Congratulations!

We have made our tedious way through all of the webpack concepts. Now we get to actually make things awesome.

As you probably noted during this lesson so far, there are some things about the process we have now that aren't exactly smooth. Overall, I wouldn't call our set up a good dev experience and we don't have production set up at all. It is functional, and that's about it. But the whole point of using all these tools is to make our lives easier. So...what's going wrong? Let's use what we've learned to make some improvements.

Switch the Git Branch

If you wish to practice the steps mentioned in the video demonstration below, use the `5-webpack-mode` branch.
Else, you can switch to the branch `6-webpack-for-convenience` corresponding to the current exercise where all the steps have already been carried out.

```
git checkout 6-webpack-for-convenience
```

Install Webpack Dev Server

It helps in live reloading of the page, only for Development mode, and automatically re-builds the application.

Up next, have you noticed how we often have to remove the dist folder manually before re-running the build script? From what I have seen, when you rebuild, new code will be added to the bundled files, but if there was old code that you got rid of, webpack build does not remove the old stuff. So, we have been removing the dist folder via the terminal and before rebuilding.

Really though, that is an extra and unnecessary step. If we wanted to go really low tech, we could just edit our build script:

```
rm -rf dist && webpack-dev-server  --config webpack.dev.js --open
```

And there's really nothing wrong with that. Honestly, being comfortable customizing your npm scripts will make so many things easier. But, it is doing a little bit of extra work. That script blindly deletes everything and then rebuilds it, even if 99% of the code remained the same.

To be a little more efficient, there is a webpack plugin called Clean. From its documentation:

> By default, this plugin will remove all files inside webpack's output.path directory, as well as all unused webpack assets after every successful rebuild.

Now, some people will choose to go with the simpler blanket dist folder delete, but just to try it, lets install the clean plugin.

```
npm i -D clean-webpack-plugin
```

Then, as we learned before, to make webpack use a plugin, we have to do two things:

Add a require statement to the top of the webpack config file:

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

Add the plugin to Plugins array in the module.exports. The clean plugin is a good example of a plugin that allows for various options. Take a look at this:

We could use CleanWebpackPlugin like this:

```
      new CleanWebpackPlugin()
```

And the above would function. When there are no options passed in, a plugin will run all the default settings, but we can also pass in our custom selections of the various plugin options, like this:

```
            new CleanWebpackPlugin({
                        // Simulate the removal of files
                        dry: true,
                        // Write Logs to Console
                        verbose: true,
                        // Automatically remove all unused webpack assets on rebuild
                        cleanStaleWebpackAssets: true,
                    protectWebpackAssets: false
          })
```

You can’t know what options a plugin allows without reading the documentation for that plugin.

Add that code to your wepack.dev.js file and rerun the build script, now you’ll see a few lines added to the webpack output that tell you the clean plugin is functioning.

Quiz

What do you think are the most time-saving developer tools for you? Which tools do you absolutely want in a development environment, or which could you do without?

Your reflection

webpack development server

Things to think about

Thanks for your response.

Interview Question

You start work on a project with webpack already installed, but there is no specific build for production. Walk me through the steps of setting up a production environment in webpack

---

Your reflection

1-Create webpack.prod.js in project root folder then fill it with the following content const path = require('path') const webpack = require('webpack') const HtmlWebPackPlugin = require("html-webpack-plugin") module.exports = { entry: './src/client/index.js', mode: 'production', module: { rules: [{ test: '/\.js$/', exclude: /node_modules/, loader: "babel-loader" }] }, plugins: [new HtmlWebPackPlugin({ template: "./src/client/views/index.html", filename: "./index.html", })] } 2-Add the following line within scripts in packge.json file "build-prod": "webpack --config webpack.prod.js", 3-run the following command to build the project in production mode npm run build-prod

Things to think about

This is a very specific webpack question, and not one you would probably come across in a jr level job anywhere, but it is a good example of a question that dives more into the specifics of a technology.