⚠ The info in this post might be out of date, check out our docs instead. View docs

9 🚀 Using Webpack To Bootstrap A JS App (Express)💠

hyperupcall

🚀 Using Webpack To Bootstrap A JS App (Express)💠

Webpack is an integral part of modern JavaScript tooling. It bundles all your JavaScript modules and has plugins for bundling CSS, JSX, and many other files. It's used for creating React and Vue single page applications and much more!

We're going to be manually configuring Webpack to work with an Express server. So let's get started! 👍

Prerequisites

- Intermediate experience with JavaScript
- Understand basic Webpack concepts

Getting Started

Typically we create our projects (in development) using webpack-dev-server. But, because we don't have access to `npm` script commands (ex. `npm run start`) in repl.it, ~~we're going to be using Webpack from an Express app.~~

Important Edit: You can easily use `webpack-dev-server` by instantiating an instance of it, rather than using a CLI command. Find the guide here. **I highly recommend using `webpack-dev-server` directly rather than trying to use some middleware with Express.**

Let's start off with a super basic express app.

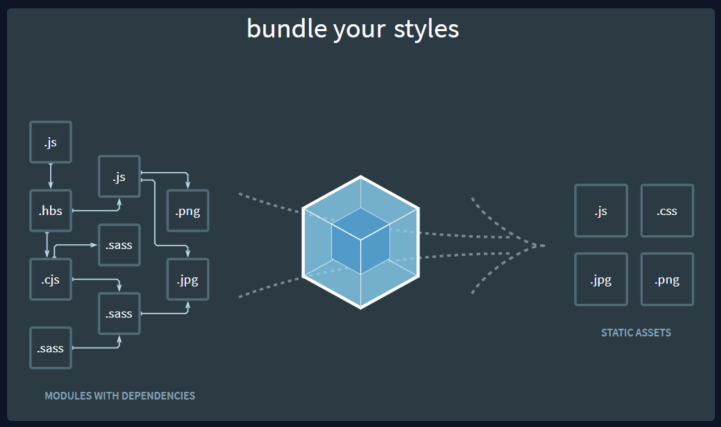
```
// index.js
let express = require("express");
let app = express();

app.get("/", (res, req) => req.send(`<p>Working</p>`))

let port = process.env.PORT || 3000;
app.listen(port, () => console.log(`App listening on ${port}`));
```

Remember to add `express` as a dependency.

Adding Webpack



We're using `webpack-dev-middleware` so we can use Webpack in this Express app. So, add `webpack-dev-middleware` and `webpack` to your dependencies and require them in your `index.js`

```
let webpack = require("webpack");
let webpackDevMiddleware = require("webpack-dev-middleware");
```

Now, let's create a configuration file for Webpack. We're going to be including all the standard stuff. Be

sure to add `friendly-errors-webpack-plugin` and `html-webpack-plugin` to your `package.json`.

```
// webpack.config.js
const path = require("path");
const webpack = require("webpack");
const FriendlyErrorsWebpackPlugin = require("friendly-errors-webpack-
plugin");
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  mode: "development",
  devtool: "cheap-module-eval-source-map",
  entry: {
    main: path.resolve(process.cwd(), "src", "main.js")
  },
  output: {
    path: path.resolve(process.cwd(), "dist"),
    publicPath: "/"
  },
  watchOptions: {
    // ignored: /node_modules/,
    aggregateTimeout: 300, // After seeing an edit, wait .3 seconds to
recompile
    poll: 500 // Check for edits every 5 seconds
  },
  plugins: [
    new FriendlyErrorsWebpackPlugin(),
    new webpack.ProgressPlugin(),
    new HtmlWebpackPlugin({
      template: path.resolve(process.cwd(), "public", "index.html")
    })
  ]
}
```

- `friendly-errors-webpack-plugin` outputs more helpful Webpack errors
- `html-webpack-plugin` creates an HTML5 file and injects `<script></script>` tags with our Webpack bundles. Recall in our config we created a bundle called `main`, which is resolved to `./src/main.js` (this is the Webpack bundle we're injecting). And, because we're adding a `template` property, it means that `html-webpack-plugin` will use our HTML file, then inject the script tags
- the `new webpack.ProgressPlugin(),` inside `plugins: []` gives Webpack's progress building your files (in the terminal)
- I chose `cheap-module-eval-source-map` for the property `devtool` because it provides fast compilation with source maps. Read more in the documentation
- `watchOptions` prevents Webpack from recompiling everything too much (and lagging your replit). See the inline comments and documentation for more info

Now, let's create our `src/main.js` and `public/index.html` that we referenced in the Webpack config!

For the `index.html`, I've included all the standard HTML boilerplate with the addition of a `div` with an `id` of `app`.

```
<!-- public/index.html -->
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Simple Webpack Boilerplate</title>
</head>

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

</html>
```

For the `main.js` I do some simple DOM manipulation and logging. This is where your application really starts.

```
// src/main.js
console.log("Delta Echo Foxtrot");

document.getElementById("app").innerHTML = "Golf Hotel India";
```

Now, this won't do anything if we don't use the `webpack-dev-middleware` in our Express app. So, inside `index.js`, add it.

First, let's add our Webpack configuration file. According to the documentation for `webpack-dev-middleware`, we add the following to create a simple configuration.

```
let webpackConfig = require("./webpack.config");
```

```
let webpackCompiler = webpack(webpackConfig);
let webpackDevMiddlewareOptions = {
  publicPath: webpackConfig.output.publicPath
};

app.use(webpackDevMiddleware(webpackCompiler, webpackDevMiddlewareOptions));
```

So, our final `index.js` should look like the following

```
// index.js
let express = require("express");
let app = express();

let webpack = require("webpack");
let webpackDevMiddleware = require("webpack-dev-middleware");

let webpackConfig = require("./webpack.config");
let webpackCompiler = webpack(webpackConfig);
let webpackDevMiddlewareOptions = {
  publicPath: webpackConfig.output.publicPath
};

app.use(webpackDevMiddleware(webpackCompiler, webpackDevMiddlewareOptions));

let port = process.env.PORT || 3000;
app.listen(port, () => console.log(`App listening on ${port}`));
```

And huzzah! We get "Golf Hotel India" on the web page.

Unsupported image

This is where the tutorial ends but where your Webpack application starts! Now you can use whatever plugins and loaders you need to create your applications!

Note that if you're creating an SPA (Single Page Application), you will have to do some extra configuration for HMR (Hot Module Replacement (using webpack-hot-middleware)) and HistoryAPI (using connect-history-api-fallback is simplest).

5 years ago

Programming languages

Python	JavaScript	TypeScript	Node.js
Nix (beta)	HTML, CSS, JS	C++	Golang