Digging into CRA

Written By - Divyansh Singh
Contact - LinkedIn

What is CRA ?

CRA stands for "Create React App," which is a command line tool used to create a new React application. It is maintained by Facebook and is designed to make it easy for developers to set up a new React project, with a basic file structure and some default configuration already in place.

The command to create a new React app using CRA is "npx create-react-app [app-name]", where [app-name] is the name of the new application. This command will create a new directory with the specified name, and inside that directory, it will create the initial file structure and install all the necessary dependencies for a basic React application.

CRA also provides some additional features like development server, eslint and prettier integration, automatic code splitting, and support for environment variable.

Once the setup is done, you can navigate to the created directory and start the development server using the command `npm start` or `yarn start`.

What is the file structure of CRA ?

The file structure of a React application created using Create React App (CRA) is similar to the following:

```
my-app/
  node_modules/
  public/
    index.html
    favicon.ico
    manifest.json
  src/
    index.js
    App.js
    ...other components and files
  package.json
  package-lock.json
```

```
README.md
```

- **node_modules/:** This directory contains all the npm packages that your application depends on.
- **public/:** This directory contains the public files of your application, such as index.html, favicon.ico, and manifest.json.
- **src/:** This is the directory where you will spend most of your time. It contains the source code of your application, including index.js, App.js, and other components and files.
- **package.json:** This file contains the metadata of your application, such as the name, version, and dependencies.
- **package-lock.json:** This file contains a detailed list of all the dependencies and their versions, which helps to ensure that the exact same versions of packages are installed on every machine.
- **README.md:** This file contains documentation for your application.

This file structure is created by default, but developers can configure or modify it as per their needs.

Create a react template without using CRA command explaining things in detail ?

Creating a React template without using the Create React App (CRA) command requires a few more steps, but it gives you more control over the initial setup and configuration of your application. Here's an overview of the process:

1. **Create a new directory:** Create a new directory for your application, and navigate to it in the command line.
2. **Initialize npm:** Run the command npm init -y to create a new package.json file for your application. This file will contain the metadata for your application, such as the name, version, and dependencies.
3. **Install React and other dependencies:** Run the command npm install react react-dom to install React and React DOM as dependencies for your application. You may also want to install other dependencies such as webpack, babel, and other development dependencies.

4. **Create the file structure:** Create the necessary directories and files for your application, such as src/ and public/. In the src/ directory, you can create your index.js file, which is the entry point for your application, and other components and files. In the public/ directory, you can create your index.html file, which is the main HTML file for your application.
5. **Configure webpack:** Create a new webpack.config.js file in the root of your application, and configure it to handle JSX and ES6 syntax, and transpile the code using babel. This will allow you to use JSX and ES6 syntax in your application, and also bundle all your javascript files.
6. **Add scripts:** In your package.json file, add scripts for starting your development server, and building the production version of your application.
7. **Start the development server:** Run the command npm start or yarn start to start the development server, and start building your application.

What is a webpack and How to configure it ?

Webpack is a JavaScript module bundler. It is a tool that takes multiple JavaScript files and combines them into a single file (or a few files) that can be run in a web browser. It also allows you to use other web technologies such as TypeScript, CSS, and images in your application and bundle them together with your JavaScript code. Additionally, Webpack allows for easy integration with other tools such as transpilers and minifiers, making it a powerful tool for building and deploying modern web applications. It is widely used in the development of single-page applications and other complex web projects.
Here's an example of how you might configure webpack for a React application:

1. **Install webpack and webpack-cli:** In the command line, navigate to the root of your application and run the command npm install webpack webpack-cli --save-dev to install webpack and the webpack command line interface as development dependencies.
2. **Create a new webpack.config.js file:** In the root of your application, create a new file called webpack.config.js. This file will contain the configuration for webpack.
3. **Configure entry and output:** In the webpack.config.js file, configure the entry point for your application and the output location for the bundled file.

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

```
module.exports = {
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'bundle.js'
    },
};
```

4. **Add loaders:** In the `webpack.config.js` file, add loaders to handle different file types. For example, you might use the `babel-loader` to transpile your code from JSX and ES6 syntax to plain JavaScript, and the `url-loader` to handle images and other files.

```
module.exports = {
    ...
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /node_modules/,
                use: {
                    loader: "babel-loader"
                }
            },
            {
                test: /\.(png|jpe?g|gif)$/i,
                use: [
                    {
                        loader: 'url-loader',
                        options: {
                            limit: 8192
                        },
                    },
                ],
            },
        ]
    },
};
```

5. **Add Plugins:** In the `webpack.config.js` file, you can add plugins to perform some action during the bundling process. For example, you might use the `html-webpack-plugin` to generate an HTML file that includes the bundled JavaScript file.

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

```
module.exports = {
    ...
    plugins: [
        new HtmlWebpackPlugin({
            template: './public/index.html',
            filename: './index.html',
        }),
    ],
};
```

This is just a basic example of how you might configure webpack for a React application, but you can customize and add more things as per your requirements.

The key concepts in webpack are entry and output points, loaders, and plugins.

- **entry:** The entry point tells webpack where to start looking for code to bundle.
- **output:** The output tells webpack where to save the bundled file(s) and what to name them.
- **loaders:** Loaders are used to transform different types of files before they are added to the bundle. They allow you to use modern JavaScript features and pre-process CSS, images and more.
- **plugins:** Plugins are used to perform a wider range of tasks such as bundle optimization, asset management and injection of environment variables.

HtmlWebpackPlugin is a plugin for webpack that simplifies the creation of HTML files to serve your webpack bundles. It automatically generates an HTML file for you and includes all the webpack generated JS and CSS files. It allows you to specify a template file and inject the webpack bundles into the template at the appropriate place, instead of having to manually add the script and link tags. This can make it easier to manage the HTML for your application, especially if you are using webpack to handle your asset pipeline. It also allows you to specify various options such as the title of the HTML file, the file name, and so on. It is commonly used in conjunction with webpack to generate a single HTML file that includes all the JavaScript and CSS files required by your application, making it easier to serve your application.

By configuring webpack, you can use modern JavaScript features, pre-process different types of files, and bundle all of your application's dependencies into a single file, or a few files, to be loaded by the browser. This can greatly improve the performance of your application and make it easier to maintain and deploy.

What is bundle.js in above code ?

In the example code I provided earlier, `bundle.js` is the name of the output file that webpack creates after bundling all of the application's dependencies.

In the `output` configuration of webpack, we have set the `path` to `path.resolve(__dirname, 'dist')` which will create a directory called dist in the root of your application where the bundled file will be saved, and `filename` is set to `bundle.js` which is the name of the bundled file.

When webpack runs, it starts at the entry point specified in the configuration, and it recursively builds a dependency graph of all the modules that your application requires. Then it bundles all those modules together into a single file, `bundle.js`, which can be included in the HTML file, and loaded by the browser.

This bundle file contains all the code of your application and its dependencies and it can be loaded by the browser in a single request, which can greatly improve the performance of your application.

What scripts are we talking about above when we say Add scripts ?

When I mentioned "Add scripts" above, I was referring to adding scripts in the `package.json` file to automate common tasks such as starting the development server and building the production version of the application.

Here's an example of how you might add scripts to the `package.json` file:

```
{
  "name": "my-app",
  "version": "1.0.0",
  "scripts": {
    "start": "webpack-dev-server --mode development --open",
    "build": "webpack --mode production"
  },
  "dependencies": {
    "react": "^16.13.1",
    "react-dom": "^16.13.1"
  },
  "devDependencies":
    "@babel/core": "^7.20.12",
    "@babel/preset-env": "^7.20.2",
```

```
    "@babel/preset-react": "^7.18.6",
    "babel-loader": "^9.1.2",
    "html-webpack-plugin": "^5.5.0",
    "webpack": "^5.75.0",
    "webpack-cli": "^5.0.1",
    "webpack-dev-server": "^4.11.1"
  }
}
```

@babel/core is the core package of the Babel JavaScript transpiler. It is responsible for transforming JavaScript code written in modern syntax into code that can be understood by older browsers and JavaScript environments. It is a powerful tool that allows developers to use the latest features of the JavaScript language, even if those features are not yet supported by all browsers. It has a set of plugins and presets that can be used to transpile the code written in JSX, typescript, flow, etc to javascript.
@babel/preset-env is a preset for Babel that allows you to use the latest JavaScript features even if they are not yet supported by all browsers. It is a preset that automatically determines the JavaScript features that need to be transpiled based on the targeted environment and applies the necessary plugins. This preset is designed to be used in conjunction with the @babel/core package, it is a collection of plugins that are responsible for transforming modern JavaScript syntax into code that can be understood by older browsers. It can also take in a list of targets (like browsers or Node version) and it will automatically determine which features need to be transpiled and which don't for that targeted environment. It is a recommended preset for most of the projects as it handles the compatibility issues in a very good way, also it is a very maintainable way of handling the compatibility issues.
@babel/preset-react is a preset for Babel that is specifically designed for use with React projects. It includes a set of plugins that are responsible for transforming JSX syntax into regular JavaScript, which is necessary for React projects because JSX is not natively understood by web browsers. This preset allows developers to write their React components using JSX, which is a syntax extension for JavaScript that allows developers to embed HTML-like elements in their code. When using this preset, the JSX code is transpiled into regular JavaScript code, which can be understood by web browsers and other JavaScript environments.
babel-loader is a webpack loader that allows you to use Babel in conjunction with webpack. A webpack loader is a module that allows you to pre-process files as they are being loaded by webpack. By using babel-loader, you can configure

webpack to transpile your JavaScript files using Babel, which makes it easy to use the latest JavaScript features even if they are not yet supported by all browsers. It works by intercepting the JavaScript files that are being imported into your application and running them through Babel before they are passed on to webpack for further processing. This allows you to write code using the latest JavaScript features and still have it work in older browsers, because the code will be transpiled to an older version of JavaScript before it is sent to the browser.
webpack-cli is a command line interface (CLI) for webpack. It allows you to run webpack commands from the command line, which can be useful for automating common tasks such as building and deploying your application. With webpack-cli, you can run webpack commands like "webpack" or "webpack-dev-server" directly from the command line, instead of having to use npm scripts or other build tools.
webpack-dev-server is a development server that allows you to run your application in a local development environment. It is built on top of webpack and allows you to use webpack's features, such as hot module replacement (HMR), while developing your application. When you run webpack-dev-server, it starts a local development server that serves your application from memory. This means that any changes you make to your application's source code will be immediately reflected in the browser without the need for a full build. The server also supports HMR which allows you to update the parts of your application that have changed without having to refresh the entire page. It also has a built-in web server that serves the application. This can be useful for development because it allows you to run your application on a local web server and access it in a web browser, which can make it easier to test and debug your application. It is typically used during development process, which allows you to see the changes made to the code immediately in the browser and also allows you to test the application in a local environment with the same configuration as your production environment.
In the above code snippet, I have added two scripts start and build in the scripts section of package.json.

- **start script:** This script starts a development server using webpack-dev-server and opens the application in the browser. This script is useful for development purposes, as it allows you to see the changes you make in real-time.
- **build script:** This script creates a production version of the application using webpack by running the command webpack --mode production. The --mode production flag tells webpack to optimize the bundle for production.

You can run these scripts by using the command npm run <script-name> or yarn <script-name> in the command line. For example, to start the development server, you would run npm run start or yarn start.

In this way, you can automate your development and production workflows, and easily manage the different dependencies and configurations for your application.

What is index.js in src folder ?

In the context of the React application created without using Create React App (CRA), `index.js` in the `src` folder is the entry point of the application. It is the starting point of the application, from where the React application starts and the root component of your application is rendered to the DOM. It's where the React application is bootstrapped and the React DOM is initialized by importing the necessary modules and dependencies, such as React, React DOM, and the root component of your application. It uses React DOM's render method to render the root component of the application to a specific DOM element, in this case, the `<div>` with the `id` of "root" in the `index.html` file.

It also contains the logic for hot-reloading, that allows developers to see the changes they make in real-time during development without having to manually refresh the browser.
Hot-reloading is a development feature that allows you to see changes made to your code in real-time, without having to manually refresh the browser. When you make changes to your code, the development server (usually webpack-dev-server) automatically detects the changes and updates the browser, so you can see the updated version of your application without having to refresh the page.

In summary, the `index.js` file in the `src` folder is the starting point of the React application, and it's responsible for bootstrapping the React application and rendering the root component to the DOM.

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App />, document.getElementById("root"));

if (module.hot) {
  module.hot.accept();
```

```
}
```

The code snippet `if (` module.hot `) {` module.hot.accept `();}` enables the hot-reloading feature in development. When webpack's development server is running and hot-module-replacement is enabled, it will detect the changes and update the browser in real-time, allowing developers to see the effects of the changes they made to the code without having to manually refresh the browser.

What is index.html in public folder ?

In a React application, the `public` folder typically contains the `index.html` file, which serves as the entry point for the app. The `index.html` file is the template that the app's JavaScript code is injected into when it is built and served. It typically includes the basic HTML structure of the app, such as the `<head>` and `<body>` tags, as well as links to the app's CSS and JavaScript files. When the React app is running, it will manipulate the contents of the `index.html` file to display the desired content to the user.

```
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
    <script src="bundle.js"></script>
  </body>
</html>
```

What is .babelrc ?

In a React application, the `.babelrc` file can be used to export presets, which are pre-configured sets of plugins and rules for Babel. Exporting presets in the `.babelrc` file allows developers to easily enable or disable specific features or syntax, rather than manually configuring them individually.

For example, the `@babel/preset-env` preset can be used to automatically configure Babel to support the latest version of JavaScript, including features like arrow functions, template literals, and destructuring. Similarly, the `@babel/preset-react` preset can be used to automatically enable support for

JSX, which is the syntax used in React for describing components. By using presets in the `.babelrc` file, developers can quickly and easily configure Babel to support the latest JavaScript features and React specific syntax, without having to manually configure the individual plugins and rules required to achieve the same result.

For example, the .babelrc file could be like this

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

This tells babel to use both preset-env and preset-react while transpiling the code.

Summary of above

When creating a React application, it's common to use a tool like Create React App (CRA) to set up the initial file structure and dependencies. However, it's also possible to create a React application without using CRA, which allows for more customization and control over the initial setup and configuration.

To create a React application without using CRA, you would need to perform the following steps:

1. Create a new directory: Create a new directory for your application.
2. Initialize npm: Initialize npm in your application by creating a new package.json file, which will contain the metadata for your application, such as the name, version, and dependencies.
3. Install React and other dependencies: Install React and React DOM as dependencies for your application. You may also want to install other dependencies such as webpack, babel, and other development dependencies.
4. Create the file structure: Create the necessary directories and files for your application, such as src/ and public/. In the src/ directory, you can create your index.js file, which is the entry point for your application, and other components and files. In the public/ directory, you can create your index.html file, which is the main HTML file for your application.

5. Configure webpack: Create a new webpack.config.js file in the root of your application, and configure it to handle JSX and ES6 syntax, and transpile the code using babel. This will allow you to use JSX and ES6 syntax in your application, and also bundle all your javascript files.
6. Add scripts: In your package.json file, add scripts for starting your development server, and building the production version of your application.
7. Start the development server: Start the development server by running the command npm start or yarn start, and start building your application.

When you configure webpack, you tell it what to do with the different types of files in your application, such as JSX and CSS files. You also specify the entry point and output location for the bundled file, which is often called bundle.js. This file contains all the code of your application and its dependencies, and it can be loaded by the browser in a single request, which can greatly improve the performance of your application.

By adding scripts in the package.json file, you can automate common tasks such as starting the development server, building the production version of your application, and more. This allows you to easily manage the different dependencies and configurations for your application and automate your development and production workflows.