**Webpack** and **Babel** are two essential tools in modern JavaScript development, especially for building complex applications like those using React.

Let's dive into what each of these tools does and why they are important.

**Webpack**

**Webpack** is a popular module bundler for JavaScript applications. Its primary purpose is to bundle JavaScript files for usage in a browser, but it can also transform, bundle, or package just about any resource or asset.

**Key Features of Webpack:**

* **Module Bundling**: Webpack takes all the code files in your project and turns them into a single bundle or multiple bundles. This reduces the number of HTTP requests needed to load your application.
* **Code Splitting**: It allows you to split your code into smaller chunks which can be loaded on demand. This helps to optimize the performance of your application.
* **Loaders**: Loaders in Webpack enable you to preprocess files as they are imported or loaded. This includes things like transpiling ES6+ JavaScript to ES5, compiling Sass to CSS, or even loading images and fonts.
* **Plugins**: Plugins provide a way to extend Webpack's capabilities. There are plugins for tasks like minifying your code, injecting environment variables, and more.

**Babel**

**Babel** is a JavaScript compiler. It is mainly used to convert ES6+ code into backwards compatible versions of JavaScript that can run in older browsers or environments.

**Key Features of Babel:**

* **Transpiling**: Converts modern JavaScript (ES6+) syntax to ES5, ensuring compatibility with older browsers.
* **Plugins and Presets**: Babel is highly configurable and uses plugins and presets to determine what transformations are applied to your code. For example, the @babel/preset-env preset includes plugins that support ES6+ features.
* **JSX Transformation**: Babel can transform JSX syntax (used in React) into regular JavaScript. This makes it an essential tool for React developers.

**How They Work Together**

In a modern JavaScript development workflow, Webpack and Babel often work together to provide a seamless development experience. Here’s how:

1. **Webpack** is used to bundle your application, managing the dependencies and assets.
2. **Babel** is used to transpile your modern JavaScript code (including JSX) into code that is compatible with all browsers.
3. Webpack can be configured to use Babel as a loader, allowing it to transpile your JavaScript files as it processes them.

**Example Configuration**

Here's a basic example of how to set up Webpack with Babel in a React project:

**webpack.config.js**:

js

const path = require('path');

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

const { CleanWebpackPlugin } = require('clean-webpack-plugin');

module.exports = {

entry: './src/index.js',

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist'),

},

module: {

rules: [

{

test: /\.(js|jsx)$/,

exclude: /node\_modules/,

use: {

loader: 'babel-loader',

},

},

{

test: /\.css$/,

use: ['style-loader', 'css-loader'],

},

],

},

resolve: {

extensions: ['.js', '.jsx'],

},

plugins: [

new CleanWebpackPlugin(),

new HtmlWebpackPlugin({

template: './public/index.html',

filename: 'index.html',

}),

],

devServer: {

contentBase: path.join(\_\_dirname, 'dist'),

compress: true,

port: 9000,

},

};

**.babelrc**:

json

{

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

}

In this configuration:

* Webpack bundles your application starting from src/index.js.
* Babel, used via babel-loader, transpiles your modern JavaScript and JSX code.
* The output is bundled into dist/bundle.js.
* HtmlWebpackPlugin and CleanWebpackPlugin are used to generate and clean the dist directory respectively.

Webpack and Babel together provide a powerful and flexible build process for modern JavaScript applications. Let me know if you need further details or help setting up your project!

Setting up Webpack with React allows you to customize your build process and optimize your application for production. Here’s a step-by-step guide to get started with Webpack and React:

**Step 1: Create a New Project**

Create a new directory for your project and navigate into it:

sh

mkdir webpack-react-app

cd webpack-react-app

**Step 2: Initialize a New Node.js Project**

Initialize a new Node.js project by running:

sh

npm init -y

This will create a package.json file in your project directory.

**Step 3: Install Dependencies**

Install the required dependencies:

sh

npm install react react-dom

npm install --save-dev webpack webpack-cli webpack-dev-server babel-loader @babel/core @babel/preset-env @babel/preset-react html-webpack-plugin clean-webpack-plugin css-loader style-loader

**Step 4: Create the Project Structure**

Create the following directory structure:

webpack-react-app/

|-- public/

| |-- index.html

|-- src/

| |-- App.js

| |-- index.js

|-- .babelrc

|-- webpack.config.js

|-- package.json

**Step 5: Configure Babel**

Create a .babelrc file in the root directory with the following content:

json

{

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

}

**Step 6: Configure Webpack**

Create a webpack.config.js file in the root directory with the following content:

js

const path = require('path');

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

const { CleanWebpackPlugin } = require('clean-webpack-plugin');

module.exports = {

entry: './src/index.js',

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist'),

},

module: {

rules: [

{

test: /\.(js|jsx)$/,

exclude: /node\_modules/,

use: {

loader: 'babel-loader',

},

},

{

test: /\.css$/,

use: ['style-loader', 'css-loader'],

},

],

},

resolve: {

extensions: ['.js', '.jsx'],

},

plugins: [

new CleanWebpackPlugin(),

new HtmlWebpackPlugin({

template: './public/index.html',

filename: 'index.html',

}),

],

devServer: {

contentBase: path.join(\_\_dirname, 'dist'),

compress: true,

port: 9000,

},

};

**Step 7: Create the HTML Template**

Create an index.html file in the public directory with the following content:

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Webpack React App</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

**Step 8: Create the React Components**

Create App.js and index.js in the src directory with the following content:

**App.js**:

jsx

import React from 'react';

const App = () => {

return <h1>Hello, Webpack with React!</h1>;

};

export default App;

**index.js**:

jsx

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

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

**Step 9: Add Scripts to package.json**

Add the following scripts to your package.json file:

json

"scripts": {

"start": "webpack serve --mode development",

"build": "webpack --mode production"

}

**Step 10: Run the Application**

Start the development server by running:

sh

npm start

This will start the Webpack Dev Server and open your application in the browser at http://localhost:9000.

You can build the production version of your application by running:

sh

npm

React is a JavaScript Library for building user interfaces. It simplifies [web development](https://www.turing.com/development-services/web) by breaking the User Interface (UI) into small blocks known as components that can be reused across pages in the website. If you are familiar with React you might’ve heard the term [create-react-app](https://www.turing.com/kb/create-a-react-app-with-typescript), create-react-app (CRA) is a command line interface that spins up a react project quickly by setting up everything under the hood and saves time for the developer to start developing any desired website. Still, under the hood, use of webpack with React is done to configure the output react project.

Table of Contents

* [1. What is webpack?](https://www.turing.com/kb/use-webpack-with-react#what-is-webpack?)
* [2. Getting started](https://www.turing.com/kb/use-webpack-with-react#getting-started)
* [3. Project setup](https://www.turing.com/kb/use-webpack-with-react#project-setup)
* [3.1. What are loaders?](https://www.turing.com/kb/use-webpack-with-react#what-are-loaders?)
* [4. Configuring webpack](https://www.turing.com/kb/use-webpack-with-react#configuring-webpack)
* [4.1. Starting the React app](https://www.turing.com/kb/use-webpack-with-react#starting-the-react-app)
* [5. Conclusion](https://www.turing.com/kb/use-webpack-with-react#conclusion)

What is webpack?

According to the webpack official site, webpack is a static module bundler for modern JavaScript applications. Its primary purpose is to bundle JavaScript to use in the browser.  
Webpack also has many features apart from bundling JavaScript code like hot reload replacement feature that removes the pain of having to refresh the browser for every new feature added to the codebase. There are other options for bundling JavaScript apart from webpack like Parcel, Vite, and Turbopack. In this article, we are going to see how to use and configure webpack from scratch in a React project.

Getting started

Create an empty directory and run the following command to initiate an npm project,

npm init -y

Install react and react-dom in the project with the following command

npm install react react-dom

Now we need to install Webpack and its dependencies for the project with the following command,

Npm i -D webpack webpack-dev-server webpack-cli

Now let’s see the use of each package we installed earlier:

* react: This is the library for creating user interfaces where we’ll import most APIs from it like useState, useContext, and useEffect.
* react-dom: This package provides Document Object Model (DOM) specific methods that can be used at the top level of your app and as an escape hatch to get outside the React model if you need to.

Next, open package.json and you should see something like this:

json

{

"name": "webpack-blog",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {},

"keywords": [],

"author": "",

"license": "MIT",

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"webpack": "^5.75.0",

"webpack-cli": "^5.0.1",

"webpack-dev-server": "^4.11.1"

}

}

Now let’s see the use of the development dependencies as shown in the package.json file above.

* webpack: This is the core library, a Javascript bundler that will be used in the project.
* webpack-cli: According to their page, this library provides a flexible set of useful commands for developers to increase speed while6 setting up a custom webpack project.
* webpack-dev-server: This will provide a live reloading feature so we don’t have to refresh the web page each time we make a change to the codebase, and it should be used in development only.

Project setup

The code base right now only consists of a package.json, package-lock.json, and a node\_modules directory. It is not resembling the React project we are used to see when we initialize with Create React App. Now let’s add some files in the src folder that will be populated later to make it much more of a React project hence the project structure will be like:

|\_\_\_\_src

| I\_\_\_\_App.js

| I\_\_\_\_index.css

| I\_\_\_\_index.html

| I\_\_\_\_index.js

|\_\_\_\_\_package-lock.json

|\_\_\_\_\_package.json

|\_\_\_\_\_webpack.config.js

Let’s start populating the files with some code, starting with index.html,

html

<!DOCTYPE html>

<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>How to use Webpack with React</title>

</head>

<body>

<div id="app"></div>

</body>

</html>

App.js

js

export default function App() {

return (

<div>

<h1>React App</h1>

</div>

);

}

index.js

js

import ReactDOM from "react-dom/client";

import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("app"));

root.render(<App />);

At this time even if you try to run the project it won’t work as we haven’t specified webpack configurations and utilized loaders.

What are loaders?

Loaders are an essential part of Webpack in a React project as they are responsible for parsing the JSX files and compiling complex JSX files to browser-understandable Javascript files. For this codebase, we will use the babel-loader to perform the task of loaders, install the library with the command below:

bash

npm install -D @babel/core @babel/preset-env @babel/preset-react babel-loader

The command above installs babel, babel-loader, and two presets.

Configuring webpack

Next, we have to instruct webpack to use babel for compiling JSX and other configurations for babel itself, the configuration is done in the webpack.config.js file as follows:

js

const path = require("path");

module.exports = {

entry: "./src/index.js",

output: { path: path.resolve(\_\_dirname, "dist") },

module: {

rules: [

{

test: /.(js|jsx)$/,

exclude: /node\_modules/,

use: {

loader: "babel-loader",

options:{

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

}

},

},

],

},

};

The above configuration makes use of the babel-loader wherever it finds files with js or jsx extension.

Starting the React app

Now we have to set up a build script in the package.json file for bundling the application:

{

"name": "webpack-blog",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"build": "webpack"

},

"keywords": [],

"author": "",

"license": "ISC",

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"@babel/core": "^7.20.12",

"@babel/preset-env": "^7.20.2",

"@babel/preset-react": "^7.18.6",

"babel-loader": "^9.1.2",

"webpack": "^5.75.0",

"webpack-cli": "^5.0.1",

"webpack-dev-server": "^4.11.1"

}

}

We’ve added the build script and tested it by running npm run build and you will see a dist folder is created that contains a bundled main.js file that can be attached to the index.html file.

<!DOCTYPE html>

<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>How to use Webpack with React</title>

</head>

<body>

<div id="app"></div>

<script src="../dist/main.js"></script>

</body>

</html>

Now open the browser and you will see the output is as it is defined in App.js

At this time the project is successfully using React and webpack.

Conclusion

Customizing webpack from scratch is really helpful for large apps and brings a good developer experience as it gives more control to the project. Give it a try and explore some more features and powers of webpack.

Now that you know how webpack can be integrated into React projects, it will be helpful to explore some other module bundlers before you choose one that fits your project needs.