**Microfrontend**

Micro frontends is an architectural style of frontend web development where an application is split into features – micro frontends – and delivered independently.

Implementing a microfrontend architecture with Vue.js involves breaking down a large frontend application into smaller, independently developed, and deployable pieces (microfrontends). Here's a complete step-by-step guide:

1. Understand Microfrontend Basics

Before starting, understand the microfrontend concept:

Each microfrontend is an independent app with its own build process.

Communication between microfrontends is minimal or uses events/global stores.

Microfrontends can share common libraries (e.g., Vue) to reduce bundle size.

2. Decide Your Integration Strategy

Choose one of these approaches for integrating microfrontends:

**Custom integration (manual HTML, JavaScript imports):** Embed apps in an orchestrator via iframe or dynamically load JavaScript bundles.

**Module Federation (Webpack 5):** Share and load apps/modules dynamically.

**Single-SPA Framework:** A popular framework for microfrontend orchestration.

**Web Components:** Package each microfrontend as a web component.

For Vue.js, Module Federation or Single-SPA are recommended.

**3. Prerequisites**

Ensure you have:

Node.js and npm/yarn installed.

Vue CLI or Vite for Vue app development.

**4. Create a Microfrontend-Orchestrator**

This is the container application that hosts and manages the microfrontends.

Steps:

Initialize the project:

vue create microfrontend-orchestrator

Install required dependencies: If you use Single-SPA, install its CLI:

bash

Copy code

npm install -g create-single-spa

Set up dynamic imports for microfrontends (e.g., via Module Federation or Single-SPA):

For Module Federation: Configure the webpack.config.js file to load remote modules.

Example:

new ModuleFederationPlugin({

  name: "orchestrator",

  remotes: {

    microfrontend1: "microfrontend1@http://localhost:8081/remoteEntry.js",

    microfrontend2: "microfrontend2@http://localhost:8082/remoteEntry.js",

  },

  shared: ["vue", "vue-router"]

});

For Single-SPA: Use the single-spa library to import and bootstrap microfrontends.

Example:

javascript

Copy code

import { registerApplication, start } from "single-spa";

registerApplication(

  "microfrontend1",

  () => import("microfrontend1"),

  (location) => location.pathname.startsWith("/app1")

);

start();

5. Create Individual Microfrontends

Each microfrontend is a separate Vue app.

Steps:

Create the app:

bash

Copy code

vue create microfrontend1

Repeat this for other microfrontends (microfrontend2, etc.).

Modify the build process:

For Module Federation: Configure Webpack's ModuleFederationPlugin in vue.config.js:

javascript

Copy code

const { ModuleFederationPlugin } = require("webpack").container;

module.exports = {

  configureWebpack: {

    plugins: [

      new ModuleFederationPlugin({

        name: "microfrontend1",

        filename: "remoteEntry.js",

        exposes: {

          './App': './src/main.js',

        },

        shared: ["vue", "vue-router"]

      }),

    ],

  },

};

For Single-SPA: Convert the Vue app into a single-spa-compatible application:

bash

Copy code

npm install single-spa-vue

Update main.js:

javascript

Copy code

import { h, createApp } from 'vue';

import singleSpaVue from 'single-spa-vue';

import App from './App.vue';

const appOptions = {

  render() {

    return h(App);

  },

};

const vueLifecycles = singleSpaVue({

  createApp,

  appOptions,

});

export const bootstrap = vueLifecycles.bootstrap;

export const mount = vueLifecycles.mount;

export const unmount = vueLifecycles.unmount;

Expose endpoints: Ensure each microfrontend serves files on unique ports (e.g., localhost:8081, localhost:8082).

Add routing to individual apps (if needed).

6. Share Dependencies Across Microfrontends

To avoid bloating the final bundle:

Use peerDependencies in package.json.

Share Vue and other common libraries via Webpack or a CDN.

For example:

javascript

Copy code

new ModuleFederationPlugin({

  shared: {

    vue: {

      singleton: true,

      requiredVersion: "^3.2.0",

    },

    "vue-router": {

      singleton: true,

    },

  },

});

7. Test Locally

Run all microfrontends and the orchestrator locally:

Start microfrontend apps:

bash

Copy code

npm run serve

Start the orchestrator:

bash

Copy code

npm run serve

Visit the orchestrator's URL to see the composed application.

8. Deployment

Deploy microfrontends and orchestrator separately:

Use a CDN (like AWS S3 or Netlify) to host microfrontend assets.

Update orchestrator to point to the correct production URLs.

9. Add Communication Between Microfrontends

If microfrontends need to share data:

Use an event bus (e.g., via Vue's mitt library).

Or use a shared global store (e.g., pinia or vuex).

Example with mitt:

javascript

Copy code

// Event Bus (event-bus.js)

import mitt from 'mitt';

export const EventBus = mitt();

// Sending Events

EventBus.emit('eventName', data);

// Listening to Events

EventBus.on('eventName', (data) => console.log(data));

10. Optimize for Production

Enable tree-shaking and lazy loading.

Minimize shared dependencies across microfrontends.

Set up monitoring for microfrontends.

By following these steps, you’ll have a functional Vue.js microfrontend architecture. Let me know if you'd like a more specific example or help with any part of the process!

has context menu