**What is Monolithic Front End (FE) architecture**

**Monolithic frontend (FE) architecture** refers to a design approach in which all parts of a web application's user interface — including components, styles, routing, and logic — are tightly integrated into a single, unified codebase. This approach mirrors traditional monolithic backend systems, where all layers of functionality (UI, logic, and data access) are centralized within one deployable unit

**Key Characteristics**

Monolithic frontend architecture typically includes the following:

* Single codebase: All frontend features, assets, and modules exist in one repository or build system, making development and deployment straightforward.​
* Tight coupling: UI components and application logic are built together, meaning changes in one area can affect others.​
* Unified build and deployment: The entire UI layer is built, tested, and deployed as one artifact (e.g., a single SPA or static bundle).​
* Shared runtime and dependencies: All modules share the same environment (framework version, state management, etc.), reducing complexity at small scale but increasing rigidity as the application grows.

**Advantages**

* Simplicity: Easier to develop initially since all code lives in one place with fewer moving parts.​
* Performance: No need for inter-module communication, leading to faster interactions within the same app runtime.​
* Unified testing and deployment: Testing is faster since all UI components are part of one system

**Disadvantages**

* ***Limited scalability:*** Difficult to scale only specific features; the entire app must be rebuilt and redeployed.​
* ***Slow iteration:*** Small changes often require redeploying the whole frontend.​
* ***Reduced flexibility:*** Large teams working concurrently face integration conflicts and code dependency issues.​
* ***Technology lock-in:*** Migrating to a new framework or tool is complex due to the app’s strong interdependencies.

**Micro Front-End (MFE)**

***A Micro Front-End (MFE)*** is an architectural style where a large front-end application is divided into smaller, independent, and self-contained pieces — each developed, tested, and deployed separately — yet all work together as one seamless user experience.

It applies the same principles of microservices (used in the backend) to the front-end world.

**Core Concept**

A micro front-end represents a distinct feature or section of a web application, such as a header, dashboard, or shopping cart, that functions as an independent application. These modules are then integrated at runtime to create a seamless experience for the user.

**Main Components**

* ***Module Composition****: The application is assembled from multiple front-end modules, each built using potentially different frameworks or technologies such as React, Angular, or Vue.​*
* ***Integration Layer:*** Responsible for combining micro front-ends through server-side, client-side, or edge-side composition, ensuring they work together cohesively.​
* ***Routing and Navigation***: Manages transitions between micro front-ends using methods like event-based communication, URL routing, or centralized routers.​
* ***Communication:*** Modules interact via Pub/Sub messaging, shared state systems, or API endpoints to exchange data and trigger actions.​
* ***Independent Deployment:*** Each micro front-end can be deployed or rolled back individually, allowing faster iterations and lower risk.

**Advantages**

* **Scalability:** Teams can scale development or add new features independently without impacting the core system.​
* **Flexibility:** Different parts of the application can use the best-fit technology or framework for their needs.​
* **Fault Isolation:** Failures in one module don’t necessarily affect others, increasing system stability.​
* **Continuous Deployment:** Enables faster CI/CD pipelines since changes in one micro front-end do not require redeploying the entire app.

**Disadvantages**

* **Complex Integration**: Ensuring multiple independent modules behave as one cohesive application can be challenging.​
* **Increased Overhead**: Managing multiple build pipelines, deployments, and repos requires more tooling and operational effort.​
* ***Performance Costs:*** If not optimized, loading several micro front-ends can impact performance due to redundant resources

**Comparison with Monolithic Frontend**

|  |  |  |
| --- | --- | --- |
| Feature | Monolithic Frontend | Micro Frontend |
| Codebase | Single unified codebase ​ | Multiple independent codebases ​ |
| Deployment | One unit deployed together ​ | Each module deployed separately ​ |
| Scalability | Harder to scale specific parts ​ | Scales per module or feature ​ |
| Technology Choice | Typically one stack ​ | Mixed frameworks possible |

**Popular Frameworks for Micro Frontends**

|  |  |  |
| --- | --- | --- |
| Framework | Description | Key Features |
| single-spa | The most popular MFE orchestrator | • Combines multiple front-end frameworks in one app• Supports React, Angular, Vue, Svelte, etc.• Handles lifecycle (mount/unmount) of micro apps |
| qiankun | Built on top of single-spa by Alibaba | • Easy setup• Supports sandboxing (isolates styles and JS)• Hot reload for sub-apps |
| Module Federation (Webpack 5) | Built into Webpack | • Enables apps to dynamically import code from other apps• Perfect for build-time integration• Widely used in React, Angular, Vue |
| Open Components (OC) | Server-side micro front-end composition | • Reusable front-end components as services• Works with Node.js backend |
| Piral | Framework for modular front-end applications | • Offers runtime loading of “pilets” (micro apps)• Works with React/Vue/Angular |
| Luigi (SAP) | Enterprise-grade micro front-end framework | • Created by SAP• Focuses on routing, authorization, and UX consistency |

**Frameworks That Can Be Used Inside MFEs**

**These are front-end frameworks that are commonly used as individual micro apps within a micro front-end setup.**

|  |  |  |
| --- | --- | --- |
| Front-End Framework | Micro Front-End Support | Integration Options |
| React | **✅ Excellent support** | **Works well with single-spa, Module Federation, Piral** |
| Angular | **✅ Excellent support** | **Has official Webpack Module Federation plugin (@angular-architects/module-federation)** |
| Vue.js | **✅ Supported** | **Integrates with single-spa-vue or Module Federation** |

In summary, micro frontend development in 2025 is dominated by frameworks like Single-SPA, Qiankun, Piral, and Luigi, with complementary technologies like Module Federation, Web Components, and Edge Composition enabling efficient integration and scalability across distributed teams.

**Which frameworks are best for React-based micro frontends?**

For React-based micro frontend architectures, several modern frameworks and tools are optimized for modularization, independent deployment, and runtime integration. The best options combine React’s component-driven model with tooling like Webpack 5 Module Federation, Single-SPA, and advanced frameworks such as Piral and Qiankun

**Compare Module Federation and single-spa for React micro frontends**

When building React-based micro frontends, both Webpack Module Federation and Single-SPA are top solutions—but they serve different purposes within the architecture. **Module Federation focuses on *code sharing and runtime integration***, **while Single-SPA manages *application orchestration and lifecycle control*.**

**Core Comparison: Module Federation vs Single-SPA**

|  |  |  |
| --- | --- | --- |
| Aspect | Module Federation | Single-SPA |
| Primary Function | **Shares and loads modules dynamically between independently deployed React apps ​.** | **Manages the lifecycle, routing, and orchestration of multiple micro frontends in one shell app ​.** |
|  |  |  |
|  |  |  |
| Integration Level | **Code-level integration (individual components or libraries shared at runtime) ​.** | **Application-level integration (mounts entire microapps into one unified SPA) ​.** |
| Lifecycle Management | **Does not include lifecycle control; relies on host app ​.** | **Provides hooks (mount, unmount, update) for lifecycle management ​.** |
| Routing | **Custom setup required; usually handled by React Router or root app ​.** | **Built-in routing mechanism supports modular navigation between micro frontends ​.** |
| Technology Compatibility | **Works best when all micro frontends use the same framework (e.g., React + React) ​.** | **Supports multiple frameworks simultaneously (React, Angular, Vue, Svelte) ​.** |
| Deployment Model | **Each module deployed independently and fetched at runtime as a remote module ​.** | **Each micro app deployed separately but orchestrated together by a root app ​.** |
| Performance | **Smaller bundles and faster load times via shared dependency caching ​.** | **Loads only active micro frontends dynamically, optimizing runtime performance ​.** |
| Setup Complexity | **Easier for small, single-framework projects (mostly React-based) ​.** | **Requires more configuration but scales better for larger heterogeneous systems ​.** |
| Best Use Case | **Multiple React apps sharing components or libs like common UI or state management ​.** | **Apps needing framework independence, lifecycle control, or complex route-based composition ​.** |

**When to Use Each**

* Choose Module Federation if all micro frontends are React-based and you need runtime code sharing, smaller bundles, and independent deployments without complex orchestration.  
  Ideal for: Component-level reuse between applications such as a shared design system, authentication library, or analytics module.​
* Choose Single-SPA if you need a robust orchestration layer that manages routes, app mounting, and lifecycle events across multiple independently deployed React apps—or even mixed frameworks.

**Combining Both**

Modern architectures often combine Single-SPA with Module Federation for greater flexibility:

* Single-SPA orchestrates micro frontend lifecycles and routing.
* Module Federation shares React components, hooks, or libraries across those micro frontends at runtime.  
  This hybrid model delivers both team autonomy and optimized runtime performance, offering “the best of both worlds” for large-scale enterprise frontends.​

In summary, Module Federation excels in runtime code sharing within the same tech stack, while Single-SPA provides orchestration and lifecycle management across diverse or complex frontend systems. For React-only ecosystems, Module Federation may suffice, but combining both results in optimal scalability and resource efficiency

**Get Coding Started:**

**let’s build a tiny, working Micro Frontend example using Webpack 5 Module Federation.  
We’ll create two React apps:**

* **remote-app — exposes a simple Button component.**
* **host-app — consumes that remote Button and renders it.**

**1 — Project structure**

**Create a workspace folder, then two subfolders:**

**mfe-demo/**

**host-app/**

**remote-app/**

**2 — Common notes**

* **This example uses Webpack 5, Babel and React.**
* **Ports:**
  + **remote-app → 3001**
  + **host-app → 3000**
* **remote-app will expose ./Button via remoteEntry.js.**
* **host-app will import the remote module at runtime.**

**3 — remote-app (exposes component)**

**Create mfe-demo/remote-app and add the following files.**

remote-app/package.json

{

"name": "remote-app",

"version": "1.0.0",

"private": true,

"scripts": {

"start": "webpack serve --config webpack.config.js --mode development",

"build": "webpack --config webpack.config.js --mode production"

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"@babel/core": "^7.22.0",

"@babel/preset-env": "^7.22.0",

"@babel/preset-react": "^7.22.0",

"babel-loader": "^9.1.2",

"html-webpack-plugin": "^5.5.0",

"webpack": "^5.85.0",

"webpack-cli": "^5.1.0",

"webpack-dev-server": "^4.15.0"

}

**}**

**remote-app/webpack.config.js**

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

**const HtmlWebpackPlugin = require('html-webpack-plugin');**

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

**module.exports = {**

**entry: path.resolve(\_\_dirname, './src/index'),**

**mode: 'development',**

**devServer: {**

**port: 3001,**

**historyApiFallback: true**

**},**

**output: {**

**publicPath: 'auto'**

**},**

**module: {**

**rules: [**

**{**

**test: /\.(js|jsx)$/,**

**loader: 'babel-loader',**

**exclude: /node\_modules/**

**}**

**]**

**},**

**resolve: {**

**extensions: ['.js', '.jsx']**

**},**

**plugins: [**

**new ModuleFederationPlugin({**

**name: 'remote\_app',**

**filename: 'remoteEntry.js',**

**exposes: {**

**'./Button': './src/Button' // expose Button component**

**},**

**shared: {**

**react: { singleton: true, requiredVersion:false,eager: true },**

**'react-dom': { singleton: true, requiredVersion: false,eager: true }**

**}**

**}),**

**new HtmlWebpackPlugin({**

**template: './public/index.html'**

**})**

**]**

**};**

**remote-app/.babelrc**

**{**

**"presets": ["@babel/preset-env", "@babel/preset-react"]**

**}**

**remote-app/public/index.html**

**<!doctype html>**

**<html>**

**<head>**

**<meta charset="utf-8" />**

**<title>Remote App</title>**

**</head>**

**<body>**

**<div id="root"></div>**

**</body>**

**</html>**

**remote-app/src/index.jsx**

import React from 'react';

import { createRoot } from 'react-dom/client';

import Button from './Button';

const App = () => (

<div style={{ padding: 20 }}>

<h2>Remote App (exposes Button)</h2>

<Button onClick={() => alert('Hello from remote Button!')}>Click me</Button>

</div>

);

createRoot(document.getElementById('root')).render(<App />);

**4 — host-app (consumes remote)**

Create mfe-demo/host-app and add files below.

host-app/package.json

{

"name": "host-app",

"version": "1.0.0",

"private": true,

"scripts": {

"start": "webpack serve --config webpack.config.js --mode development",

"build": "webpack --config webpack.config.js --mode production"

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"@babel/core": "^7.22.0",

"@babel/preset-env": "^7.22.0",

"@babel/preset-react": "^7.22.0",

"babel-loader": "^9.1.2",

"html-webpack-plugin": "^5.5.0",

"webpack": "^5.85.0",

"webpack-cli": "^5.1.0",

"webpack-dev-server": "^4.15.0"

}

}

host-app/webpack.config.js

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

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

module.exports = {

  entry: path.resolve(\_\_dirname, './src/index'),

  mode: 'development',

  devServer: {

    port: 3000,

    historyApiFallback: true

  },

  output: {

    publicPath: 'auto'

  },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        loader: 'babel-loader',

        exclude: /node\_modules/

      }

    ]

  },

  resolve: {

    extensions: ['.js', '.jsx']

  },

  plugins: [

    new ModuleFederationPlugin({

      name: 'host\_app',

      remotes: {

        // remote\_app is loaded from remoteEntry.js of remote-app on port 3001

        remote\_app: 'remote\_app@http://localhost:3001/remoteEntry.js'

      },

      shared: {

        react: { singleton: true, requiredVersion:false,eager: true },

        'react-dom': { singleton: true, requiredVersion: false,eager: true }

      }

    }),

    new HtmlWebpackPlugin({

      template: './public/index.html'

    })

  ]

};

host-app/.babelrc

{

"presets": ["@babel/preset-env", "@babel/preset-react"]

}

**host-app/public/index.html**

<!doctype html>

<html>

<head>

<meta charset="utf-8" />

<title>Host App</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

**host-app/src/index.jsx**

import React, { Suspense } from 'react';

import { createRoot } from 'react-dom/client';

// dynamic import of a federated module

const RemoteButton = React.lazy(() => import('remote\_app/Button'));

const App = () => {

return (

<div style={{ padding: 20 }}>

<h1>Host App</h1>

<p>This app consumes <strong>Button</strong> exported by remote\_app.</p>

<Suspense fallback={<div>Loading remote component...</div>}>

<RemoteButton onClick={() => alert('Hello from remote Button inside Host!')}>

Federated Button

</RemoteButton>

</Suspense>

</div>

);

};

createRoot(document.getElementById('root')).render(<App />);

**5 — Install & Run**

From your terminal, run these commands.

**Install dependencies**

Open two terminals (one per project) OR run sequentially.

Terminal A — for remote-app:

cd mfe-demo/remote-app

npm install

npm start

This starts remote on http://localhost:3001/.

Terminal B — for host-app:

cd mfe-demo/host-app

npm install

npm start

This starts host on http://localhost:3000/.

Now open http://localhost:3000/. You should see the Host app and the federated Button rendered from the remote app. The remote runs separately and exposes remoteEntry.js which host loads.

**🧩 What’s happening inside Module Federation**

Module Federation allows multiple **separate builds** (called *remotes* and *hosts*) to share modules **at runtime** — for example, sharing **React** between apps so only one copy is loaded.

When your apps start:

1. The **host** loads its bundle.
2. Then it dynamically downloads the **remote’s remoteEntry.js** file.
3. Module Federation then looks at the **shared configuration** in both apps to decide how to load common libraries (like React).

**⚙️ Example of how sharing works**

Let’s say both apps define:

shared: {

react: { singleton: true },

'react-dom': { singleton: true },

}

Now when your host imports something from the remote:

import('remote\_app/Button')

Webpack does this internally:

* It **waits** until runtime (in the browser)
* Then it checks: “Is React already loaded?”
* If yes → use the host’s React
* If not → load the remote’s React

This “lazy resolution” ensures **only one React instance** is active and shared safely.

**Shared State Management In Micro Front End**

**Common Patterns for Shared State**

**Shared Store as a Federated Module**

* Expose a **state management module** (like Redux, Zustand, Recoil, or RxJS store) from a remote.
* Other apps import it via **Module Federation**.
* Single source of truth across apps.

**Example: Redux store shared via Module Federation**

A **full working example of a Shared Redux Store as a Federated Module** using **Webpack 5 Module Federation** and **React 18**.

We’ll have:

* **remote-app** → exposes the Redux store and actions
* **host-app** → consumes the shared store and dispatches actions

Continue the same step of previous project

**2️⃣ remote-app setup**

**remote-app/package.json**

{

  "name": "remote-app",

  "version": "1.0.0",

  "private": true,

  "scripts": {

    "start": "webpack serve --config webpack.config.js --mode development",

    "build": "webpack --config webpack.config.js --mode production"

  },

  "dependencies": {

    "@reduxjs/toolkit": "^2.9.2",

    "react": "^18.2.0",

    "react-dom": "^18.2.0",

    "react-redux": "^9.2.0"

  },

  "devDependencies": {

    "@babel/core": "^7.22.0",

    "@babel/preset-env": "^7.22.0",

    "@babel/preset-react": "^7.22.0",

    "babel-loader": "^9.1.2",

    "html-webpack-plugin": "^5.5.0",

    "webpack": "^5.85.0",

    "webpack-cli": "^5.1.0",

    "webpack-dev-server": "^4.15.0"

  }

}

**remote-app/webpack.config.js**

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

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

module.exports = {

  entry: path.resolve(\_\_dirname, './src/index'),

  mode: 'development',

  devServer: {

    port: 3001,

    historyApiFallback: true

  },

  output: {

    publicPath: 'auto'

  },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        loader: 'babel-loader',

        exclude: /node\_modules/

      }

    ]

  },

  resolve: {

    extensions: ['.js', '.jsx']

  },

  plugins: [

    new ModuleFederationPlugin({

      name: 'remote\_app',

      filename: 'remoteEntry.js',

      exposes: {

        './Button': './src/Button', // expose Button component

        './Store': './src/store' // Expose the Redux store

      },

      shared: {

        react: { singleton: true, requiredVersion: false, eager: true },

        'react-dom': { singleton: true, requiredVersion: false, eager: true },

        '@reduxjs/toolkit': { singleton: true, requiredVersion: false, eager: true },

        'react-redux': { singleton: true, requiredVersion: false, eager: true }

      }

    }),

    new HtmlWebpackPlugin({

      template: './public/index.html'

    })

  ]

};

**remote-app/src/index.js (optional local demo)**

import React from 'react';

import { createRoot } from 'react-dom/client';

import { Provider, useSelector, useDispatch } from 'react-redux';

import { store, addItem } from './store';

const App = () => {

  const cart = useSelector(state => state.cart);

  const dispatch = useDispatch();

  return (

    <div style={{ padding: 20 }}>

      <h2>Remote App Store Demo</h2>

      <button onClick={() => dispatch(addItem({ id: 1, name: 'Apple' }))}>

        Add Apple

      </button>

      <pre>{JSON.stringify(cart, null, 2)}</pre>

    </div>

  );

};

const root = createRoot(document.getElementById('root'));

root.render(

  <Provider store={store}>

    <App />

  </Provider>

);

**3️⃣ host-app setup**

**host-app/package.json**

{

  "name": "host-app",

  "version": "1.0.0",

  "private": true,

  "scripts": {

    "start": "webpack serve --config webpack.config.js --mode development",

    "build": "webpack --config webpack.config.js --mode production"

  },

  "dependencies": {

    "@reduxjs/toolkit": "^2.9.2",

    "react": "^18.2.0",

    "react-dom": "^18.2.0",

    "react-redux": "^9.2.0"

  },

  "devDependencies": {

    "@babel/core": "^7.22.0",

    "@babel/preset-env": "^7.22.0",

    "@babel/preset-react": "^7.22.0",

    "babel-loader": "^9.1.2",

    "html-webpack-plugin": "^5.5.0",

    "webpack": "^5.85.0",

    "webpack-cli": "^5.1.0",

    "webpack-dev-server": "^4.15.0"

  }

}

**host-app/webpack.config.js**

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

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

module.exports = {

  entry: path.resolve(\_\_dirname, './src/index'),

  mode: 'development',

  devServer: {

    port: 3000,

    historyApiFallback: true

  },

  output: {

    publicPath: 'auto'

  },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        loader: 'babel-loader',

        exclude: /node\_modules/

      }

    ]

  },

  resolve: {

    extensions: ['.js', '.jsx']

  },

  plugins: [

    new ModuleFederationPlugin({

      name: 'host\_app',

      remotes: {

        // remote\_app is loaded from remoteEntry.js of remote-app on port 3001

        remote\_app: 'remote\_app@http://localhost:3001/remoteEntry.js'

      },

      shared: {

        react: { singleton: true, requiredVersion: false, eager: true },

        'react-dom': { singleton: true, requiredVersion: false, eager: true },

        '@reduxjs/toolkit': { singleton: true, requiredVersion: false, eager: true },

        'react-redux': { singleton: true, requiredVersion: false, eager: true }

      }

    }),

    new HtmlWebpackPlugin({

      template: './public/index.html'

    })

  ]

};

**host-app/public/index.html**

<!doctype html>

<html>

<head>

<meta charset="utf-8" />

<title>Host App</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

**host-app/src/index.js (consume remote store)**

import React, { Suspense, useEffect } from 'react';

import { createRoot } from 'react-dom/client';

import { Provider, useSelector, useDispatch } from 'react-redux';

const RemoteStorePromise = import('remote\_app/Store');

const RemoteApp = () => {

    const [store, setStore] = React.useState(null);

    useEffect(() => {

        RemoteStorePromise.then(mod => setStore(mod.store));

    }, []);

    if (!store) return <div>Loading Remote Store...</div>;

    const App = () => {

        const cart = useSelector(state => state.cart);

        const dispatch = useDispatch();

        return (

            <div style={{ padding: 20 }}>

                <h2>Host App using Remote Store</h2>

                <button onClick={() => dispatch({ type: 'cart/addItem', payload: { id: 2, name: 'Banana' } })}>

                    Add Banana

                </button>

                <pre>{JSON.stringify(cart, null, 2)}</pre>

            </div>

        );

    };

    return (

        <Provider store={store}>

            <App />

        </Provider>

    );

};

const root = createRoot(document.getElementById('root'));

root.render(<RemoteApp />);

**4️⃣ Run the apps**

1. **Install dependencies**:

cd remote-app

npm install

cd ../host-app

npm install

1. **Start remote first**:

cd remote-app

npm start

1. **Start host**:

cd host-app

npm start

1. Open **host app** at http://localhost:3000 → you should see the Host App UI with **shared Redux store**.

* Clicking buttons in **host** updates the **remote store**
* You can also open http://localhost:3001 (remote) to see its state reflecting changes if you render the store locally.

**✅ Key Notes**

* **Singleton**: Both host & remote must share react, react-dom, react-redux, and @reduxjs/toolkit as **singletons**.
* **Lazy import**: Remote store must be dynamically imported (import('remote\_app/Store')) to avoid eager errors.
* **Single source of truth**: Both apps are using the **same store instance**.

**React routing with micro front ends with Module Federation**

combine **React Router** with **Micro Frontends (Module Federation)** so that the **host** can route to **different remote apps** or components

**⚙️ Goal**

* **Host App**: React + React Router handles routes.
* **Remote Apps**: Each remote exposes React components (pages) via Module Federation.
* The **host routes** load the remote component lazily.

🧱 Folder Structure

/microfrontend-demo /

shell /

app1 /

app2

**1. Shell (Host)**

shell/package.json

{

  "name": "shell",

  "version": "1.0.0",

  "private": true,

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.28.5",

    "@babel/preset-env": "^7.28.5",

    "@babel/preset-react": "^7.28.5",

    "babel-loader": "^10.0.0",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

**shell/webpack.config.js**

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

module.exports = {

mode: "development",

entry: "./src/index.js",

devServer: { port: 3000, historyApiFallback: true },

output: { publicPath: "auto" },

plugins: [

new ModuleFederationPlugin({

name: "shell",

remotes: {

app1: "app1@http://localhost:3001/remoteEntry.js",

app2: "app2@http://localhost:3002/remoteEntry.js"

},

shared: { react: { singleton: true }, "react-dom": { singleton: true } }

}),

new HtmlWebpackPlugin({ template: "./public/index.html" })

],

};

**Shell/.babelrc**

{

  "presets": ["@babel/preset-env", "@babel/preset-react"]

}

**shell/src/index.js**

import("./bootstrap");

**shell/src/bootstrap.js**

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

const root = createRoot(document.getElementById("root"));

root.render(<App />);

**shell/src/App.js**

import React, { Suspense } from "react";

import { BrowserRouter, Routes, Route, NavLink } from "react-router-dom";

const App1 = React.lazy(() => import("app1/App"));

const App2 = React.lazy(() => import("app2/App"));

export default function App() {

  return (

    <BrowserRouter>

      <nav style={{ margin: 10 }}>

        <NavLink to="/app1" style={{ margin: 5 }}>App1</NavLink>

        <NavLink to="/app2" style={{ margin: 5 }}>App2</NavLink>

      </nav>

      <Suspense fallback={<div>Loading...</div>}>

        <Routes>

          <Route path="/app1/\*" element={<App1 />} />

          <Route path="/app2/\*" element={<App2 />} />

        </Routes>

      </Suspense>

    </BrowserRouter>

  );

}

**shell/public/index.html**

<!DOCTYPE html>

<html>

<head><title>Shell App</title></head>

<body><div id="root"></div></body>

</html>

**2. App1 (Remote)**

**app1/package.json**

{

  "name": "app1",

  "version": "1.0.0",

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.28.5",

    "@babel/preset-env": "^7.28.5",

    "@babel/preset-react": "^7.28.5",

    "babel-loader": "^10.0.0",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

**app1/webpack.config.js**

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

const path = require("path");

module.exports = {

  mode: "development",

  entry: "./src/index.js",

  devServer: { port: 3001 },

  output: { publicPath: "auto" },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        exclude: /node\_modules/,

        use: { loader: "babel-loader" }

      }

    ]

  },

  resolve: {

    extensions: [".js", ".jsx"]

  },

  plugins: [

    new ModuleFederationPlugin({

      name: "app1",

      filename: "remoteEntry.js",

      exposes: { "./App": "./src/App" },

      shared: { react: { singleton: true }, "react-dom": { singleton: true } }

    }),

    new HtmlWebpackPlugin({ template: "./public/index.html" })

  ]

};

**App1/.babelrc**

{

  "presets": ["@babel/preset-env", "@babel/preset-react"]

}

**app1/src/index.js**

**import**("./bootstrap");

**app1/src/bootstrap.js**

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

createRoot(document.getElementById("root")).render(<App />);

**app1/src/App.js**

import React from "react";

import { MemoryRouter, Routes, Route, Link } from "react-router-dom";

const Home = () => <h2>App1 Home</h2>;

const About = () => <h2>App1 About</h2>;

export default function App() {

  return (

    <MemoryRouter>

      <div>

        <h1>App 1</h1>

        <Link to="/">Home</Link> | <Link to="/about">About</Link>

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/about" element={<About />} />

        </Routes>

      </div>

    </MemoryRouter>

  );

}

**App1/public/index.html**

<!DOCTYPE html>

<html>

<head><title>Shell App</title></head>

<body><div id="root"></div></body>

</html>

**3. App2 (Remote)**

**app2/package.json**

{

  "name": "app2",

  "version": "1.0.0",

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.28.5",

    "@babel/preset-env": "^7.28.5",

    "@babel/preset-react": "^7.28.5",

    "babel-loader": "^10.0.0",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

**app2/webpack.config.js**

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

const path = require("path");

module.exports = {

  mode: "development",

  entry: "./src/index.js",

  devServer: { port: 3002 },

  output: { publicPath: "auto" },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        exclude: /node\_modules/,

        use: { loader: "babel-loader" }

      }

    ]

  },

  resolve: {

    extensions: [".js", ".jsx"]

  },

  plugins: [

    new ModuleFederationPlugin({

      name: "app2",

      filename: "remoteEntry.js",

      exposes: { "./App": "./src/App" },

      shared: { react: { singleton: true }, "react-dom": { singleton: true } }

    }),

    new HtmlWebpackPlugin({ template: "./public/index.html" })

  ]

};

**App2/.babelrc**

{

  "presets": ["@babel/preset-env", "@babel/preset-react"]

}

**App2/src/index.js**

**import**("./bootstrap");

**app2/src/bootstrap.js**

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

createRoot(document.getElementById("root")).render(<App />);

**app2/src/App.js**

import React from "react";

import { MemoryRouter, Routes, Route, Link } from "react-router-dom";

const Dashboard = () => <h2>Dashboard</h2>;

const Profile = () => <h2>Profile</h2>;

export default function App() {

return (

<MemoryRouter>

<div>

<h1>App 2</h1>

<Link to="/">Dashboard</Link> | <Link to="/profile">Profile</Link>

<Routes>

<Route path="/" element={<Dashboard />} />

<Route path="/profile" element={<Profile />} />

</Routes>

</div>

</MemoryRouter>

);

}

**App2/public/index.html**

<!DOCTYPE html>

<html>

<head><title>Shell App</title></head>

<body><div id="root"></div></body>

</html>

**Run setup**

From the root directory:

bash

cd app1 && npm install && npm start

cd app2 && npm install && npm start

cd shell && npm install && npm start

Then open http://localhost:3000.

**Pub /Sub Pattern in Micro Front End**

/microfrontends-demo /

shell /

app1 /

app2 /

**2. Shell (Host App)**

Port: 3000  
Purpose: Loads App1 and App2 dynamically and exposes shared utilities.

Shell/package.json

{

  "name": "shell",

  "version": "1.0.0",

  "private": true,

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.28.5",

    "@babel/preset-env": "^7.28.5",

    "@babel/preset-react": "^7.28.5",

    "babel-loader": "^10.0.0",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

**Shell/.babelrc**

{

  "presets": ["@babel/preset-env", "@babel/preset-react"]

}

**Shell/webconfig.js**

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

module.exports = {

  mode: "development",

  entry: "./src/index.js",

  devServer: { port: 3000, historyApiFallback: true },

  output: { publicPath: "auto" },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        exclude: /node\_modules/,

        use: { loader: "babel-loader" }

      }

    ]

  },

  resolve: {

    extensions: [".js", ".jsx"]

  },

  plugins: [

    new ModuleFederationPlugin({

      name: "shell",

      filename: "remoteEntry.js",

      exposes: {

**"./events": "./src/events", // 👈 must exist here**

      },

      remotes: {

**app1: "app1@http://localhost:3001/remoteEntry.js",**

**app2: "app2@http://localhost:3002/remoteEntry.js"**

      },

      shared: { react: { singleton: true }, "react-dom": { singleton: true } }

    }),

    new HtmlWebpackPlugin({ template: "./public/index.html" })

  ],

};

Shell/public/index.html

<!DOCTYPE html>

<html>

  <head><title>Shell</title></head>

  <body><div id="root"></div></body>

</html>

Shell/event.js

// export const EventBus = {

//   publish: (event, data) => {

//     console.log("[Bus] Publishing event:", event, data);

//     window.dispatchEvent(new CustomEvent(event, { detail: data }));

//   },

//   subscribe: (event, callback) => {

//     const handler = (e) => {

//       console.log("[Bus] Received event:", event, e.detail);

//       callback(e.detail);

//     };

//     window.addEventListener(event, handler);

//     return () => window.removeEventListener(event, handler);

//   },

// };

// Global shared store with queuing

if (!window.\_\_stateStore\_\_) {

  window.\_\_stateStore\_\_ = {

    queue: [], // holds messages until consumed

    listeners: {},

  };

}

export const StateStore = {

  publish: (event, data) => {

    const message = { event, data, timestamp: Date.now() };

    window.\_\_stateStore\_\_.queue.push(message);

    // Trigger all listeners subscribed to this event

    (window.\_\_stateStore\_\_.listeners[event] || []).forEach((cb) => cb(data));

  },

  subscribe: (event, callback) => {

    // Register listener

    if (!window.\_\_stateStore\_\_.listeners[event]) {

      window.\_\_stateStore\_\_.listeners[event] = [];

    }

    window.\_\_stateStore\_\_.listeners[event].push(callback);

    console.log(`[Queue] Subscribed to ${event}`);

    // Replay past events from queue (so late consumers catch up)

    window.\_\_stateStore\_\_.queue

      .filter((msg) => msg.event === event)

      .forEach((msg) => callback(msg.data));

    return () => {

      window.\_\_stateStore\_\_.listeners[event] =

        window.\_\_stateStore\_\_.listeners[event].filter((cb) => cb !== callback);

    };

  },

};

**Here’s how you can add a queueing state mechanism between your React micro frontends using a shared global queue like window.\_\_stateStore\_\_.  
This ensures that when one MFE publishes an event before another MFE is mounted, the new one can still consume queued updates**

**Overview**

**We’ll build a small message queue sitting on the global window object, accessible by all micro frontends:**

* **Any microfrontend can queue state updates or messages.**
* **Subscribers can read past events and listen for new ones.**
* **Works even when MFEs mount asynchronously.**

**Shell/bootstrap.js**

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

const root = createRoot(document.getElementById("root"));

root.render(<App />);

**shell/index.js**

import("./bootstrap");

**shell/App.js**

import React, { Suspense } from "react";

import { BrowserRouter, Routes, Route, NavLink } from "react-router-dom";

const App1 = React.lazy(() => import("app1/App"));

const App2 = React.lazy(() => import("app2/App"));

export default function Shell() {

  return (

    <BrowserRouter>

      <nav style={{ margin: 10 }}>

        <NavLink to="/app1" style={{ margin: 5 }}>App1</NavLink>

        <NavLink to="/app2" style={{ margin: 5 }}>App2</NavLink>

      </nav>

      <Suspense fallback={<div>Loading...</div>}>

        <Routes>

          <Route path="/app1/\*" element={<App1 />} />

          <Route path="/app2/\*" element={<App2 />} />

        </Routes>

      </Suspense>

    </BrowserRouter>

  );

}

**App1**

App1/package.json

{

  "name": "app1",

  "version": "1.0.0",

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.28.5",

    "@babel/preset-env": "^7.28.5",

    "@babel/preset-react": "^7.28.5",

    "babel-loader": "^10.0.0",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

App1/webpack.config.js

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

const path = require("path");

module.exports = {

  mode: "development",

  entry: "./src/index.js",

  devServer: { port: 3001 },

  output: { publicPath: "auto" },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        exclude: /node\_modules/,

        use: { loader: "babel-loader" }

      }

    ]

  },

  resolve: {

    extensions: [".js", ".jsx"]

  },

  plugins: [

    new ModuleFederationPlugin({

      name: "app1",

      filename: "remoteEntry.js",

      remotes: {

        shell: "shell@http://localhost:3000/remoteEntry.js", // 👈 must include shell remote

      },

      exposes: { "./App": "./src/App" },

      shared: { react: { singleton: true }, "react-dom": { singleton: true } }

    }),

    new HtmlWebpackPlugin({ template: "./public/index.html" })

  ]

};

App1/.babelrc

{

  "presets": ["@babel/preset-env", "@babel/preset-react"]

}

**App1/src/bootstrap.js**

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

const root = createRoot(document.getElementById("root"));

root.render(<App />);

**App1/src/index.js**

import("./bootstrap");

App1/src/App.js

import React from "react";

import { StateStore } from "shell/events"; // 👈 now resolvable

export default function App() {

  const handleAddToCart = () => {

    const product = { id: Date.now(), name: "Wireless Mouse" };

    StateStore.publish("ADD\_TO\_CART", product);

  };

  return (

    <div>

      <h2>Catalog (App 1)</h2>

      <button onClick={handleAddToCart}>Add to Cart</button>

    </div>

  );

}

**App1/public/index.html**

<!DOCTYPE html>

<html>

  <head><title>App 1</title></head>

  <body><div id="root"></div></body>

</html>

**App2**

**App2/package.json**

{

  "name": "app2",

  "version": "1.0.0",

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.28.5",

    "@babel/preset-env": "^7.28.5",

    "@babel/preset-react": "^7.28.5",

    "babel-loader": "^10.0.0",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

**App2/webpack.config.js**

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

const path = require("path");

module.exports = {

  mode: "development",

  entry: "./src/index.js",

  devServer: { port: 3002 },

  output: { publicPath: "auto" },

  module: {

    rules: [

      {

        test: /\.(js|jsx)$/,

        exclude: /node\_modules/,

        use: { loader: "babel-loader" }

      }

    ]

  },

  resolve: {

    extensions: [".js", ".jsx"]

  },

  plugins: [

    new ModuleFederationPlugin({

      name: "app2",

      filename: "remoteEntry.js",

      remotes: {

        shell: "shell@http://localhost:3000/remoteEntry.js", // 👈 must include shell remote

      },

      exposes: { "./App": "./src/App" },

      shared: { react: { singleton: true }, "react-dom": { singleton: true } }

    }),

    new HtmlWebpackPlugin({ template: "./public/index.html" })

  ]

};

**App2/.babelrc**

{

  "presets": ["@babel/preset-env", "@babel/preset-react"]

}

**App2/src/bootstrap.js**

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

const root = createRoot(document.getElementById("root"));

root.render(<App />);

**App2/src/index.js**

import("./bootstrap");

**App2/src/App.js**

import React, { useState, useEffect } from "react";

import { StateStore } from "shell/events";

export default function App() {

  const [cart, setCart] = useState([]);

  useEffect(() => {

    const unsubscribe = StateStore.subscribe("ADD\_TO\_CART", (product) => {

      setCart((prev) => [...prev, product]);

    });

    return unsubscribe;

  }, []);

  return (

    <div>

      <h2>Cart (App 2)</h2>

      <ul>

        {cart.map((item) => <li key={item.id}>{item.name}</li>)}

      </ul>

    </div>

  );

}

**Test Flow**

1. Run Shell, App1, and App2 simultaneously on ports 3000, 3001, and 3002.
2. Open <http://localhost:3000/app1> → click *Add to Cart* multiple times.
3. Then navigate to <http://localhost:3000/app2> → you’ll instantly see all queued items in the cart.

**Authentication state sharing in micro front end**

In a micro frontend architecture, sharing authentication state consistently across multiple independently deployed applications is essential yet tricky — because each microfrontend (MFE) must remain isolated but still recognize a common authenticated user. Below are several proven architectural strategies and a working implementation example using a shared global auth service.

**1. Common Authentication Patterns in Micro Frontends**

**Pattern A: Centralized Authentication Service**

A single sign-on (SSO) style service handles login, token storage, and refresh logic.  
All MFEs query this shared service, ensuring a single source of truth for auth state.

Characteristics

* One global authentication entry (e.g., /auth/ or hosted SSO service).
* Uses secure tokens in cookies or memory.
* The shell or shared module exposes an interface like AuthService.getUser() or AuthService.isAuthenticated().

Typical Tech: OAuth2, OIDC with Keycloak, Auth0, AWS Cognito, or a custom token API.

**Pattern B: Shared Auth Store via Module Federation**

When using Webpack Module Federation, the host can expose an authentication module (authStore.js) that all remotes use.

Each MFE accesses the same in-memory state through a singleton import (e.g., import { AuthStore } from "shell/authStore";).

This pattern guarantees instant propagation of auth updates (login/logout) to all MFEs without re-login or token duplication.

**Pattern C: Distributed via Web Storage (LocalStorage or SessionStorage)**

Each MFE reads the same token from localStorage (or from a cookie) and constructs its own state.

Example:

js

**const** token = localStorage.getItem("auth\_token");

Pros: Simple setup, works across reloads.  
Cons: No automatic reactive updates; other microfrontends must poll or listen for storage events.

**Pattern D: Global State Queue or Event Bus**

A lightweight solution: treating authentication updates as global events, e.g. LOGIN\_SUCCESS or LOGOUT.

Publisher (Shell or MFE handling login) fires:

js

window.dispatchEvent(**new** CustomEvent("AUTH\_UPDATE", { detail: { user, token } }));

Subscriber MFEs listen and sync:

js

window.addEventListener("AUTH\_UPDATE", (e) => updateAuth(e.detail));

This keeps MFEs decoupled and framework-agnostic.​

**Comparison of Authentication Sharing Methods**

| **Method** | **Isolation** | **Real-Time Sync** | **Persistence** | **Cross-Framework Compatible** |
| --- | --- | --- | --- | --- |
| Shared AuthService (Federation) | Medium | ✅ Yes | ✅ Yes | ⚠️ React/JS-based |
| Event Bus (CustomEvent) | High | ✅ Yes | ⚠️ No | ✅ Yes |
| LocalStorage / Cookies | High | ⚠️ Manual Sync | ✅ Yes | ✅ Yes |
| Centralized OAuth / OIDC | High | ✅ Yes | ✅ Yes | ✅ Yes |

**Best Practices**

1. Single Source of Truth: Only one service should own token and user session logic.
2. Context-Agnostic Tokens: Use cookies or secure storage accessible to all MFEs under the same domain.
3. Replay Authentication Events: New MFEs should call AuthService.getUser() on mount to catch up on prior auth state.
4. Cross-Origin Safety: For MFEs running on different domains, use a secure backend SSO redirect instead of relying on localStorage.
5. Federate Shared Auth Service – use Module Federation to prevent duplicate instances of AuthService.

**Shared AuthService (Federation)**

Shell/package.json

{

  "name": "shell",

  "version": "1.0.0",

  "private": true,

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "history": "^5.3.0",

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.22.15",

    "@babel/preset-env": "^7.22.20",

    "@babel/preset-react": "^7.22.15",

    "babel-loader": "^9.2.1",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

**Shell/webpack.config.js**

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

module.exports = {

    mode: "development",

    devServer: {

        port: 3000,

        historyApiFallback: true,

        headers: {   // ✅ must be defined here

            "Access-Control-Allow-Origin": "\*",

            "Access-Control-Allow-Methods": "GET, POST, PUT, DELETE, PATCH, OPTIONS",

            "Access-Control-Allow-Headers": "X-Requested-With, Content-Type, Authorization"

        },

    },

    entry: "./src/index.js",

    devtool: "cheap-module-source-map",

    output: {

        publicPath: "auto",

        crossOriginLoading: "anonymous",  // 👈 enables proper script sharing between origins

    },

    module: {

        rules: [{ test: /\.(js|jsx)$/, exclude: /node\_modules/, use: "babel-loader" }],

    },

    resolve: { extensions: [".js", ".jsx"] },

    plugins: [

        new ModuleFederationPlugin({

            name: "shell",

            filename: "remoteEntry.js",

            exposes: {

                "./AuthStore": "./src/authStore",

                "./Navigation": "./src/navigation"  // 👈 expose navigation API

            },

            remotes: {

                app1: "app1@http://localhost:3001/remoteEntry.js",

                app2: "app2@http://localhost:3002/remoteEntry.js",

            },

            shared: { react: { singleton: true }, "react-dom": { singleton: true } },

        }),

        new HtmlWebpackPlugin({ template: "./public/index.html" }),

    ],

};

**Shell/.babelrc**

{ "presets": ["@babel/preset-env", "@babel/preset-react"] }

Shell/src/navigation.js

export const navigate = (path) => {

  window.history.pushState({}, "", path);

  // Dispatch a popstate event so React Router re‑syncs

  window.dispatchEvent(new PopStateEvent("popstate"));

};

**Shell/src/authStore.js**

// Singleton AuthStore shared across MFEs

if (!window.\_\_authStore\_\_) {

  window.\_\_authStore\_\_ = { user: null, token: null, listeners: [] };

}

export const AuthStore = {

  login(user, token) {

    window.\_\_authStore\_\_.user = user;

    window.\_\_authStore\_\_.token = token;

    localStorage.setItem("auth\_token", token);

    this.\_notify();

  },

  logout() {

    window.\_\_authStore\_\_.user = null;

    window.\_\_authStore\_\_.token = null;

    localStorage.removeItem("auth\_token");

    this.\_notify();

  },

  getUser() {

    return window.\_\_authStore\_\_.user;

  },

  isAuthenticated() {

    return !!window.\_\_authStore\_\_.token;

  },

  subscribe(cb) {

    window.\_\_authStore\_\_.listeners.push(cb);

    return () => {

      window.\_\_authStore\_\_.listeners =

        window.\_\_authStore\_\_.listeners.filter((fn) => fn !== cb);

    };

  },

  \_notify() {

    window.\_\_authStore\_\_.listeners.forEach((cb) =>

      cb(window.\_\_authStore\_\_.user)

    );

  },

};

Shell/src/bootstrap.js

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

const root = createRoot(document.getElementById("root"));

root.render(<App />);

**Shell/src/index.js**

import("./bootstrap");

**Shell/src/App.js**

import { BrowserRouter, Routes, Route, Link } from "react-router-dom";

import React, { Suspense } from "react";

const App1 = React.lazy(() => import("app1/App"));

const App2 = React.lazy(() => import("app2/App"));

export default function Shell() {

  return (

    <BrowserRouter>    {/\* ✅ Use BrowserRouter \*/}

      <nav style={{ padding: 10, borderBottom: "1px solid #ccc" }}>

        <Link to="/app1" style={{ marginRight: 10 }}>Login</Link>

        <Link to="/app2">Dashboard</Link>

      </nav>

      <Suspense fallback={<div>Loading...</div>}>

        <Routes>

          <Route path="/app1/\*" element={<App1 />} />

          <Route path="/app2/\*" element={<App2 />} />

        </Routes>

      </Suspense>

    </BrowserRouter>

  );

}

**Shell/public/index.html**

<!DOCTYPE html>

<html>

  <head><title>Shell App</title></head>

  <body><div id="root"></div></body>

</html>

App1

App1/package.json

{

  "name": "app1",

  "version": "1.0.0",

  "private": true,

  "scripts": {

    "start": "webpack serve --mode development --open"

  },

  "dependencies": {

    "history": "^5.3.0",

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "@babel/core": "^7.22.15",

    "@babel/preset-env": "^7.22.20",

    "@babel/preset-react": "^7.22.15",

    "babel-loader": "^9.2.1",

    "html-webpack-plugin": "^5.5.3",

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1"

  }

}

App1/webpack.config.js

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

module.exports = {

  mode: "development",

  devServer: {

    port: 3001,

    historyApiFallback: true,

    headers: {   // ✅ must be defined here

      "Access-Control-Allow-Origin": "\*",

      "Access-Control-Allow-Methods": "GET, POST, PUT, DELETE, PATCH, OPTIONS",

      "Access-Control-Allow-Headers": "X-Requested-With, Content-Type, Authorization"

    },

  },

  output: {

    publicPath: "auto",

    crossOriginLoading: "anonymous",  // 👈 enables proper script sharing between origins

  },

  devtool: "cheap-module-source-map",

  module: {

    rules: [{ test: /\.(js|jsx)$/, exclude: /node\_modules/, use: "babel-loader" }],

  },

  resolve: { extensions: [".js", ".jsx"] },

  plugins: [

    new ModuleFederationPlugin({

      name: "app1",

      filename: "remoteEntry.js",

      remotes: {

        shell: "shell@http://localhost:3000/remoteEntry.js",

      },

      exposes: { "./App": "./src/App" },

      shared: { react: { singleton: true }, "react-dom": { singleton: true } },

    }),

    new HtmlWebpackPlugin({ template: "./public/index.html" }),

  ],

};

**App1/.babelrc**

{ "presets": ["@babel/preset-env", "@babel/preset-react"] }

Shell/public/index.html

<!DOCTYPE html>

<html>

<head><title>Shell App</title></head>

<body><div id="root"></div></body>

</html>

**App1/src/bootstrap.js**

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

const root = createRoot(document.getElementById("root"));

root.render(<App />);

**App1/src/index.js**

import("./bootstrap");

**App1/src/App.js**

import React, { useState } from "react";

import { AuthStore } from "shell/AuthStore";

import { navigate } from "shell/Navigation"; // 👈 import from Shell

export default function App() {

  const [email, setEmail] = useState("");

  const handleLogin = () => {

    const user = { name: email };

    AuthStore.login(user, "FAKE\_TOKEN\_123");

    console.log("User logged in:", user);

    // 👇 Redirect user to App2 (Dashboard)

    navigate("/app2");

  };

  return (

    <div>

      <h2>App1 - Login</h2>

      <input

        type="email"

        placeholder="Enter your email"

        onChange={(e) => setEmail(e.target.value)}

      />

      <button onClick={handleLogin}>Login</button>

    </div>

  );

}

**App1/public/index.html**

<!DOCTYPE html>

<html>

<head><title>Shell App</title></head>

<body><div id="root"></div></body>

</html>

**App2**

App2/package.json

{

  "name": "app2",

  "version": "1.0.0",

  "private": true,

  "scripts": { "start": "webpack serve --mode development --open" },

  "dependencies": {

    "react": "^18.3.0",

    "react-dom": "^18.3.0",

    "react-router-dom": "^6.22.0"

  },

  "devDependencies": {

    "webpack": "^5.90.0",

    "webpack-cli": "^5.1.4",

    "webpack-dev-server": "^4.15.1",

    "html-webpack-plugin": "^5.5.3",

    "babel-loader": "^9.2.1",

    "@babel/core": "^7.22.15",

    "@babel/preset-env": "^7.22.20",

    "@babel/preset-react": "^7.22.15"

  }

}

App2/webpack.config.js

const HtmlWebpackPlugin = require("html-webpack-plugin");

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

module.exports = {

  mode: "development",

  devServer: {

    port: 3002,

    historyApiFallback: true,

    headers: {   // ✅ must be defined here

      "Access-Control-Allow-Origin": "\*",

      "Access-Control-Allow-Methods": "GET, POST, PUT, DELETE, PATCH, OPTIONS",

      "Access-Control-Allow-Headers": "X-Requested-With, Content-Type, Authorization"

    },

  },

  module: {

    rules: [{ test: /\.(js|jsx)$/, exclude: /node\_modules/, use: "babel-loader" }],

  },

  devtool: "cheap-module-source-map",

  output: {

    publicPath: "auto",

    crossOriginLoading: "anonymous",  // 👈 enables proper script sharing between origins

  },

  resolve: { extensions: [".js", ".jsx"] },

  plugins: [

    new ModuleFederationPlugin({

      name: "app2",

      filename: "remoteEntry.js",

      remotes: {

        shell: "shell@http://localhost:3000/remoteEntry.js",

      },

      exposes: { "./App": "./src/App" },

      shared: { react: { singleton: true }, "react-dom": { singleton: true } },

    }),

    new HtmlWebpackPlugin({ template: "./public/index.html" }),

  ],

};

**App2/.babelrc**

{ "presets": ["@babel/preset-env", "@babel/preset-react"] }

App2/public/index.html

<!DOCTYPE html>

<html>

<head><title>Shell App</title></head>

<body><div id="root"></div></body>

</html>

App2/src/bootstrap.js

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

const root = createRoot(document.getElementById("root"));

root.render(<App />);

App2/src/index.js

import("./bootstrap");

App2/src/App.js

import React, { useEffect, useState } from "react";

import { AuthStore } from "shell/AuthStore";

export default function App() {

  const [user, setUser] = useState(AuthStore.getUser());

  useEffect(() => {

    const unsubscribe = AuthStore.subscribe((data) => setUser(data));

    return unsubscribe;

  }, []);

  return (

    <div>

      <h2>App2 - Dashboard</h2>

      {user ? (

        <p>Welcome, {user.name}!</p>

      ) : (

        <p>Please login from App1.</p>

      )}

    </div>

  );

}

**Micro-Frontends with React and Vite**

**Micro-Frontends with React and Vite** using the official vite-plugin-federation plugin (from Webpack’s Module Federation adapted for Vite).

We’ll build **two apps**:

* **Host App (Container)** → Loads remote components
* **Remote App (Remote)** → Exposes a component

🧩 1. Setup Overview

You’ll create two folders:

microfrontend/

├── host/

└── remote/

Both will be separate React + Vite apps.

**⚙️ 2. Remote App**

**Create project**

npm create vite@latest remote --template react

cd remote

npm install @originjs/vite-plugin-federation

remote/vite.config.js

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

import federation from '@originjs/vite-plugin-federation'

export default defineConfig({

  plugins: [

    react(),

    federation({

      name: 'remote',

      filename: 'remoteEntry.js',

      exposes: {

        './HelloRemote': './src/HelloRemote.jsx',

      },

      shared: ['react', 'react-dom'],

    }),

  ],

  build: {

    target: 'esnext',

  },

  server: {

    port: 5001,

  },

})

Remote/package.json

{

  "name": "remote",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "vite build",

    "lint": "eslint .",

    "preview": "vite preview"

  },

  "dependencies": {

    "@originjs/vite-plugin-federation": "^1.4.1",

    "react": "^19.1.1",

    "react-dom": "^19.1.1"

  },

  "devDependencies": {

    "@eslint/js": "^9.36.0",

    "@types/react": "^19.1.16",

    "@types/react-dom": "^19.1.9",

    "@vitejs/plugin-react": "^5.0.4",

    "eslint": "^9.36.0",

    "eslint-plugin-react-hooks": "^5.2.0",

    "eslint-plugin-react-refresh": "^0.4.22",

    "globals": "^16.4.0",

    "vite": "^7.1.7"

  }

}

src/HelloRemote.jsx

export default function HelloRemote() {

  return <h2 style={{ color: 'green' }}>👋 Hello from Remote App!</h2>

}

**🏠 3. Host App**

**Create project**

cd ..

npm create vite@latest host --template react

cd host

npm install @originjs/vite-plugin-federation

host/vite.config.js

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

import federation from '@originjs/vite-plugin-federation'

export default defineConfig({

  plugins: [

    react(),

    federation({

      name: 'host',

      remotes: {

        remote: 'http://localhost:4173/assets/remoteEntry.js',

      },

      shared: ['react', 'react-dom'],

    }),

  ],

  build: {

    target: 'esnext',

  },

  server: {

    cors: true,

    port: 5000,

  },

})

Host/package.json

{

  "name": "host",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "vite build",

    "lint": "eslint .",

    "preview": "vite preview"

  },

  "dependencies": {

    "@originjs/vite-plugin-federation": "^1.4.1",

    "react": "^19.1.1",

    "react-dom": "^19.1.1"

  },

  "devDependencies": {

    "@eslint/js": "^9.36.0",

    "@types/react": "^19.1.16",

    "@types/react-dom": "^19.1.9",

    "@vitejs/plugin-react": "^5.0.4",

    "eslint": "^9.36.0",

    "eslint-plugin-react-hooks": "^5.2.0",

    "eslint-plugin-react-refresh": "^0.4.22",

    "globals": "^16.4.0",

    "vite": "^7.1.7"

  }

}

Src/App.jsx

import React, { Suspense } from 'react'

// Dynamically load remote component

const HelloRemote = React.lazy(() => import('remote/HelloRemote'))

function App() {

  return (

    <div style={{ padding: 20 }}>

      <h1>🏠 Host App</h1>

      <Suspense fallback={<p>Loading Remote...</p>}>

        <HelloRemote />

      </Suspense>

    </div>

  )

}

export default App

Testing:

Note:

Vite’s **dev server** does **not** automatically generate or serve remoteEntry.js during vite dev.  
The vite-plugin-federation plugin **only emits it during build time** (vite build).

So if you’re running the **remote** with npm run dev, the file remoteEntry.js simply doesn’t exist

Run Remote in Preview Mode

Run these commands in your **remote app** folder:

npm run build

npm run preview

host :

npm run dev

<http://localhost:5000/>

**Micro Frontend example using React Router + Vite Federation.**

We’ll use **two apps**:

* 🏠 **Host App** → Loads routes dynamically from a remote app
* 🌐 **Remote App** → Exposes its own routed component(s)

Using @originjs/vite-plugin-federation

**🧱 Folder Structure**

microfrontends/

├── host/

└── remote/

**🌐 1️⃣ Remote App (Exposes routes)**

**Create project**

npm create vite@latest remote --template react

cd remote

npm install react-router-dom @originjs/vite-plugin-federation

***remote/package.json***

{

  "name": "remote",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "vite build",

    "lint": "eslint .",

    "preview": "vite preview"

  },

  "dependencies": {

    "@originjs/vite-plugin-federation": "^1.4.1",

    "react": "^19.1.1",

    "react-dom": "^19.1.1",

    "react-router-dom": "^7.9.4"

  },

  "devDependencies": {

    "@eslint/js": "^9.36.0",

    "@types/react": "^19.1.16",

    "@types/react-dom": "^19.1.9",

    "@vitejs/plugin-react": "^5.0.4",

    "eslint": "^9.36.0",

    "eslint-plugin-react-hooks": "^5.2.0",

    "eslint-plugin-react-refresh": "^0.4.22",

    "globals": "^16.4.0",

    "vite": "^7.1.7"

  }

}

***Remote/vite.config.js***

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

import federation from '@originjs/vite-plugin-federation'

export default defineConfig({

  preview: {

    port: 8080, // Set your desired production preview port here

  },

  plugins: [

    react(),

    federation({

      name: 'remote',

      filename: 'remoteEntry.js',

      exposes: {

        './RemoteRoutes': './src/RemoteRoutes.jsx', // Expose routes

      },

      shared: ['react', 'react-dom'],

      dev: true

    }),

  ],

  build: {

    target: 'esnext',

    modulePreload: false,

  },

  server: {

    port: 5001,

    cors: true,

  },

})

**src/RemoteRoutes.jsx**

import React from "react";

import {

  Routes,

  Route,

  Link,

  MemoryRouter,

  useInRouterContext,

} from "react-router-dom";

const RemoteHome = () => <h2>🏡 Remote Home Page</h2>;

const RemoteAbout = () => <h2>ℹ️ Remote About Page</h2>;

function RemoteRoutesInner() {

  return (

    <div style={{ padding: 20 }}>

      <h3>🌐 Remote Navigation</h3>

      <nav>

        <Link to="/remote">Remote Home</Link> |{" "}

        <Link to="/remote/about">Remote About</Link>

      </nav>

      <Routes>

        <Route path="/remote" element={<RemoteHome />} />

        <Route path="/remote/about" element={<RemoteAbout />} />

      </Routes>

    </div>

  );

}

export default function RemoteRoutes() {

  const insideRouter = useInRouterContext();

  // ✅ If rendered inside Host Router, use context.

  // ✅ If standalone, wrap with MemoryRouter.

  return insideRouter ? (

    <RemoteRoutesInner />

  ) : (

    <MemoryRouter>

      <RemoteRoutesInner />

    </MemoryRouter>

  );

}

🏠 2️⃣ Host App (Consumes remote routes)

**Create project**

cd ..

npm create vite@latest host --template react

cd host

npm install react-router-dom @originjs/vite-plugin-federation

**host/package.json**

{

  "name": "host",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "vite build",

    "lint": "eslint .",

    "preview": "vite preview"

  },

  "dependencies": {

    "@originjs/vite-plugin-federation": "^1.4.1",

    "react": "^19.1.1",

    "react-dom": "^19.1.1",

    "react-router-dom": "^7.9.4"

  },

  "devDependencies": {

    "@eslint/js": "^9.36.0",

    "@types/react": "^19.1.16",

    "@types/react-dom": "^19.1.9",

    "@vitejs/plugin-react": "^5.0.4",

    "eslint": "^9.36.0",

    "eslint-plugin-react-hooks": "^5.2.0",

    "eslint-plugin-react-refresh": "^0.4.22",

    "globals": "^16.4.0",

    "vite": "^7.1.7"

  }

}

**Host/vite.config.js**

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

import federation from '@originjs/vite-plugin-federation'

export default defineConfig({

  plugins: [

    react(),

    federation({

      name: 'host',

      remotes: {

        remote: 'http://localhost:8080/assets/remoteEntry.js',

      },

      shared: ['react', 'react-dom', 'react-router-dom'],

    }),

  ],

  server: {

    port: 5000,

  },

  build: {

    target: 'esnext',

    modulePreload: false,

  },

})

Host/src/App.jsx

import React, { Suspense } from "react";

import { BrowserRouter, Routes, Route, Link } from "react-router-dom";

const RemoteRoutes = React.lazy(() => import("remote/RemoteRoutes"));

const HostHome = () => <h2>🏠 Host Home Page</h2>;

const HostAbout = () => <h2>ℹ️ Host About Page</h2>;

export default function App() {

  return (

    <BrowserRouter>

      <div style={{ padding: 20 }}>

        <h1>Microfrontend with React Router</h1>

        <nav>

          <Link to="/">Host Home</Link> | <Link to="/about">Host About</Link> |{" "}

          <Link to="/remote">Remote Pages</Link>

        </nav>

        <hr />

        <Suspense fallback={<p>Loading Remote...</p>}>

          <Routes>

            <Route path="/" element={<HostHome />} />

            <Route path="/about" element={<HostAbout />} />

            <Route path="/remote/\*" element={<RemoteRoutes />} />

          </Routes>

        </Suspense>

      </div>

    </BrowserRouter>

  );

}

**Testing:**

**npm run build**

**npm run preview**

**host :**

**npm run dev**

[**http://localhost:5000/**](http://localhost:5000/)

**State Management**

Each microfrontend is an **independent bundle**, so:

* Each has its **own React runtime** (unless shared properly)
* Local React state (useState, useReducer) is isolated per MFE
* Cross-MFE communication requires a **shared store or event bus**

Each microfrontend is an **independent bundle**, so:

* Each has its **own React runtime** (unless shared properly)
* Local React state (useState, useReducer) is isolated per MFE
* Cross-MFE communication requires a **shared store or event bus**

So we decide:

|  |  |
| --- | --- |
| Case | Recommended Approach |
| Each app manages its own state | Use React Context / Redux / Zustand independently |
| Need shared global state (e.g., user info, theme, auth) | Expose a shared store via Module Federation |
| Cross-MFE communication only (no tight coupling) | Use custom event bus (window.dispatchEvent) or RxJS |

**Shared Redux Store Across Host + Remote**

We’ll use **Redux Toolkit** and share the store instance across apps using **vite-plugin-federation**

**Remote App (will read/write shared store)**

**Install dependencies**

cd remote

npm install @reduxjs/toolkit react-redux

src/store.js

import { configureStore, createSlice } from '@reduxjs/toolkit'

const counterSlice = createSlice({

name: 'counter',

initialState: { value: 0 },

reducers: {

increment: (state) => { state.value += 1 },

decrement: (state) => { state.value -= 1 },

},

})

export const { increment, decrement } = counterSlice.actions

export const store = configureStore({

reducer: { counter: counterSlice.reducer },

})

Remote/vite.config.js

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

import federation from '@originjs/vite-plugin-federation'

export default defineConfig({

  preview: {

    port: 8080, // Set your desired production preview port here

  },

  plugins: [

    react(),

    federation({

      name: 'remote',

      filename: 'remoteEntry.js',

      exposes: {

        './RemoteRoutes': './src/RemoteRoutes.jsx', // Expose routes

        './RemoteCounter': './src/RemoteCounter.jsx',

        './store': './src/store.js',

      },

      shared: {

        react: {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

        'react-dom': {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

        'react-redux': {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

        '@reduxjs/toolkit': {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

      },

      dev: true

    }),

  ],

  build: {

    target: 'esnext',

    modulePreload: false,

  },

  server: {

    port: 5001,

    cors: true,

  },

})

src/RemoteCounter.jsx

import React from 'react'

import { useSelector, useDispatch } from 'react-redux'

import { increment, decrement } from './store'

export default function RemoteCounter() {

const value = useSelector((state) => state.counter.value)

const dispatch = useDispatch()

return (

<div style={{ padding: 20, border: '1px solid gray', marginTop: 20 }}>

<h3>🌐 Remote Counter</h3>

<p>Value: {value}</p>

<button onClick={() => dispatch(increment())}>+</button>

<button onClick={() => dispatch(decrement())}>-</button>

</div>

)

}

**🔹 Host App (will import and use same store)**

**Install same dependencies**

cd ../host

npm install @reduxjs/toolkit react-redux

host/ vite.config.js

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

import federation from '@originjs/vite-plugin-federation'

export default defineConfig({

  plugins: [

    react(),

    federation({

      name: 'host',

      remotes: {

        remote: 'http://localhost:8080/assets/remoteEntry.js',

      },

      shared: {

        react: {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

        'react-dom': {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

        'react-redux': {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

        '@reduxjs/toolkit': {

          singleton: true,

          eager: true,

          requiredVersion: false,

        },

      },

    }),

  ],

  server: {

    port: 5000,

  },

  build: {

    target: 'esnext',

    modulePreload: false,

  },

})

src/App.jsx

import React, { Suspense } from 'react'

import { Provider } from 'react-redux'

import { store } from 'remote/store' // shared store imported

import { BrowserRouter, Routes, Route, Link } from 'react-router-dom'

const RemoteCounter = React.lazy(() => import('remote/RemoteCounter'))

export default function App() {

return (

<Provider store={store}>

<BrowserRouter>

<div style={{ padding: 20 }}>

<h1>🏠 Host + Remote Shared State Example</h1>

<nav>

<Link to="/">Home</Link> | <Link to="/remote">Remote Counter</Link>

</nav>

<Routes>

<Route path="/" element={<p>Host Home Page</p>} />

<Route

path="/remote"

element={

<Suspense fallback={<p>Loading Remote...</p>}>

<RemoteCounter />

</Suspense>

}

/>

</Routes>

</div>

</BrowserRouter>

</Provider>

)

}

**Testing:**

**npm run build**

**npm run preview**

**host :**

**npm run dev**

[**http://localhost:5000/**](http://localhost:5000/)