Before creating custom webpack configuration, check to see if there's a Gatsby plugin already built that handles your use case in the plugins section. If there's not yet one and your use case is a general one, consider contributing your plugin to the Gatsby Plugin Library so it's available to others (including your future self).

To add custom webpack configurations, create (if there's not one already) a `gatsby-node.js` file in your root directory. Inside this file, export a function called `onCreateWebpackConfig`.

When Gatsby creates its webpack config, this function will be called allowing you to modify the default webpack config using webpack-merge.

Gatsby does multiple webpack builds with somewhat different configuration. Gatsby calls each build type a "stage". The following stages exist:

1. develop: when running the `gatsby develop` command. Has configuration for hot reloading and CSS injection into page
2. develop-html: same as develop but without react-hmre in the babel config for rendering the HTML component.
3. build-javascript: production JavaScript and CSS build. Creates route JavaScript bundles as well as common chunks for JavaScript and CSS.
4. build-html: production build static HTML pages

Check webpack.config.js for the source.

There are many plugins in the Gatsby repo using this API to look to for examples e.g. Sass, TypeScript, Glamor, and many more!

Examples

Here is an example adding an additional global variable via the `DefinePlugin` and the `less-loader`:

```
exports.onCreateWebpackConfig = ({
  stage,
  rules,
  loaders,
  plugins,
  actions,
}) => {
  actions.setWebpackConfig({
    module: {
      rules: [
        {
          test: /\.less$/,
          use: [
            // You don't need to add the matching ExtractText plugin
            // because gatsby already includes it and makes sure it's only
            // run at the appropriate stages, e.g. not in development
            loaders.miniCssExtract(),
            loaders.css({ importLoaders: 1 }),
            // the postcss loader comes with some nice defaults
            // including autoprefixer for our configured browsers
            loaders.postcss(),
            `less-loader`,
          ],
        },
      ],
    },
    plugins: [
      plugins.define({
        __DEVELOPMENT__: stage === `develop` || stage === `develop-html`,
      }),
    ],
  })
}
```

Absolute imports

Instead of writing `import Header from '../../components/header'` over and over again you can write `import Header from 'components/header'` with absolute imports:

```
exports.onCreateWebpackConfig = ({ stage, actions }) => {
  actions.setWebpackConfig({
    resolve: {
      modules: [path.resolve(__dirname, "src"), "node_modules"],
    },
  })
}
```

You can always find more information on *resolve* and other options in the official webpack docs.

Importing non-webpack tools using `yarn`

Note that using absolute imports only applies to webpack resolutions and will not work for other tools, e.g. ESLint or TypeScript. But if you are using yarn, then the best practice is to set up your imports in `package.json` as shown below:

```
{
  "dependencies": {
    "hooks": "link:./src/hooks"
  }
}
```

Modifying the Babel loader

You need this if you want to do things like transpile parts of `node_modules`.

```
exports.onCreateWebpackConfig = ({ actions, loaders, getConfig }) => {
  const config = getConfig()

  config.module.rules = [
    // Omit the default rule where test === '\.jsx?$'
    ...config.module.rules.filter(
      rule => String(rule.test) !== String(/\.jsx?$/)
    ),
    // Recreate it with custom exclude filter
    {
      // Called without any arguments, `loaders.js()` will return an
      // object like:
      // {
      //   options: undefined,
      //   loader: '/path/to/node_modules/gatsby/dist/utils/babel-loader.js',
      // }
      // Unless you're replacing Babel with a different transpiler, you probably
      // want this so that Gatsby will apply its required Babel
      // presets/plugins.  This will also merge in your configuration from
      // `babel.config.js`.
      ...loaders.js(),

      test: /\.jsx?$/,

      // Exclude all node_modules from transpilation, except for 'swiper' and 'dom7'
      exclude: modulePath =>
        /node_modules/.test(modulePath) &&
        !/node_modules\/(swiper|dom7)/.test(modulePath),
    },
  ]

  // This will completely replace the webpack config with the modified object.
  actions.replaceWebpackConfig(config)
}
```