Building Micro-Frontends With Single-spa, React, and Vue

by Mayur Ingle CORE ·Apr. 16, 20 · Web Dev Zone · Tutorial

When I see something new and controversial like this, I always want to try it out myself to see what all of the hype is about and also so I can form my own opinions about the subject.

This led me down the path to creating a micro-frontend application that rendered two separate React applications, along with a single Vue application.

In this tutorial, I'll share what I've learned and show you how to build a micro-frontend app consisting of a React and a Vue application.

To view the final code for this application, click here.

Single SPA

The tool we will be using to create our project is Single SPA— a JavaScript framework for frontend microservices.

Single SPA enables you to use multiple frameworks in a single-page application, allowing you to split code by functionality and have Angular, React, Vue.js, etc. apps all living together.

You may be used to the days of the Create React APP CLI and the Vue CLI. With these tools, you can quickly spin up an entire project, complete with Webpack configurations, dependencies, and boilerplate ready to go for you.

If you're used to this ease of setup, then this first part may be somewhat jarring. That is because we will be creating everything from scratch, including installing all of the dependencies we need as well as creating the Webpack and Babel configuration from scratch.

Getting Started

The first thing you'll need to do is create a new folder to hold the application and change into the directory:

Shell

```
mkdir single-spa-app

cd single-spa-app
```

Next, we'll initialize a new **package.json** file:

Shell

```
npm init -y
```

Now, *this is the fun part*. We will install all of the dependencies that we will need for this project. I will split these up into separate steps.

Installing Regular Dependencies

Shell

```
npm install react react-dom single-spa single-spa-react single-spa-vue vue
```

Installing Babel Dependencies

Shell

```
npm install @babel/core @babel/plugin-proposal-object-rest-spread @babel/plugin-syntax-dynamic-import
```

Installing Webpack Dependencies

Shell

```
npm install webpack webpack-cli webpack-dev-server clean-webpack-plugin css-loader html-loader style
```

Now, all of the dependencies have been installed, and we can create our folder structure.

The main code of our app will live in an **src** directory. This **src** directory will hold subfolders for each of our applications. Let's go ahead and create the React and Vue application folders within the **src** folder:

Shell

```
mkdir src src/vue src/react
```

Now, we can create the configuration for both Webpack and Babel.

Creating Webpack Configuration

In the root of the main application, create a `webpack.config.js` file with the following code:

JavaScript

```
const path = require('path');
```

```
const webpack = require('webpack');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
const VueLoaderPlugin = require('vue-loader/lib/plugin')

module.exports = {
  mode: 'development',
  entry: {
    'single-spa.config': './single-spa.config.js',
  },
  output: {
    publicPath: '/dist/',
    filename: '[name].js',
    path: path.resolve(__dirname, 'dist'),
  },
  module: {
    rules: [
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader']
      }, {
        test: /\.js$/,
        exclude: [path.resolve(__dirname, 'node_modules')],
        loader: 'babel-loader',
      },
      {
        test: /\.vue$/,
        loader: 'vue-loader'
      }
    ],
  },
  node: {
    fs: 'empty'
  },
  resolve: {
    alias: {
      vue: 'vue/dist/vue.js'
    },
    modules: [path.resolve(__dirname, 'node_modules')],
  },
  plugins: [
    new CleanWebpackPlugin(),
    new VueLoaderPlugin()
  ],
  devtool: 'source-map',
  externals: [],
  devServer: {
    historyApiFallback: true
  }
};
```

Creating Babel Configuration

In the root of the main application, create a `.babelrc` file with the following code:

JSON

```
{
  "presets": [
    ["@babel/preset-env", {
      "targets": {
        "browsers": ["last 2 versions"]
      }
    }],
    ["@babel/preset-react"]
  ],
  "plugins": [
    "@babel/plugin-syntax-dynamic-import",
    "@babel/plugin-proposal-object-rest-spread"
  ]
}
```

Initializing Single-spa

Registering applications is how we tell **single-spa** when and how to bootstrap, mount, and unmount an application.

In the `webpack.config.js` file, we set the entry point to be `single-spa.config.js`.

Let's go ahead and create that file in the root of the project and configure it.

Single-spa.config.js

JavaScript

```
import { registerApplication, start } from 'single-spa'

registerApplication(
  'vue',
  () => import('./src/vue/vue.app.js'),
  () => location.pathname === "/react" ? false : true
);

registerApplication(
  'react',
  () => import('./src/react/main.app.js'),
  () => location.pathname === "/vue"  ? false : true
);

start();
```

This file is where you register all of the applications that will be part of the main Single Page App. Each call to `registerApplication` registers a new application and takes three arguments:

1. App name.
2. Loading function (what entry point to load).
3. Activity function (logic to tell whether to load the app).

Next, we need to create the code for each of our apps.

React App

In **src/react**, create the following two files:

Shell

```
touch main.app.js root.component.js
```

src/react/main.app.js

JavaScript

```
import React from 'react';
import ReactDOM from 'react-dom';
import singleSpaReact from 'single-spa-react';
import Home from './root.component.js';

function domElementGetter() {
  return document.getElementById("react")
}

const reactLifecycles = singleSpaReact({
  React,
  ReactDOM,
  rootComponent: Home,
  domElementGetter,
})

export const bootstrap = [
  reactLifecycles.bootstrap,
];

export const mount = [
  reactLifecycles.mount,
];

export const unmount = [
  reactLifecycles.unmount,
];
```

src/react/root.component.js

JavaScript

```
import React from "react"

const App = () => <h1>Hello from React</h1>

export default App
```

Vue app

In **src/vue**, create the following two files:

Shell

```
touch vue.app.js main.vue
```

src/vue/vue.app.js

JavaScript

```
import Vue from 'vue';
import singleSpaVue from 'single-spa-vue';
import Hello from './main.vue'

const vueLifecycles = singleSpaVue({
  Vue,
  appOptions: {
    el: '#vue',
    render: r => r(Hello)
  }
});

export const bootstrap = [
  vueLifecycles.bootstrap,
];

export const mount = [
  vueLifecycles.mount,
];

export const unmount = [
  vueLifecycles.unmount,
];
```

src/vue/main.vue

HTML

```
<template>
  <div>
      <h1>Hello from Vue</h1>
  </div>
</template>
```

Next, create the index.html file in the root of the app:

Shell

```
touch index.html
```

Index.html

HTML

```html
<html>
  <body>
    <div id="react"></div>
    <div id="vue"></div>

  </body>
</html>
```

Updating Package.json With Scripts

To run the app, let's add the start script as well as a build script in package.json:

JSON

```json
"scripts": {
  "start": "webpack-dev-server --open",
  "build": "webpack --config webpack.config.js -p"
}
```

Running the app

To run the app, run the `start` script:

Shell

```
npm start
```

Now, you can visit the following URLs:

Plain Text

```
# renders both apps
 http://localhost:8080/

# renders only react
 http://localhost:8080/react

# renders only vue
 http://localhost:8080/vue
```

Conclusion

Overall, setting up this project was fairly painless with the exception of all of the initial boilerplate setup.

I think in the future, it would be nice to have some sort of CLI that handles much of the boilerplate and initial project setup.

If you have the need for this type of architecture, Single-spadefinitely seems like the most mature way to do it as of today and was really nice to work with.