Published in Geek Culture

Eze Henry Follow
Dec 7, 2021 · 5 min read · Listen

Implementing Webpack's Module federation in a Vue 2 application.

Photo by Tim Johnson on Unsplash

Micro Frontends:

The micro frontend is a frontend architecture where modules of the same product/application are built independently of each other.

design system, I copy/pasted components that were mutual to both applications which is not maintainable, and copy/pasting the components goes against the DRY principle. I decided to find a more scalable solution which made me select Webpack's module federation.

For this demo, we would be setting up Webpack's module federation from start to finish. So, fasten your seatbelts 😃.

Requirements

1. Two already set up vue2 projects: We won't be covering this on this blog post. If you are having issues setting up your vue project, visit the vue documentation.

Installations

To properly configure Webpack, we would need to add some dev dependencies to our project. P.S: These steps should be done for both projects. To add dev dependencies, we would run the command:

```
yarn add webpack webpack-cli webpack-dev-server vue-loader url-
loader sass-loader mini-css-extract-plugin html-webpack-plugin dart-
sass css-loader -D
```

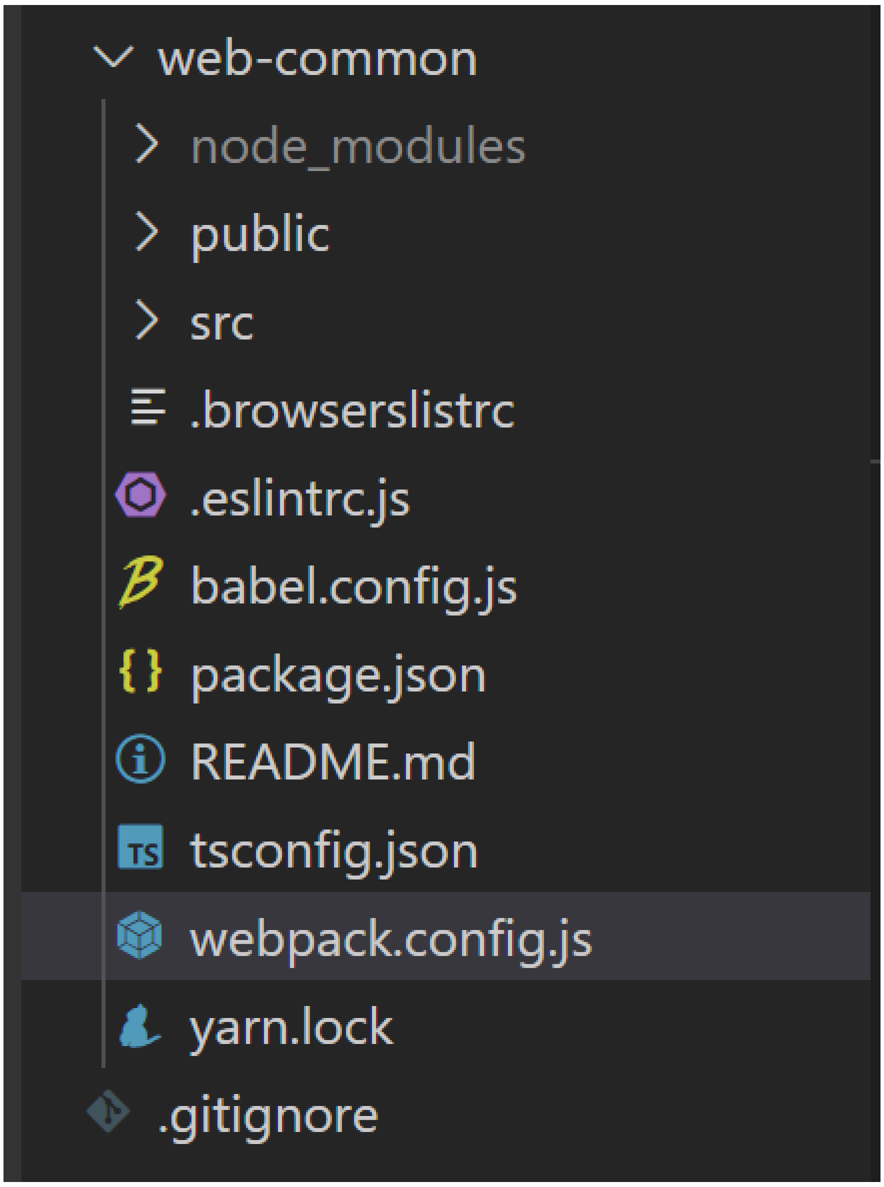
When that has been done, we would need to add a start script to our package.json. The goal of this script is to utilize Webpack-CLI when starting the server as opposed to the vue-cli. The scripts section of our package.json should now contain a start script:

```
"scripts": {

"serve": "vue-cli-service serve",

"build": "vue-cli-service build",

"start": "webpack-cli serve",

"lint": "vue-cli-service lint"

},
```

Now let's Begin…

In this blog post, we would have two projects. Micro-main is the project that needs the shared components and web-common is the project that contains the shared components.

In both projects, we would create a `webpack.config.js` file in the root directory:



```
web-common
  node_modules
  public
  src
  .browserslistrc
  .eslintrc.js
  babel.config.js
  package.json
  README.md
  tsconfig.json
  webpack.config.js
  yarn.lock
.gitignore
```

The location of the webpack.config.js file in the project.

Now that we have created our `webpack.config.js` file, we would populate it with:

Open in app

The `webpack.config.js` file of both applications should contain the following code **BUT remember to give the two projects different port variables**. By now, if you run the `yarn start` command, you should have a happy server 😄.

Now let's explain the important parts of what's been written here.

1. Entry Point: The entry point is the file that Webpack uses as the starting point. By default, Webpack uses the `src/index.js` as the entry point. But to conform with the Vue approach we set the entry point as `src/main.ts` .

2. Output: The output is where we specify where our bundled application should be placed after the build. By default, Vue applications are built into the dist directory so it's only right to place the Webpack build in the dist folder.

3. Resolve: Resolve changes how modules are resolved. It also helps in setting aliases

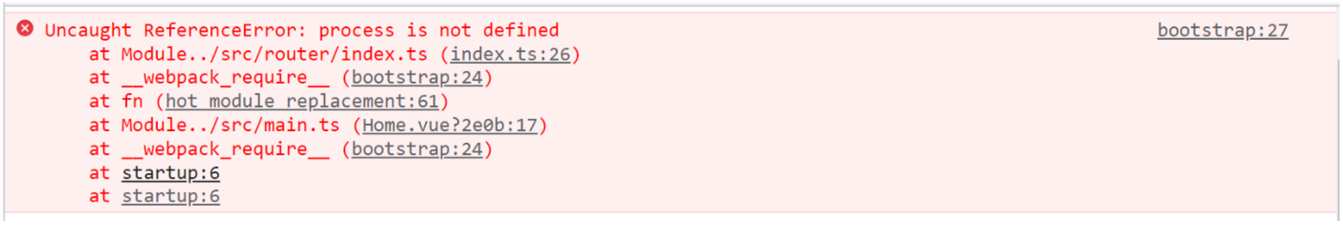
4. Rules: By default, Webpack only understands Javascript and JSON files. And that is where loaders come in. Loaders make it possible for Webpack to understand other files like typescript, Vue, CSS, e.t.c. In our case, we have added loaders for Vue, typescript, CSS, and SASS.

5. Plugins: Plugins are used to customize the Webpack build process in a variety of ways.

6. devServer: This is where the configuration for what's needed to develop the application is placed.

After populating your `webpack.config.js` , navigate to `http://localhost:${port}` . You would notice that it's an empty page without the logo of the application and the server is probably crying 😢. To fix this, replace dynamic variables in the `public/index.html` e.g:

After this, if you still have a blank page, check the console and you may see:



```
Uncaught ReferenceError: process is not defined                                    bootstrap:27
    at Module../src/router/index.ts (index.ts:26)
    at __webpack_require__ (bootstrap:24)
    at fn (hot module replacement:61)
    at Module../src/main.ts (Home.vue?2e0b:17)
    at __webpack_require__ (bootstrap:24)
    at startup:6
    at startup:6
```

An error resulting from using environment variables in the router.

To fix this issue, we would add a new plugin:

After this, you can now navigate to your website and see a happy website 👌.

Module Federation

We are done setting up Webpack and we now have happy servers and a working website. Now, it's time for us to implement the module federation.

All the sauce of the module federation is contained in the `ModuleFederationPlugin` . In this part of the configuration, the two codebases have different configurations.

The end configuration of the web_common project

1. The ModuleFederationPlugin is imported from `require("webpack").container` and not from `require("webpack")`.
2. The `exposes` object of the ModuleFederationPlugin is what's responsible for sharing code from the project. Also, the keys of the `exposes` object must be in the format of `./[KEY_NAME]`.
3. The filename can be named anything. But `remoteEntry` or `web_commonRemoteEntry` is a great name for it.
4. The name can be named anything as well, but I would give it the name of the project.

The End configuration of the main project.

Few things to note:

1. The `remotes` is where we add remotes of projects we are receiving shared code from.
2. The `web_common` is the name of the project we are receiving shared code from.
3. the remote of the project sharing code must be of the format `[PROJECT_NAME]@[PROJECT_URL]/[PROJECT_filename]` in our case, the filename is remoteEntry.js

To test the implementation, we would globally import the component. I.E:

Global importation of the HelloWorld component

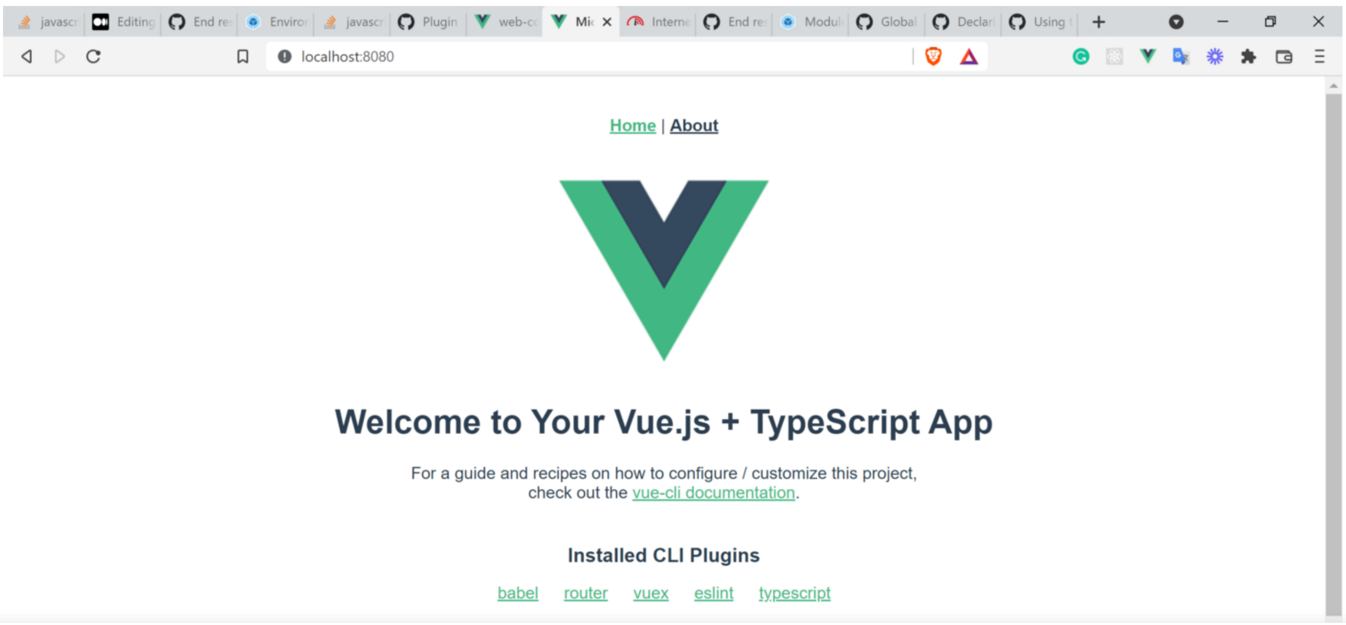
If you make use of typescript, you may need to declare `web_common` as a module.

By now, the server should be happy.

Now, the HelloWorld component can be used in the main project.

Open in app

And, Here is the result:



Welcome to Your Vue.js + TypeScript App

For a guide and recipes on how to configure / customize this project,
check out the vue-cli documentation.

Installed CLI Plugins

babel router vuex eslint typescript

One more issue....

Hurray!! we are done with the module federation… But you have probably noticed that navigating to another route and reloading the page would cause a 404 which hampers the developer experience😿.

The fix is to set the historyApiFallback to true.

```
// webpack.config.js
module.exports = {
  //...
  devServer: {
    historyApiFallback: true,
  },
};
```

Conclusion

In this blog post, I gave a clear step-by-step procedure on how to set up Webpack and how to properly set up the module federation plugin. Feel free to reach out to me if this article does not do justice to your use case.

Open in app