Certainly! Let's build a \*\*microfrontend\*\* application using \*\*React\*\*, \*\*TypeScript\*\*, and \*\*TailwindCSS\*\* with the \*\*Wells Fargo\*\* color scheme. We'll create two separate applications:

1. \*\*Shell App\*\*: Contains the header and footer.

2. \*\*UI App\*\*: Contains the login form (username, password, and login button).

We'll utilize \*\*Webpack 5's Module Federation\*\* to integrate these microfrontends without using CRACO or ejecting from Create React App (CRA). Instead, we'll set up both applications from scratch with custom Webpack configurations to keep things simple and maintain control over the build process.

## Table of Contents

1. [Prerequisites](#1-prerequisites)

2. [Project Structure Overview](#2-project-structure-overview)

3. [Setting Up the Shell App](#3-setting-up-the-shell-app)

4. [Setting Up the UI App](#4-setting-up-the-ui-app)

5. [Running the Applications](#5-running-the-applications)

6. [Conclusion](#6-conclusion)

---

## 1. Prerequisites

Before we begin, ensure you have the following installed on your machine:

- \*\*Node.js\*\* (Latest LTS version recommended)

[Download Node.js](https://nodejs.org/en/download/)

- \*\*npm\*\* (Comes with Node.js)

Verify the installation:

```bash

node -v

npm -v

```

- \*\*Git\*\* (Optional, for version control)

[Download Git](https://git-scm.com/downloads)

- \*\*Code Editor\*\*: VSCode is recommended.

[Download VSCode](https://code.visualstudio.com/)

---

## 2. Project Structure Overview

We'll create two separate projects in a single workspace for simplicity:

```

microfrontend-app/

├── shell-app/

│ ├── public/

│ │ └── index.html

│ ├── src/

│ │ ├── components/

│ │ │ ├── Footer.tsx

│ │ │ └── Header.tsx

│ │ ├── App.tsx

│ │ ├── index.tsx

│ │ └── styles/

│ │ └── index.css

│ ├── package.json

│ ├── tsconfig.json

│ ├── webpack.config.js

│ ├── tailwind.config.js

│ └── postcss.config.js

└── ui-app/

├── public/

│ └── index.html

├── src/

│ ├── components/

│ │ └── Login.tsx

│ ├── App.tsx

│ ├── index.tsx

│ └── styles/

│ └── index.css

├── package.json

├── tsconfig.json

├── webpack.config.js

├── tailwind.config.js

└── postcss.config.js

```

---

## 3. Setting Up the Shell App

The \*\*Shell App\*\* serves as the host application containing the header and footer. It dynamically loads the \*\*UI App\*\* (login form) as a remote module.

### 3.1. Initialize the Shell App

1. \*\*Create the Project Directory:\*\*

```bash

mkdir microfrontend-app

cd microfrontend-app

mkdir shell-app

cd shell-app

```

2. \*\*Initialize `package.json`:\*\*

```bash

npm init -y

```

3. \*\*Install Dependencies:\*\*

```bash

npm install react react-dom

npm install --save-dev typescript @types/react @types/react-dom webpack webpack-cli webpack-dev-server html-webpack-plugin ts-loader css-loader style-loader postcss postcss-loader tailwindcss autoprefixer @types/node

```

4. \*\*Initialize TypeScript:\*\*

```bash

npx tsc --init

```

Modify `tsconfig.json` as follows:

```json

{

"compilerOptions": {

"target": "ES6",

"module": "ESNext",

"lib": ["DOM", "DOM.Iterable", "ESNext"],

"allowJs": true,

"skipLibCheck": true,

"esModuleInterop": true,

"allowSyntheticDefaultImports": true,

"strict": true,

"forceConsistentCasingInFileNames": true,

"moduleResolution": "node",

"resolveJsonModule": true,

"isolatedModules": true,

"noEmit": true,

"jsx": "react-jsx"

},

"include": ["src"]

}

```

5. \*\*Set Up TailwindCSS:\*\*

```bash

npx tailwindcss init -p

```

- \*\*`tailwind.config.js`:\*\*

```javascript

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: [

"./src/\*\*/\*.{js,jsx,ts,tsx}",

"./public/index.html",

],

theme: {

extend: {

colors: {

'wells-fargo-red': '#CC0000',

'wells-fargo-gray': '#666666',

// Add more colors as needed

},

},

},

plugins: [],

}

```

- \*\*`postcss.config.js`:\*\*

```javascript

module.exports = {

plugins: {

tailwindcss: {},

autoprefixer: {},

},

}

```

6. \*\*Create Project Files:\*\*

- \*\*`public/index.html`:\*\*

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Shell App</title>

</head>

<body>

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

</body>

</html>

```

- \*\*`src/styles/index.css`:\*\*

```css

@tailwind base;

@tailwind components;

@tailwind utilities;

```

- \*\*`src/components/Header.tsx`:\*\*

```tsx

import React from 'react';

const Header: React.FC = () => {

return (

<header className="bg-wells-fargo-red text-white p-4">

<h1 className="text-xl">Shell App Header</h1>

</header>

);

};

export default Header;

```

- \*\*`src/components/Footer.tsx`:\*\*

```tsx

import React from 'react';

const Footer: React.FC = () => {

return (

<footer className="bg-wells-fargo-gray text-white p-4 mt-auto">

<p className="text-center">© 2024 Shell App Footer</p>

</footer>

);

};

export default Footer;

```

- \*\*`src/App.tsx`:\*\*

```tsx

import React, { Suspense } from 'react';

import Header from './components/Header';

import Footer from './components/Footer';

const RemoteLogin = React.lazy(() => import('ui/Login'));

const App: React.FC = () => {

return (

<div className="flex flex-col min-h-screen">

<Header />

<main className="flex-grow p-4">

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

<RemoteLogin />

</Suspense>

</main>

<Footer />

</div>

);

};

export default App;

```

- \*\*`src/index.tsx`:\*\*

```tsx

import React from 'react';

import ReactDOM from 'react-dom';

import './styles/index.css';

import App from './App';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

```

7. \*\*Configure Webpack:\*\*

- \*\*`webpack.config.js`:\*\*

```javascript

const path = require('path');

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

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

module.exports = {

entry: './src/index.tsx',

mode: 'development',

devServer: {

static: {

directory: path.join(\_\_dirname, 'public'),

},

port: 3000,

historyApiFallback: true,

},

output: {

publicPath: 'auto',

},

resolve: {

extensions: ['.tsx', '.ts', '.jsx', '.js', '.json'],

},

module: {

rules: [

{

test: /\.(ts|tsx)$/,

loader: 'ts-loader',

exclude: /node\_modules/,

},

{

test: /\.css$/i,

use: ['style-loader', 'css-loader', 'postcss-loader'],

},

],

},

plugins: [

new ModuleFederationPlugin({

name: 'shell',

remotes: {

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

},

shared: {

react: {

singleton: true,

requiredVersion: '^18.0.0',

},

'react-dom': {

singleton: true,

requiredVersion: '^18.0.0',

},

},

}),

new HtmlWebpackPlugin({

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

}),

],

};

```

8. \*\*Update `package.json` Scripts:\*\*

Add the following scripts to your `package.json`:

```json

"scripts": {

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

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

}

```

### 3.2. Shell App `package.json`

Here's the complete `package.json` for the Shell App:

```json

{

"name": "shell-app",

"version": "1.0.0",

"description": "Shell application containing header and footer",

"main": "src/index.tsx",

"scripts": {

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

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

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"@types/node": "^20.4.1",

"@types/react": "^18.2.14",

"@types/react-dom": "^18.2.6",

"autoprefixer": "^10.4.14",

"css-loader": "^6.8.1",

"html-webpack-plugin": "^5.5.3",

"postcss": "^8.4.28",

"postcss-loader": "^7.3.3",

"style-loader": "^3.3.3",

"tailwindcss": "^3.3.3",

"ts-loader": "^9.4.4",

"typescript": "^5.1.6",

"webpack": "^5.88.2",

"webpack-cli": "^5.1.4",

"webpack-dev-server": "^4.15.1"

}

}

```

---

## 4. Setting Up the UI App

The \*\*UI App\*\* is a separate application that exposes the `Login` component, which will be consumed by the Shell App.

### 4.1. Initialize the UI App

1. \*\*Navigate Back to the Workspace:\*\*

```bash

cd ..

mkdir ui-app

cd ui-app

```

2. \*\*Initialize `package.json`:\*\*

```bash

npm init -y

```

3. \*\*Install Dependencies:\*\*

```bash

npm install react react-dom

npm install --save-dev typescript @types/react @types/react-dom webpack webpack-cli webpack-dev-server html-webpack-plugin ts-loader css-loader style-loader postcss postcss-loader tailwindcss autoprefixer @types/node

```

4. \*\*Initialize TypeScript:\*\*

```bash

npx tsc --init

```

Modify `tsconfig.json` as follows:

```json

{

"compilerOptions": {

"target": "ES6",

"module": "ESNext",

"lib": ["DOM", "DOM.Iterable", "ESNext"],

"allowJs": true,

"skipLibCheck": true,

"esModuleInterop": true,

"allowSyntheticDefaultImports": true,

"strict": true,

"forceConsistentCasingInFileNames": true,

"moduleResolution": "node",

"resolveJsonModule": true,

"isolatedModules": true,

"noEmit": true,

"jsx": "react-jsx"

},

"include": ["src"]

}

```

5. \*\*Set Up TailwindCSS:\*\*

```bash

npx tailwindcss init -p

```

- \*\*`tailwind.config.js`:\*\*

```javascript

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: [

"./src/\*\*/\*.{js,jsx,ts,tsx}",

"./public/index.html",

],

theme: {

extend: {

colors: {

'wells-fargo-red': '#CC0000',

'wells-fargo-gray': '#666666',

// Add more colors as needed

},

},

},

plugins: [],

}

```

- \*\*`postcss.config.js`:\*\*

```javascript

module.exports = {

plugins: {

tailwindcss: {},

autoprefixer: {},

},

}

```

6. \*\*Create Project Files:\*\*

- \*\*`public/index.html`:\*\*

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>UI App</title>

</head>

<body>

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

</body>

</html>

```

- \*\*`src/styles/index.css`:\*\*

```css

@tailwind base;

@tailwind components;

@tailwind utilities;

```

- \*\*`src/components/Login.tsx`:\*\*

```tsx

import React, { useState } from 'react';

const Login: React.FC = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const handleLogin = () => {

// Handle login logic

alert(`Username: ${username}\nPassword: ${password}`);

};

return (

<div className="max-w-md mx-auto mt-10 p-6 bg-white rounded shadow-md">

<h2 className="text-2xl mb-4 text-wells-fargo-red">Login</h2>

<div className="mb-4">

<label className="block text-gray-700">Username</label>

<input

type="text"

value={username}

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

className="w-full px-3 py-2 border rounded"

placeholder="Enter username"

/>

</div>

<div className="mb-4">

<label className="block text-gray-700">Password</label>

<input

type="password"

value={password}

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

className="w-full px-3 py-2 border rounded"

placeholder="Enter password"

/>

</div>

<button

onClick={handleLogin}

className="w-full bg-wells-fargo-red text-white py-2 rounded hover:bg-red-700"

>

Login

</button>

</div>

);

};

export default Login;

```

- \*\*`src/App.tsx`:\*\*

```tsx

import React from 'react';

import './styles/index.css';

const App: React.FC = () => {

return (

<div className="App">

<h1 className="text-3xl font-bold underline">

UI App is running.

</h1>

</div>

);

};

export default App;

```

- \*\*`src/index.tsx`:\*\*

```tsx

import React from 'react';

import ReactDOM from 'react-dom';

import './styles/index.css';

import App from './App';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

```

7. \*\*Configure Webpack:\*\*

- \*\*`webpack.config.js`:\*\*

```javascript

const path = require('path');

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

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

module.exports = {

entry: './src/index.tsx',

mode: 'development',

devServer: {

static: {

directory: path.join(\_\_dirname, 'public'),

},

port: 3001,

historyApiFallback: true,

},

output: {

publicPath: 'auto',

},

resolve: {

extensions: ['.tsx', '.ts', '.jsx', '.js', '.json'],

},

module: {

rules: [

{

test: /\.(ts|tsx)$/,

loader: 'ts-loader',

exclude: /node\_modules/,

},

{

test: /\.css$/i,

use: ['style-loader', 'css-loader', 'postcss-loader'],

},

],

},

plugins: [

new ModuleFederationPlugin({

name: 'ui',

filename: 'remoteEntry.js',

exposes: {

'./Login': './src/components/Login',

},

shared: {

react: {

singleton: true,

requiredVersion: '^18.0.0',

},

'react-dom': {

singleton: true,

requiredVersion: '^18.0.0',

},

},

}),

new HtmlWebpackPlugin({

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

}),

],

};

```

8. \*\*Update `package.json` Scripts:\*\*

Add the following scripts to your `package.json`:

```json

"scripts": {

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

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

}

```

### 4.2. UI App `package.json`

Here's the complete `package.json` for the UI App:

```json

{

"name": "ui-app",

"version": "1.0.0",

"description": "UI application exposing the Login component",

"main": "src/index.tsx",

"scripts": {

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

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

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"@types/node": "^20.4.1",

"@types/react": "^18.2.14",

"@types/react-dom": "^18.2.6",

"autoprefixer": "^10.4.14",

"css-loader": "^6.8.1",

"html-webpack-plugin": "^5.5.3",

"postcss": "^8.4.28",

"postcss-loader": "^7.3.3",

"style-loader": "^3.3.3",

"tailwindcss": "^3.3.3",

"ts-loader": "^9.4.4",

"typescript": "^5.1.6",

"webpack": "^5.88.2",

"webpack-cli": "^5.1.4",

"webpack-dev-server": "^4.15.1"

}

}

```

---

## 5. Running the Applications

Now that both the \*\*Shell App\*\* and \*\*UI App\*\* are set up, let's run them simultaneously.

### 5.1. Running the UI App

1. \*\*Navigate to the UI App Directory:\*\*

```bash

cd ../ui-app

```

2. \*\*Install Dependencies:\*\*

```bash

npm install

```

3. \*\*Start the UI App:\*\*

```bash

npm start

```

- The UI App will run on [http://localhost:3001](http://localhost:3001).

- It exposes the `Login` component via Module Federation at `http://localhost:3001/remoteEntry.js`.

### 5.2. Running the Shell App

1. \*\*Open a New Terminal Window or Tab.\*\*

2. \*\*Navigate to the Shell App Directory:\*\*

```bash

cd microfrontend-app/shell-app

```

3. \*\*Install Dependencies:\*\*

```bash

npm install

```

4. \*\*Start the Shell App:\*\*

```bash

npm start

```

- The Shell App will run on [http://localhost:3000](http://localhost:3000).

- It consumes the `Login` component from the UI App via Module Federation.

### 5.3. Accessing the Application

1. \*\*Open Your Browser and Navigate to:\*\*

[http://localhost:3000](http://localhost:3000)

2. \*\*You Should See:\*\*

- \*\*Header\*\*: "Shell App Header" with Wells Fargo red background.

- \*\*Main Content\*\*: The `Login` form loaded from the UI App.

- \*\*Footer\*\*: "© 2024 Shell App Footer" with Wells Fargo gray background.

![Shell App Screenshot](https://i.imgur.com/your-screenshot-link.png) \*(Replace with an actual screenshot if available)\*

---

## 6. Conclusion

You've successfully built a \*\*microfrontend\*\* architecture using \*\*React\*\*, \*\*TypeScript\*\*, and \*\*TailwindCSS\*\* with \*\*Webpack 5's Module Federation\*\*. The \*\*Shell App\*\* serves as the host containing the header and footer, while the \*\*UI App\*\* provides the login functionality. Both applications run independently on `localhost` and can be developed, tested, and deployed separately, enhancing scalability and maintainability.

### Additional Recommendations

1. \*\*Production Build:\*\*

To build both applications for production, run the build scripts:

- \*\*UI App:\*\*

```bash

cd microfrontend-app/ui-app

npm run build

```

- \*\*Shell App:\*\*

```bash

cd microfrontend-app/shell-app

npm run build

```

Ensure that your production server serves the built assets correctly and that the `publicPath` in the Webpack configuration points to the correct URLs.

2. \*\*Styling Consistency:\*\*

Ensure that both applications share the same TailwindCSS configuration to maintain consistent styling across microfrontends.

3. \*\*Shared Libraries:\*\*

Currently, both apps share `react` and `react-dom` as singletons. If you introduce more shared libraries (like state management tools), update the `shared` section in the Webpack configuration accordingly.

4. \*\*Routing:\*\*

For more complex applications, consider implementing routing within the Shell App and passing routing props to microfrontends as needed.

5. \*\*State Management:\*\*

If multiple microfrontends need to share state, explore state management solutions like Redux or Context API with proper scoping.

6. \*\*Authentication:\*\*

Implement secure authentication mechanisms, especially when handling login functionalities.

7. \*\*Error Handling:\*\*

Enhance error handling for better user experience, such as fallback UI for failed microfrontend loads.

8. \*\*Performance Optimization:\*\*

Optimize bundle sizes and leverage caching strategies to improve application performance.

Feel free to extend this setup by adding more microfrontends or enhancing existing functionalities as per your project requirements. Happy coding!