Creating a Micro UI using React involves several steps. Here's a simplified example of setting up a basic Micro UI architecture with two independent React applications that communicate with each other. We'll use Webpack 5 with Module Federation to demonstrate this.

**Step 1: Set Up the Host Application**

1. **Initialize the Host Application:**

npx create-react-app host-app

cd host-app

Install Dependencies:

npm install webpack webpack-cli html-webpack-plugin webpack-dev-server @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev

1. **Configure Webpack:**

Create webpack.config.js in the root of the host-app directory:

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

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

module.exports = {

mode: 'development',

devServer: {

port: 3000,

},

module: {

rules: [

{

test: /\.jsx?$/,

loader: 'babel-loader',

options: {

presets: ['@babel/preset-react'],

},

},

],

},

plugins: [

new ModuleFederationPlugin({

name: 'host',

remotes: {

remoteApp: 'remoteApp@http://localhost:3001/remoteEntry.js',

},

}),

new HtmlWebpackPlugin({

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

}),

],

};

Update index.js:

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

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

Create App.js:

import React, { Suspense } from 'react';

const RemoteButton = React.lazy(() => import('remoteApp/Button'));

const App = () => (

<div>

<h1>Host Application</h1>

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

<RemoteButton />

</Suspense>

</div>

);

export default App;

**Step 2: Set Up the Remote Application**

1. **Initialize the Remote Application:**

npx create-react-app remote-app

cd remote-app

Install Dependencies:

npm install webpack webpack-cli html-webpack-plugin webpack-dev-server @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev

1. **Configure Webpack:**

Create webpack.config.js in the root of the remote-app directory:

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

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

module.exports = {

mode: 'development',

devServer: {

port: 3001,

},

module: {

rules: [

{

test: /\.jsx?$/,

loader: 'babel-loader',

options: {

presets: ['@babel/preset-react'],

},

},

],

},

plugins: [

new ModuleFederationPlugin({

name: 'remoteApp',

filename: 'remoteEntry.js',

exposes: {

'./Button': './src/Button',

},

}),

new HtmlWebpackPlugin({

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

}),

],

};

1. **Create a Component to Expose:**

Create src/Button.js:

import React from 'react';

const Button = () => <button>Remote Button</button>;

export default Button;

Update index.js:

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

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

Create App.js:

import React from 'react';

const App = () => (

<div>

<h1>Remote Application</h1>

<p>This is a remote application exposing a button component.</p>

</div>

);

export default App;

**Step 3: Run the Applications**

1. **Run the Host Application:**

cd host-app

npm start

Run the Remote Application:

cd remote-app

npm start

**Summary**

* **Host Application** (host-app): This application imports and uses the Button component from the remote application.
* **Remote Application** (remote-app): This application exposes a Button component via Module Federation.

This setup demonstrates how you can create a Micro UI architecture with independent deployment and development using React and Webpack 5's Module Federation. You can expand this example by adding more components and configuring more advanced communication and state management between the micro frontends.