**Tutorial on micro frontend architecture with React, module federation and rs-pack**

**Introduction**

In modern web development, building large-scale applications can become complex and unwieldy. Micro Frontends is an architectural pattern that breaks down a monolithic frontend into smaller, independent, and self-contained applications. Each micro frontend can be developed, tested, and deployed independently, enabling teams to work in parallel and scale efficiently. In this tutorial, we’ll explore the concept of micro frontends and build a project using React, TypeScript, rs-pack, and Module Federation. Rspack is a fast, Rust-based web bundler designed to be highly compatible with Webpack. It aims to provide faster build times and better performance compared to traditional JavaScript-based bundlers like Webpack.

**What Are Micro Frontends?**

Micro frontends extend the principles of microservices to the frontend. Instead of building a single, monolithic frontend application, the UI is split into smaller, independent applications. Each micro frontend:

* Can be developed and deployed independently.
* Can use different technologies (e.g., React, Vue, Angular).
* Communicates with other micro frontends through well-defined APIs or events.

This approach improves scalability, maintainability, and team autonomy.

**Overview of the Project**

We’ll build a simple application with one micro frontend and a host container:

* *About* micro frontend: displays a banner and a get more info button
* *Tutorial* container: imports the components from the micro frontend and displays them along other UI

These micro frontends will be integrated into a single application using Module Federation.

**How to Create This Architecture**

*note: you will need to have npm installed*

Step 1: Set Up the Project

We’ll use *npx create-mf-app* to scaffold the micro frontends and the host application.mf-app

* Create the Users Micro Frontend:

| npx create-mf-app |
| --- |

1. Pick a name - tutorial
2. Pick Application as project type
3. Set the port number (3000)
4. Choose react-18 as the framework
5. Choose Tailwind as the CSS framework

Repeat the process to create the *about* micro frontend (only name will differ and the port will be 3001)*.*

Step 2: Develop the Micro Frontends

First go to each folder and install the dependencies using

| yarn install |
| --- |

In the src directory in the home application, create two components, Banner.tsx and GetMoreInfoButton.tsx:

Banner.tsx:

| import React from "react" export default function Banner() {  return (  <**div** className="p-5 bg-blue-500 text-white -text-3xl font-bold">  Micro Frontend Banner  </**div**>  ) } |
| --- |

GetMoreInfoButton.tsx:

| import React from 'react';  const GetMoreInfoButton = () => {  const handleClick = () => {  alert('Here is more information!');  };   return (  <**div**>  <**button** onClick={handleClick}>Get More Info</**button**>  </**div**>  ); };  export default GetMoreInfoButton; |
| --- |

Next, let’s modify the App.tsx from the tutorial micro frontend

import React from "react";

import ReactDOM from "react-dom";

import "./index.scss";

const App = () => (

<div className="text-3xl mx-auto max-w-6xl">

<div className="text-center">

<p className="text-gray-500">Let's learn React!</p>

</div>

</div>

);

Step 3: Configure Module Federation

Module Federation allows micro frontends to share code and dependencies. Open the module-federation.config.ts file in each micro frontend and configure the ModuleFederationPlugin:

* In the *about* folder modify the *exposes* part so that our components can be shared with *tutorial*:

exposes: {

"./Banner": "./src/Banner",

"./GetMoreInfoButton": "./src/GetMoreInfoButton"

}

* In the tutorial folder modify the *remotes* part so that we can import the components from *about:*

remotes*: {*

*about: "about@http://localhost:3001/remoteEntry.js"*

*}*

Step 4: Integrate Micro Frontends in the Host Application

In the Host Application (tutorial) in App.tsx file, dynamically load the micro frontends:

* add the following imports

import Banner from "about/Banner";

import GetMoreInfoButton from "about/GetMoreInfoButton";

* add the Banner and GetMoreInfoButton components below and above the current UI
* don’t forget to add the rendering of the app at the bottom of the file:

const root = ReactDOM.createRoot(document.getElementById("app") as HTMLElement);

root.render(<App />);

Step 5: Run the Project

Start the micro frontend and the host application:

| cd about && yarn build && yarn start cd tutorial && yarn build && yarn start |
| --- |

Open the host application in your browser (http://localhost:3000). You should see the integrated micro frontend.

The tutorial code can be found at <https://github.com/DianaIfrosa/microfrontends-tutorial>

**Advantages of Micro Frontends**

* Independent Development: teams can work on different micro frontends independently.
* Technology Agnostic: each micro frontend can use different frameworks or libraries.
* Scalability: easier to scale large applications by breaking them into smaller parts.
* Faster Deployment: micro frontends can be deployed independently, reducing deployment risks.
* Improved Maintainability: smaller codebases are easier to maintain and debug.

**Conclusion**

Micro frontends are a powerful architectural pattern for building scalable and maintainable web applications. By using React, TypeScript, rs-pack, and Module Federation, we can create independent micro frontends and integrate them into a single application. This approach enables teams to work efficiently and deliver high-quality applications.