ReactJS - Creating a React Application

As we learned earlier, React library can be used in both simple and complex application. Simple application normally includes the React library in its script section. In complex application, developers have to split the code into multiple files and organize the code into a standard structure. Here, React toolchain provides pre-defined structure to bootstrap the application. Also, developers are free to use their own project structure to organize the code.

Let us see how to create simple as well as complex React application −

- Simple application using CDN
- Complex application using *React Create App* cli
- Complex application using customized method

Using Rollup bundler

Rollup is one of the small and fast JavaScript bundlers. Let us learn how to use rollup bundler in this chapter.

Open a terminal and go to your workspace.

```
cd /go/to/your/workspace
```

Next, create a folder, *expense-manager-rollup* and move to newly created folder. Also, open the folder in your favorite editor or IDE.

```
mkdir expense-manager-rollup
cd expense-manager-rollup
```

Next, create and initialize the project.

```
npm init -y
```

Next, install React libraries *(react and react-dom)*.

```
npm install react@^17.0.0 react-dom@^17.0.0 --save
```

Next, install babel and its preset libraries as development dependency.

```
npm install @babel/preset-env @babel/preset-react
@babel/core @babel/plugin-proposal-class-properties -D
```

Next, install rollup and its plugin libraries as development dependency.

```
npm i -D rollup postcss@8.1 @rollup/plugin-babel
@rollup/plugin-commonjs @rollup/plugin-node-resolve
@rollup/plugin-replace rollup-plugin-livereload
rollup-plugin-postcss rollup-plugin-serve postcss@8.1
postcss-modules@4 rollup-plugin-postcss
```

Next, install corejs and regenerator runtime for async programming.

```
npm i regenerator-runtime core-js
```

Next, create a babel configuration file, .babelrc under the root folder to configure the babel compiler.

```
{
   "presets": [
      [
         "@babel/preset-env",
         {
            "useBuiltIns": "usage",
            "corejs": 3,
            "targets": "> 0.25%, not dead"
         }
      ],
      "@babel/preset-react"
   ],
   "plugins": [
      "@babel/plugin-proposal-class-properties"
   ]
}
```

Next, create a *rollup.config.js* file in the root folder to configure the rollup bundler.

```
import babel from '@rollup/plugin-babel';
import resolve from '@rollup/plugin-node-resolve';
import commonjs from '@rollup/plugin-commonjs';
import replace from '@rollup/plugin-replace';
import serve from 'rollup-plugin-serve';
import livereload from 'rollup-plugin-livereload';
import postcss from 'rollup-plugin-postcss'

export default {
   input: 'src/index.js',
   output: {
      file: 'public/index.js',
      format: 'iife',
   },
   plugins: [
      commonjs({
         include: [
            'node_modules/**',
         ],
         exclude: [
            'node_modules/process-es6/**',
         ],
      }),
      resolve(),
      babel({
         exclude: 'node_modules/**'
      }),
      replace({
         'process.env.NODE_ENV': JSON.stringify('production'),
      }),
      postcss({
         autoModules: true
      }),
      livereload('public'),
      serve({
         contentBase: 'public',
         port: 3000,
         open: true,
      }), // index.html should be in root of project
   ]
}
```

Next, update the *package.json* and include our entry point *(public/index.js and public/styles.css)* and command to build and run the application.

```
...
"main": "public/index.js",
"style": "public/styles.css",
"files": [
   "public"
],
```

```
"scripts": {
   "start": "rollup -c -w",
   "build": "rollup"
},
...
```

Next, create a src folder in the root directory of the application, which will hold all the source code of the application.

Next, create a folder, components under src to include our React components. The idea is to create two files, <component>.js to write the component logic and <component.css> to include the component specific styles.

The final structure of the application will be as follows −

```
|-- package-lock.json
|-- package.json
|-- rollup.config.js
|-- .babelrc
`-- public
   |-- index.html
`-- src
   |-- index.js
   `-- components
   |  |-- mycom.js
   |  |-- mycom.css
```

Let us create a new component, *HelloWorld* to confirm our setup is working fine. Create a file, *HelloWorld.js* under components folder and write a simple component to emit *Hello World* message.

```
import React from "react";

class HelloWorld extends React.Component {
   render() {
      return (
         <div>
            <h1>Hello World!</h1>
         </div>
      );
   }
}
export default HelloWorld;
```

Next, create our main file, *index.js* under *src* folder and call our newly created component.

```
import React from 'react';
import ReactDOM from 'react-dom';
import HelloWorld from './components/HelloWorld';

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

Next, create a *public* folder in the root directory.

Next, create a html file, index.html (under public folder*), which will be our entry point of the application.

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8">
      <title>Expense Manager :: Rollup version</title>
   </head>
   <body>
      <div id="root"></div>
      <script type="text/JavaScript" src="./index.js"></script>
   </body>
</html>
```

Next, build and run the application.

```
npm start
```

The *npm* build command will execute the *rollup* and bundle our application into a single file, *dist/index*.js file and start serving the application. The *dev* command will recompile the code whenever the source code is changed and also reload the changes in the browser.

```
> expense-manager-rollup@1.0.0 build /path/to/your/workspace/expense-manager-rollup
> rollup -c
rollup v2.36.1
bundles src/index.js → dist\index.js...
LiveReload enabled
http://localhost:10001 -> /path/to/your/workspace/expense-manager-rollup/dist
created dist\index.js in 4.7s

waiting for changes...
```

Next, open the browser and enter *http://localhost:3000* in the address bar and press enter. serve application will serve our webpage as shown below.

Hello World
Image not found or type unknown

Using Parcel bundler

Parcel is fast bundler with zero configuration. It expects just the entry point of the application and it will resolve the dependency itself and bundle the application. Let us learn how to use parcel bundler in this chapter.

First, install the parcel bundler.

```
npm install -g parcel-bundler
```

Open a terminal and go to your workspace.

```
cd /go/to/your/workspace
```

Next, create a folder, *expense-manager-parcel* and move to newly created folder. Also, open the folder in your favorite editor or IDE.

```
mkdir expense-manager-parcel
cd expense-manager-parcel
```

Next, create and initialize the project.

```
npm init -y
```

Next, install React libraries *(react and react-dom)*.

```
npm install react@^17.0.0 react-dom@^17.0.0 --save
```

Next, install babel and its preset libraries as development dependency.

```
npm install @babel/preset-env @babel/preset-react @babel/core @babel/plugin-proposal-class-prope
```

Next, create a babel configuration file, .babelrc under the root folder to configure the babel compiler.

```
{
   "presets": [
      "@babel/preset-env",
      "@babel/preset-react"
   ],
   "plugins": [
      "@babel/plugin-proposal-class-properties"
   ]
}
```

Next, update the package.json and include our entry point (src/index.js) and commands to build and run the application.

```
...
"main": "src/index.js",
"scripts": {
   "start": "parcel public/index.html",
   "build": "parcel build public/index.html --out-dir dist"
},
...
```

Next, create a *src* folder in the root directory of the application, which will hold all the source code of the application.

Next, create a folder, *components* under src to include our React components. The idea is to create two files, *<component>.js* to write the component logic and *<component.css>* to include the component specific styles.

The final structure of the application will be as follows −

```
|-- package-lock.json
|-- package.json
|-- .babelrc
`-- public
   |-- index.html
`-- src
   |-- index.js
   `-- components
   |  |-- mycom.js
   |  |-- mycom.css
```

Let us create a new component, *HelloWorld* to confirm our setup is working fine. Create a file, *HelloWorld.js* under components folder and write a simple component to *emit Hello World*

message.

```
import React from "react";

class HelloWorld extends React.Component {
   render() {
      return (
         <div>
            <h1>Hello World!</h1>
         </div>
      );
   }
}
export default HelloWorld;
```

Next, create our main file, *index.js* under *src* folder and call our newly created component.

```
import React from 'react';
import ReactDOM from 'react-dom';
import HelloWorld from './components/HelloWorld';

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

Next, create a *public* folder in the root directory.

Next, create a html file, *index.html* (in the *public* folder), which will be our entry point of the application.

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8">
      <title>Expense Manager :: Parcel version</title>
   </head>
   <body>
      <div id="root"></div>
      <script type="text/JavaScript" src="../src/index.js"></script>
   </body>
</html>
```

Next, build and run the application.

```
npm start
```

The *npm* build command will execute the parcel command. It will bundle and serve the application on the fly. It recompiles whenever the source code is changed and also reload the changes in the browser.

```
> expense-manager-parcel@1.0.0 dev /go/to/your/workspace/expense-manager-parcel
> parcel index.html Server running at http://localhost:1234
√ Built in 10.41s.
```

Next, open the browser and enter **http://localhost:1234** in the address bar and press enter.

Hello World
Image not found or type unknown

To create the production bundle of the application to deploy it in production server, use *build* command. It will generate a *index.js* file with all the bundled source code under dist folder.

```
npm run build
> expense-manager-parcel@1.0.0 build /go/to/your/workspace/expense-manager-parcel
> parcel build index.html --out-dir dist

√  Built in 6.42s.

dist\src.80621d09.js.map    270.23 KB     79ms
dist\src.80621d09.js        131.49 KB    4.67s
dist\index.html                 221 B    1.63s
```

Revision #1
Created 14 December 2022 10:37:57 by Admin
Updated 14 December 2022 10:38:41 by Admin