This repo has all the code used in this blog post.

First, initialize your project with npm and git

```
npm init
git init
```

Let's quickly create a .gitignore file to ignore the following folders

```
node_modules
build
```

Now, let's look at what are the basic dependencies that are needed to run a React app.

react and react-dom

These are the only two runtime dependencies you need.

```
npm install react react-dom --save
```

Transpiler(Babel)

Transpiler converts ECMAScript 2015+ code into a backward-compatible version of JavaScript in current and older browsers. We also use this to transpile JSX by adding presets.

```
npm install @babel/core @babel/preset-env @babel/preset-react --save-dev
```

a property in package.json.

```
{
    "presets": [
        "@babel/preset-env",
        "@babel/preset-react"
    ]
}
```

You can add various presets and plugins based on your need.

Bundler(Webpack)

Bundler bundles your code and all its dependencies together in one bundle file(or more if you use code splitting).

```
npm install webpack webpack-cli webpack-dev-server babel-loader css-loader style-loader
```

A simple webpack.config.js for React application looks like this.

```
const path = require('path');
const HtmlWebPackPlugin = require('html-webpack-plugin');

module.exports = {
  output: {
    path: path.resolve(__dirname, 'build'),
    filename: 'bundle.js',
  },
  resolve: {
    modules: [path.join(__dirname, 'src'), 'node_modules'],
    alias: {
      react: path.join(__dirname, 'node_modules', 'react'),
    },
  },
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
        },
      },
      {
        test: /\.css$/,
        use: [
          {
            loader: 'style-loader',
```

```
          },
        ],
      },
    ],
  },
  plugins: [
    new HtmlWebPackPlugin({
      template: './src/index.html',
    }),
  ],
};
```

You can add various loaders based on your need. Check out my blog post on webpack optimizations where I talk about various webpack configs that you can add to make your React app production-ready.

That is all the dependencies we need. Now let's add an HTML template file and a react component.

Let's create src folder and add index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>React Boilerplate</title>
</head>
<body>
  <div id="root"></div>
</body>
</html>
```

Let's create a HelloWorld.js react component in the src folder

```
import React from 'react';

const HelloWorld = () => {
  return (
      <h3>Hello World</h3>
  );
};

export default HelloWorld;
```

Let's add index.js file to the src folder

```
import React from 'react';
```

```
import HelloWorld from './HelloWorld';

render(<HelloWorld />, document.getElementById('root'));
```

Finally, let's add the start and build scripts in package.json

```
"scripts": {
    "start": "webpack-dev-server --mode=development --open --hot",
    "build": "webpack --mode=production"
  }
```

That is it. Now our react app is ready to run. Try the commands `npm start` and `npm run build`.

Now, let's implement the **Get started with a single command** advantage from CRA. Basically, we are going to use an executable JS file that runs when we type a specific command(your boilerplate name) in the command line. Eg. `reactjs-boilerplate new-project` For this, we are going to use bin property in package.json.

Let's first create the executable JS file. Install fs-extra

```
npm i fs-extra
```

Create `bin/start.js` file on your project root with the following content.

```
#!/usr/bin/env node
const fs = require("fs-extra");
const path = require("path");
const https = require("https");
const { exec } = require("child_process");

const packageJson = require("../package.json");

const scripts = `"start": "webpack-dev-server --mode=development --open --hot",
"build": "webpack --mode=production"`;

const babel = `"babel": ${JSON.stringify(packageJson.babel)}`;

const getDeps = (deps) =>
  Object.entries(deps)
    .map((dep) => `${dep[0]}@${dep[1]}`)
    .toString()
    .replace(/,/g, " ")
    .replace(/^/g, "")
    // exclude the dependency only used in this file, nor relevant to the boilerplate
    .replace(/fs-extra[^\s]+/g, "");

console.log("Initializing project..");
```

```
 mkdir ${process.argv[2]} && cd ${process.argv[2]} && npm init -f",
(initErr, initStdout, initStderr) => {
  if (initErr) {
    console.error(`Everything was fine, then it wasn't:
  ${initErr}`);
    return;
  }
  const packageJSON = `${process.argv[2]}/package.json`;
  // replace the default scripts
  fs.readFile(packageJSON, (err, file) => {
    if (err) throw err;
    const data = file
      .toString()
      .replace(
        '"test": "echo \\"Error: no test specified\\" && exit 1"',
        scripts
      )
      .replace('"keywords": []', babel);
    fs.writeFile(packageJSON, data, (err2) => err2 || true);
  });

  const filesToCopy = ["webpack.config.js"];

  for (let i = 0; i < filesToCopy.length; i += 1) {
    fs.createReadStream(path.join(__dirname, `../${filesToCopy[i]}`)).pipe(
      fs.createWriteStream(`${process.argv[2]}/${filesToCopy[i]}`)
    );
  }

  // npm will remove the .gitignore file when the package is installed, therefore it c
  https.get(
    "https://raw.githubusercontent.com/Nikhil-Kumaran/reactjs-boilerplate/master/.giti
    (res) => {
      res.setEncoding("utf8");
      let body = "";
      res.on("data", (data) => {
        body += data;
      });
      res.on("end", () => {
        fs.writeFile(
          `${process.argv[2]}/.gitignore`,
          body,
          { encoding: "utf-8" },
          (err) => {
            if (err) throw err;
          }
        );
      });
    }
  );
```

```
    console.log("Installing deps -- it might take a few minutes..");
    const devDeps = getDeps(packageJson.devDependencies);
    const deps = getDeps(packageJson.dependencies);
    exec(
      `cd ${process.argv[2]} && git init && node -v && npm -v && npm i -D ${devDeps} &&
      (npmErr, npmStdout, npmStderr) => {
        if (npmErr) {
          console.error(`Some error while installing dependencies
      ${npmErr}`);
          return;
        }
        console.log(npmStdout);
        console.log("Dependencies installed");

        console.log("Copying additional files..");
        // copy additional source files
        fs.copy(path.join(__dirname, "../src"), `${process.argv[2]}/src`)
          .then(() =>
            console.log(
              `All done!\n\nYour project is now ready\n\nUse the below command to run th
            )
          )
          .catch((err) => console.error(err));
      }
    );
  }
);
```

Now let's map the executable JS file with a command. Paste this in your package.json

```
"bin": {
    "your-boilerplate-name": "./bin/start.js"
  }
```

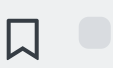
Now let's link the package(boilerplate) locally by running

```
npm link
```

Now, when this command is typed in the terminal(command prompt), `your-boilerplate-name my-app`, our `start.js` executable is invoked and it creates a new folder named `my-app`, copies `package.json`, `webpack.config.js`, `gitignore`, `src/` and installs the dependencies inside `my-app` project.

Great, now this works in your local. You can bootstrap React projects(with your own build configs) with just a single command.





push your code to GitHub and follow these instructions.

Hurray! We created our alternative to create-react-app within a few minutes, which is not bloated(you can add dependencies as per your requirement) and easier to add/modify build configs.

Of course, our set up is very minimal, and it's certainly not ready for production. You have to add a few more webpack configs to optimize your build.

I've created a reactjs-boilerplate with the production-ready build set up, with linters and pre-commit hooks. Give it a try. Suggestions and contributions are welcome.

Recap

- We saw the advantages and disadvantages of CRA.
- We decided to take **Get started with a single command** advantage from CRA and implement it in our project and eliminate all of its drawbacks.
- We added minimal webpack and babel configs required to run a react application
- We created a HelloWorld.js react component, ran it using dev server, and build it.
- We created an executable JS file and mapped it with a command name via bin property in the package.json.
- We used `npm link` to link our boilerplate and made our boilerplate to bootstrap new react projects with a single command.

That's it, folks, Thanks for reading this blog post. Hope it's been useful for you. Please do comment your questions and suggestions.

References

- https://medium.com/netscape/a-guide-to-create-a-nodejs-command-line-package-c2166ad0452e
- https://github.com/Nikhil-Kumaran/reactjs-boilerplate
- https://reactjs.org/docs/create-a-new-react-app.html#creating-a-toolchain-from-scratch
- https://medium.com/the-node-js-collection/modern-javascript-explained-for-dinosaurs-f695e9747b70

build stack is just a single disadvantage, but it's a *huge* one.

31

Nikhil Kumaran S • Aug 2 '20

Maintaining is not that difficult. You just have to be informed with new webpack and babel releases just like you do for React and other feature releases.

3

Alain Van Hout • Aug 2 '20

There is a reason why a great many developers abhor the word 'just'.

25

Antti Pihlaja • Aug 5 '20

CRA is not boilerplate. It's build tool itself. You can DIY same kind of setup but you are really underestimating how much work it is to keep it up to date. CRA is managing build toolchain versioning for all "hidden" dependencies. When project lasts more than a couple of months, it's huge benefit.

9

James Hubert • Nov 19 '20

Thank you for putting this together. A really worthwhile exercise that- even if you don't use it- helps students understand what is going on inside all of those packages and extra files in CRA.

5

Carlos Tighe • Aug 1 '20

"The only advantage that we can take from CRA is Get started with a single command"

Really?

If that was the only advantage you wouldn't need the code that was in the rest of this article.

29

- Only one build dependency `react-scripts` - Easy to upgrade
- Because of point 1, fewer things to learn - Concentrate on React alone.
- Get started with a single command.

Because of the first two points, we are sacrificing two things

- Difficult to add custom build configs based on your requirement - You can eject but it takes out the first 2 advs.
- Abstracts everything. - As a dev, I'd like to have control over what's happening in my app and configure app as per my need.

So we cannot incorporate the first two advs, so "The only advantage that we can take from CRA is Get started with a single command".

In the rest of the article, I wrote a step by step guide to add your own reactjs setup with webpack and babel. And finally implemented the "Single command" to bootstrap react app advantage using JS executable file and bin property.

If there is any other advantage of CRA, please share. Let's discuss how to implement them.

7

Carlos Tighe • Aug 2 '20

" I wrote a step by step guide to add your own reactjs setup with webpack and babel."

The other advantage is you don't have to do your own setup with webpack and Babel

6 *Thread*

Nikhil Kumaran S • Aug 2 '20

Like I said before, some devs like myself would like to have control over the configs.

2

rockiger • Aug 2 '20

I really have to disagree with the premise. Maintaining the whole build process is a lot of work and very error prone.

I usually go a step futher and use the react-boilerplate template for create-react-app:

```
npx create-react-app --template cra-template-rb my-app
```

18

Nikhil Kumaran S • Aug 2 '20

Sure, it ultimately boils down to your requirement. I'd like to have control over the configs and I'm confident about maintaining the build process so I gave it a try☺

2

Federico Vázquez • Aug 2 '20 • Edited

I'd say it's ok and probably a must-do at some point of every React and FE developer journey; to do some experimenting with the tooling we use every day, which we don't even care about it (webpack, babel, eslint, etc).
But that's the point to use said technologies, they were created for us to configure once and forget about them, so we can focus on our business logic intead.
IMO it isn't wise to encourage people to do this for every project. Yes, do it once and for learning purposes, but this is clearly a foot gun.
The experience will teach you to not do this. Once the projects grows, your dependencies get obsolete and because your build system is totally custom, you're many steps behind, not only to upgrade something like your React version (which might be straightforward), but also to gain advantages of many cool improvements the people behind CRA came up with, and you're losing that train.
"Let's measure how many lines of code we didn't have to write instead"

9

Nikhil Kumaran S • Aug 3 '20

I see your point. But hear me out, my project grows and I create my own design component library and I want this library to be bundled separately from other vendor bundles. If I have my own build configs I can easily add `cacheGroups` in webpack to achieve this but it's difficult to do so in CRA(without ejecting). It ultimately boils down to your project requirements☺

5

Federico Vázquez • Aug 4 '20

To be honest, that's when you should start considering how you structure your project. You are probably adding some unnecessary complexity into its build system.
I'd use a separate project for my UI library, heck, I wouldn't even use webpack on it, I prefer rollup for libraries, but that's totally my opinion.

Matthew Clark • Aug 2 '20

Well said - I think its very important to know the tools we use under the hood on a daily basis. However projects like CRA were made and maintained by the some of the greatest minds in web development - might be best to stick with it unless it's for educational purposes.

3

Alexandru-Dan Pop • Aug 2 '20

I like this article and agree with some of the points. Most important one is that you need to know what is running under the hood.

Considering that, I would like to point out the main disadvantage - you need to update manually all the build dependencies, or maintain outdated configs.

Create React App is not to be used for every app, it depends on the use case. **Gatsby** and **Next.Js** might also be great candidates for this 0 build tooling approach.

I wrote about it here:

> When to pick Gatsby, Next.Js or Create React App
>
> Alexandru-Dan Pop · Jul 13 '20 · 4 min read
>
> #react #javascript #gatsby #healthydebate

11

Nikhil Kumaran S • Aug 2 '20

Great article.

2

Alexandru-Dan Pop • Aug 2 '20

Thanks!

2

Jason Steinhauser • Aug 2 '20

I couldn't agree more! CRA is waaaaaay too bloated in my opinion. It took me a while to learn Webpack when I wanted to do some customization, and I think relying on CRA was part of it.

Annis Monadjem • Sep 11 '20

Nikhil thank you very much for your excellent article!

Just a small typo in 'webpack.config.js', inside:

```
plugins: [
new HtmlWebPackPlugin({
template: './src/index.html',
}),
],
```

instead of `template: './src/index.html'` should be: `template: './index.html'`

Now, i'm able to `yarn start` the project!

2

David Oliva Tirado • Aug 2 '20

Some months ago I made a project to do, in some way, the same you do here. With some more experience in the field and more knowledge I can assure you that I learnt a lot, but it was a PAIN.

If you want to do it to understand what is behind webpack, react and babel its a good exercise. Elsewhere, you are just wasting time. IMO.

3

Andrew Baisden • Aug 2 '20 • Edited

Its cool however setting up your own React boilerplate seems kind of tedious and adds an additional layer of complexity. For one you now have your own custom build which is fine but now your setup differs from that of all other developers who are using create-react-app which is not great if its a team project. There could be unknown errors as its not been battle tested. Plus creating this setup every-time you want to start a project is just going to add more time and of course you are not going to do this during an interview.

Also you added lots of custom code which a beginner might not understand and it would not be good practice to just copy and paste the code without actually knowing what it all does and how to write it. Using Next.js would be far better its more lightweight.

2

good practice to just copy and paste the code without actually knowing what it all does and how to write it."
Agreed. But at the same time its key to understand how webpack and babel works

"Plus creating this setup every-time you want to start a project is just going to add more time and of course you are not going to do this during an interview"

You don't have to do this every time. That's why we use executable JS file and bin property to bootstrap react app with single command just like CRA.

4

Robert Myers • Aug 2 '20

Since react is really a build step, is there any reason not to have react and react-dom in devDependencies instead of dependencies?

2

Chris Naismith • Aug 2 '20

Dan Abramov had mentioned in a CRA, since everything is going through a build step. You should think of it more as everything is a dependency instead of a dev dependency.

Bundle analyzing tools or websites (example bundlephobia) do not read your dev dependencies. I'd recommend keeping it as a dependency!

5

Nikhil Kumaran S • Aug 2 '20

React is not a build step, you need React to run your application.
classic.yarnpkg.com/en/docs/depend...

1

Robert Myers • Aug 2 '20

The current project I'm working on has it all in devDependencies and it works fine.

The project is a little overloaded, it's basically a monorepo with server and client in the same project. All of the client stuff is in devDependencies, everything needed for the server runtime is in regular dependencies.

Federico Vázquez • Aug 2 '20

React per-se doesn't run without this build step. That's why he says that you can list everything under devDeps

2

Ulf Byskov • Sep 14 '20

Maybe it's just bad luck, but every React project I have be working on, since CRA was made, had something which required me to either eject or use my own setup. And if that is the choice, I will go with my own setup as an ejected CRA is far from simple to deal with.

2

Gautam krishna R • Aug 8 '20 • Edited

Maintaining your own fork of CRA is the best option if you want to add custom build config. You can still pull the upstream changes with this approach.

create-react-app.dev/docs/alternat...

1

Paul John Butad • Mar 8 • Edited

In one of my Django projects (web framework using python)...

I've used REACT to handle all the frontend stuffs. Coz' Django works using Model-View-Template (MVT), so Django can use my REACT APP in its own template engine.

But, you can't rely on CRA when youre using REACT with Django. That's why I was forced to learn all of these "webpack and babel stuffs".

And gladly, it wasn't actually hard to learn. That's why I completely agree with all of the advantages that the author stated.

1

a-tonchev • Oct 15 '20

Even I need sometimes to edit webpack, to start and keep webpack really up to date with all stuff can be really pain in the as*. 'Don't use CRA...' is a little bit missleading title, better to be something like 'How to create your own CRA' :)

Maksym Minenko • Aug 7 '20

I like it. It's a third of the create-react-app size and starts much much faster.

Do we really need the resolve (modules: ... alias:) block though?

2

Nikhil Kumaran S • Aug 7 '20

I'm glad you liked it ❤

> Do we really need the resolve (modules: ... alias:) block though?

It's not required for the example mentioned in this blog post. resolve is used to tell webpack how to resolve our imports(modules).
Eg: If you are using `import 'pages/about/About'` (instead of relative paths for cleaner syntax) you need to specify `modules: [path.join(__dirname, 'src'), 'node_modules']` so webpack will first look for `pages` folder inside `src`. If you don't specify `modules` then webpack by default will check in the node_modules and throw this error `Module not found: Error: Can't resolve 'pages/about/About'`.

1

Mainendra • Jan 17

Use `"start": "webpack serve --mode=development --hot"` for webpack cli 4 😄

1

Mainendra • Jan 17

Can you add typescript too?

1

hlvu • Aug 2 '20

"Let's do this", "Let's do that". How about "why"?

1

Nikhil Kumaran S • Aug 2 '20

a single command like CRA. With this, you have complete control over your app and also you can boot up a new react project with just a single command.

5

View full discussion (38 comments)

Code of Conduct • Report abuse

Nikhil Kumaran S

Web dev at Cloudera | Reactjs | Tech Speaker/Writer | Mentor | Ex - Qube Cinema

WORK
Engineer - Development at Qube Cinema

LOCATION
Chennai, India

EDUCATION
SVCE

JOINED
Jan 15, 2020

More from Nikhil Kumaran S

You probably don't need Redux: Use React Context + useReducer hook

#react #redux #javascript #webdev

Building your own React Hooks.

#react #javascript #webdev

Web Workers: For non-blocking User Interface

#javascript #webperf