Understood! We'll build the microfrontend application entirely with Webpack 5 from the start, avoiding Vite. This approach ensures consistency and leverages Webpack's powerful Module Federation feature from the ground up.

We'll create two separate applications:

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

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

Finally, we'll integrate them using Webpack's Module Federation to run them together locally.

### Table of Contents

1. [Prerequisites](#prerequisites)

2. [Step 1: Create the Independent Shell App](#step-1-create-the-independent-shell-app)

- [1.1. Initialize the Shell App](#11-initialize-the-shell-app)

- [1.2. Install Dependencies](#12-install-dependencies)

- [1.3. Configure TypeScript](#13-configure-typescript)

- [1.4. Configure TailwindCSS](#14-configure-tailwindcss)

- [1.5. Create Header and Footer Components](#15-create-header-and-footer-components)

- [1.6. Configure Webpack for the Shell App](#16-configure-webpack-for-the-shell-app)

- [1.7. Run the Shell App](#17-run-the-shell-app)

3. [Step 2: Create the Independent UI App](#step-2-create-the-independent-ui-app)

- [2.1. Initialize the UI App](#21-initialize-the-ui-app)

- [2.2. Install Dependencies](#22-install-dependencies)

- [2.3. Configure TypeScript](#23-configure-typescript)

- [2.4. Configure TailwindCSS](#24-configure-tailwindcss)

- [2.5. Create the Login Form Component](#25-create-the-login-form-component)

- [2.6. Configure Webpack for the UI App](#26-configure-webpack-for-the-ui-app)

- [2.7. Run the UI App](#27-run-the-ui-app)

4. [Step 3: Add Module Federation and Integrate Both Apps](#step-3-add-module-federation-and-integrate-both-apps)

- [3.1. Configure Module Federation in the UI App](#31-configure-module-federation-in-the-ui-app)

- [3.2. Configure Module Federation in the Shell App](#32-configure-module-federation-in-the-shell-app)

- [3.3. Update Shell App to Consume Remote Module](#33-update-shell-app-to-consume-remote-module)

- [3.4. TypeScript Declarations for Module Federation](#34-typescript-declarations-for-module-federation)

- [3.5. Run Both Applications](#35-run-both-applications)

5. [Conclusion](#conclusion)

---

## Prerequisites

Ensure you have the following installed on your machine:

- \*\*Node.js\*\*: Latest LTS version (e.g., v18.x.x or higher)

- \*\*npm\*\*: Comes with Node.js

- \*\*Git\*\*: Optional, for version control

---

## Step 1: Create the Independent Shell App

The Shell App will contain only the header and footer, serving as the container for our microfrontends.

### 1.1. Initialize the Shell App

1. \*\*Create a new directory for the shell app:\*\*

```bash

mkdir shell-app

cd shell-app

```

2. \*\*Initialize a new npm project:\*\*

```bash

npm init -y

```

### 1.2. Install Dependencies

Install the necessary dependencies for React, TypeScript, TailwindCSS, and Webpack.

```bash

# Install React and ReactDOM

npm install react react-dom

# Install TypeScript and type definitions

npm install -D typescript @types/react @types/react-dom

# Install TailwindCSS and its dependencies

npm install -D tailwindcss postcss autoprefixer

# Initialize TailwindCSS

npx tailwindcss init -p

# Install Webpack and related dependencies

npm install -D webpack webpack-cli webpack-dev-server html-webpack-plugin webpack-merge ts-loader css-loader style-loader postcss-loader

```

### 1.3. Configure TypeScript

Create a `tsconfig.json` file in the root of `shell-app`:

```json

{

"compilerOptions": {

"target": "ES6",

"module": "ESNext",

"jsx": "react-jsx",

"strict": true,

"moduleResolution": "node",

"esModuleInterop": true,

"skipLibCheck": true,

"forceConsistentCasingInFileNames": true

},

"include": ["src"]

}

```

### 1.4. Configure TailwindCSS

1. \*\*Update `tailwind.config.js`:\*\*

```javascript

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

module.exports = {

content: [

"./public/index.html",

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

],

theme: {

extend: {

colors: {

// Wells Fargo color scheme

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

'wells-fargo-gray': '#F2F2F2',

'wells-fargo-dark-gray': '#333333',

},

},

},

plugins: [],

};

```

2. \*\*Update/Create `src/index.css` with Tailwind directives:\*\*

```css

@tailwind base;

@tailwind components;

@tailwind utilities;

```

### 1.5. Create Header and Footer Components

1. \*\*Create `src/components/Header.tsx`:\*\*

```tsx

// shell-app/src/components/Header.tsx

import React from 'react';

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

return (

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

<h1 className="text-2xl">Shell App Header</h1>

</header>

);

};

export default Header;

```

2. \*\*Create `src/components/Footer.tsx`:\*\*

```tsx

// shell-app/src/components/Footer.tsx

import React from 'react';

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

return (

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

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

</footer>

);

};

export default Footer;

```

3. \*\*Create `src/App.tsx`:\*\*

```tsx

// shell-app/src/App.tsx

import React from 'react';

import Header from './components/Header';

import Footer from './components/Footer';

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

return (

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

<Header />

<main className="flex-grow container mx-auto p-4">

<h2 className="text-xl">Welcome to the Shell App</h2>

</main>

<Footer />

</div>

);

};

export default App;

```

4. \*\*Create `src/index.tsx`:\*\*

```tsx

// shell-app/src/index.tsx

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import './index.css';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

```

5. \*\*Create `public/index.html`:\*\*

```html

<!-- shell-app/public/index.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>

```

### 1.6. Configure Webpack for the Shell App

1. \*\*Create `webpack.common.js`:\*\*

```javascript

// shell-app/webpack.common.js

const path = require('path');

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

module.exports = {

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

resolve: {

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

},

module: {

rules: [

{

test: /\.tsx?$/,

use: 'ts-loader',

exclude: /node\_modules/,

},

{

test: /\.css$/i,

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

},

],

},

plugins: [

new HtmlWebpackPlugin({

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

}),

],

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist'),

clean: true,

},

};

```

2. \*\*Create `webpack.dev.js`:\*\*

```javascript

// shell-app/webpack.dev.js

const { merge } = require('webpack-merge');

const common = require('./webpack.common.js');

module.exports = merge(common, {

mode: 'development',

devServer: {

static: './dist',

port: 3000,

historyApiFallback: true,

hot: true,

},

devtool: 'inline-source-map',

});

```

3. \*\*Create `webpack.prod.js`:\*\*

```javascript

// shell-app/webpack.prod.js

const { merge } = require('webpack-merge');

const common = require('./webpack.common.js');

module.exports = merge(common, {

mode: 'production',

devtool: 'source-map',

});

```

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

```json

// shell-app/package.json

{

"name": "shell-app",

"version": "1.0.0",

"private": true,

"scripts": {

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

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

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"@types/react": "^18.0.28",

"@types/react-dom": "^18.0.11",

"autoprefixer": "^10.4.14",

"css-loader": "^6.8.1",

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

"postcss": "^8.4.23",

"postcss-loader": "^7.3.3",

"react": "^18.2.0",

"react-dom": "^18.2.0",

"style-loader": "^3.3.3",

"tailwindcss": "^3.3.2",

"ts-loader": "^9.4.4",

"typescript": "^5.1.6",

"webpack": "^5.88.2",

"webpack-cli": "^5.1.4",

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

"webpack-merge": "^5.8.0"

}

}

```

### 1.7. Run the Shell App

1. \*\*Start the development server:\*\*

```bash

npm start

```

2. \*\*Access the Shell App:\*\*

Open your browser and navigate to [http://localhost:3000](http://localhost:3000). You should see the header, a welcome message, and the footer styled with the Wells Fargo color scheme.

---

## Step 2: Create the Independent UI App

The UI App will contain the login form and will be integrated into the Shell App using Module Federation.

### 2.1. Initialize the UI App

1. \*\*Navigate back to the parent directory:\*\*

```bash

cd ..

```

2. \*\*Create a new directory for the UI app:\*\*

```bash

mkdir ui-app

cd ui-app

```

3. \*\*Initialize a new npm project:\*\*

```bash

npm init -y

```

### 2.2. Install Dependencies

Install the necessary dependencies for React, TypeScript, TailwindCSS, and Webpack.

```bash

# Install React and ReactDOM

npm install react react-dom

# Install TypeScript and type definitions

npm install -D typescript @types/react @types/react-dom

# Install TailwindCSS and its dependencies

npm install -D tailwindcss postcss autoprefixer

# Initialize TailwindCSS

npx tailwindcss init -p

# Install Webpack and related dependencies

npm install -D webpack webpack-cli webpack-dev-server html-webpack-plugin webpack-merge ts-loader css-loader style-loader postcss-loader

```

### 2.3. Configure TypeScript

Create a `tsconfig.json` file in the root of `ui-app`:

```json

{

"compilerOptions": {

"target": "ES6",

"module": "ESNext",

"jsx": "react-jsx",

"strict": true,

"moduleResolution": "node",

"esModuleInterop": true,

"skipLibCheck": true,

"forceConsistentCasingInFileNames": true

},

"include": ["src"]

}

```

### 2.4. Configure TailwindCSS

1. \*\*Update `tailwind.config.js`:\*\*

```javascript

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

module.exports = {

content: [

"./public/index.html",

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

],

theme: {

extend: {

colors: {

// Wells Fargo color scheme

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

'wells-fargo-gray': '#F2F2F2',

'wells-fargo-dark-gray': '#333333',

},

},

},

plugins: [],

};

```

2. \*\*Update/Create `src/index.css` with Tailwind directives:\*\*

```css

@tailwind base;

@tailwind components;

@tailwind utilities;

```

### 2.5. Create the Login Form Component

1. \*\*Create `src/components/LoginForm.tsx`:\*\*

```tsx

// ui-app/src/components/LoginForm.tsx

import React, { useState } from 'react';

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

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

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

const handleLogin = () => {

// Implement login logic here

alert(`Logged in as ${username}`);

};

return (

<div className="max-w-md mx-auto bg-wells-fargo-gray p-6 rounded shadow-md">

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

<div className="mb-4">

<label className="block text-wells-fargo-dark-gray mb-2">Username</label>

<input

type="text"

value={username}

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

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

/>

</div>

<div className="mb-4">

<label className="block text-wells-fargo-dark-gray mb-2">Password</label>

<input

type="password"

value={password}

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

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

/>

</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 LoginForm;

```

2. \*\*Create `src/App.tsx`:\*\*

```tsx

// ui-app/src/App.tsx

import React from 'react';

import LoginForm from './components/LoginForm';

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

return (

<div className="flex items-center justify-center min-h-screen bg-wells-fargo-gray">

<LoginForm />

</div>

);

};

export default App;

```

3. \*\*Create `src/index.tsx`:\*\*

```tsx

// ui-app/src/index.tsx

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import './index.css';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

```

4. \*\*Create `public/index.html`:\*\*

```html

<!-- ui-app/public/index.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>

```

### 2.6. Configure Webpack for the UI App

1. \*\*Create `webpack.common.js`:\*\*

```javascript

// ui-app/webpack.common.js

const path = require('path');

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

module.exports = {

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

resolve: {

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

},

module: {

rules: [

{

test: /\.tsx?$/,

use: 'ts-loader',

exclude: /node\_modules/,

},

{

test: /\.css$/i,

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

},

],

},

plugins: [

new HtmlWebpackPlugin({

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

}),

],

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist'),

clean: true,

},

};

```

2. \*\*Create `webpack.dev.js`:\*\*

```javascript

// ui-app/webpack.dev.js

const { merge } = require('webpack-merge');

const common = require('./webpack.common.js');

module.exports = merge(common, {

mode: 'development',

devServer: {

static: './dist',

port: 3001,

historyApiFallback: true,

hot: true,

},

devtool: 'inline-source-map',

});

```

3. \*\*Create `webpack.prod.js`:\*\*

```javascript

// ui-app/webpack.prod.js

const { merge } = require('webpack-merge');

const common = require('./webpack.common.js');

module.exports = merge(common, {

mode: 'production',

devtool: 'source-map',

});

```

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

```json

// ui-app/package.json

{

"name": "ui-app",

"version": "1.0.0",

"private": true,

"scripts": {

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

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

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"@types/react": "^18.0.28",

"@types/react-dom": "^18.0.11",

"autoprefixer": "^10.4.14",

"css-loader": "^6.8.1",

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

"postcss": "^8.4.23",

"postcss-loader": "^7.3.3",

"react": "^18.2.0",

"react-dom": "^18.2.0",

"style-loader": "^3.3.3",

"tailwindcss": "^3.3.2",

"ts-loader": "^9.4.4",

"typescript": "^5.1.6",

"webpack": "^5.88.2",

"webpack-cli": "^5.1.4",

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

"webpack-merge": "^5.8.0"

}

}

```

### 2.7. Run the UI App

1. \*\*Start the development server:\*\*

```bash

npm start

```

2. \*\*Access the UI App:\*\*

Open your browser and navigate to [http://localhost:3001](http://localhost:3001). You should see the login form styled with the Wells Fargo color scheme.

---

## Step 3: Add Module Federation and Integrate Both Apps

Now, we'll configure Webpack's Module Federation to allow the Shell App to consume the LoginForm component from the UI App.

### 3.1. Configure Module Federation in the UI App

1. \*\*Install `@babel/plugin-syntax-dynamic-import` (Optional):\*\*

For dynamic imports, ensure support is present. Though Webpack 5 natively supports dynamic imports, this step is generally optional.

```bash

npm install -D @babel/plugin-syntax-dynamic-import

```

2. \*\*Update `webpack.common.js` in the UI App to expose the LoginForm:\*\*

```javascript

// ui-app/webpack.common.js

const path = require('path');

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

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

module.exports = {

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

resolve: {

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

},

module: {

rules: [

{

test: /\.tsx?$/,

use: 'ts-loader',

exclude: /node\_modules/,

},

{

test: /\.css$/i,

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

},

],

},

plugins: [

new ModuleFederationPlugin({

name: 'uiApp',

filename: 'remoteEntry.js',

exposes: {

'./LoginForm': './src/components/LoginForm',

},

shared: {

react: { singleton: true, requiredVersion: '^18.0.0' },

'react-dom': { singleton: true, requiredVersion: '^18.0.0' },

},

}),

new HtmlWebpackPlugin({

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

}),

],

output: {

publicPath: 'http://localhost:3001/',

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist'),

clean: true,

},

};

```

\*\*Explanation:\*\*

- \*\*ModuleFederationPlugin:\*\*

- \*\*name\*\*: Unique name for the remote module.

- \*\*filename\*\*: The name of the manifest file that will be loaded by the host.

- \*\*exposes\*\*: Specifies which modules/components are exposed to other applications.

- \*\*shared\*\*: Defines shared dependencies to avoid duplication.

3. \*\*Rebuild and Restart the UI App:\*\*

```bash

npm run build

npm start

```

4. \*\*Verify `remoteEntry.js`:\*\*

Ensure that `remoteEntry.js` is being served at [http://localhost:3001/remoteEntry.js](http://localhost:3001/remoteEntry.js).

### 3.2. Configure Module Federation in the Shell App

1. \*\*Update `webpack.common.js` in the Shell App to consume the UI App:\*\*

```javascript

// shell-app/webpack.common.js

const path = require('path');

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

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

module.exports = {

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

resolve: {

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

},

module: {

rules: [

{

test: /\.tsx?$/,

use: 'ts-loader',

exclude: /node\_modules/,

},

{

test: /\.css$/i,

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

},

],

},

plugins: [

new ModuleFederationPlugin({

name: 'shellApp',

remotes: {

uiApp: 'uiApp@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',

}),

],

output: {

publicPath: 'http://localhost:3000/',

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist'),

clean: true,

},

};

```

\*\*Explanation:\*\*

- \*\*remotes\*\*: Specifies the remote modules that the Shell App will consume.

- \*\*uiApp\*\*: The name of the remote module, followed by the URL to its `remoteEntry.js`.

2. \*\*Ensure that Webpack can handle dynamic imports.\*\* No additional configuration is necessary as Webpack 5 handles this by default.

### 3.3. Update Shell App to Consume Remote Module

1. \*\*Update `src/App.tsx` to dynamically load the LoginForm from the UI App:\*\*

```tsx

// shell-app/src/App.tsx

import React, { Suspense } from 'react';

import Header from './components/Header';

import Footer from './components/Footer';

// Dynamically import the LoginForm from the remote UI App

const RemoteLoginForm = React.lazy(() => import('uiApp/LoginForm'));

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

return (

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

<Header />

<main className="flex-grow container mx-auto p-4">

<h2 className="text-xl mb-4">Welcome to the Shell App</h2>

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

<RemoteLoginForm />

</Suspense>

</main>

<Footer />

</div>

);

};

export default App;

```

\*\*Explanation:\*\*

- \*\*React.lazy\*\*: Dynamically imports the `LoginForm` component from the remote `uiApp`.

- \*\*Suspense\*\*: Provides a fallback UI while the remote component is loading.

### 3.4. TypeScript Declarations for Module Federation

TypeScript needs to be aware of the modules exposed by the remote app.

1. \*\*Install `@types/webpack-env`:\*\*

```bash

npm install -D @types/webpack-env

```

2. \*\*Create `src/declarations.d.ts` in the Shell App:\*\*

```typescript

// shell-app/src/declarations.d.ts

declare module 'uiApp/LoginForm' {

const LoginForm: React.ComponentType;

export default LoginForm;

}

```

\*\*Explanation:\*\*

- \*\*declare module\*\*: Tells TypeScript about the shape of the module being imported.

- This prevents TypeScript errors when importing modules from the remote app.

### 3.5. Run Both Applications

1. \*\*Ensure Both Apps Are Running:\*\*

- \*\*UI App\*\*: [http://localhost:3001](http://localhost:3001)

- \*\*Shell App\*\*: [http://localhost:3000](http://localhost:3000)

2. \*\*Access the Shell App:\*\*

Open your browser and navigate to [http://localhost:3000](http://localhost:3000).

\*\*Expected Outcome:\*\*

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

- \*\*Main Content\*\*:

- \*\*Title\*\*: "Welcome to the Shell App"

- \*\*Login Form\*\*: Loaded from the UI App, styled with the Wells Fargo color scheme.

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

3. \*\*Test the Login Form:\*\*

- Enter a username and password.

- Click the "Login" button to see an alert confirming the login.

\*\*Troubleshooting Tips:\*\*

- \*\*CORS Issues\*\*: Since both apps are running on different ports, Module Federation handles the necessary CORS headers. If you encounter CORS errors, ensure that your Webpack Dev Server is correctly configured and that the `publicPath` is set appropriately.

- \*\*Network Errors\*\*: Verify that `remoteEntry.js` is accessible at [http://localhost:3001/remoteEntry.js](http://localhost:3001/remoteEntry.js).

- \*\*Module Not Found\*\*: Ensure that the `exposes` path in the UI App matches the path used in the Shell App's import statement.

---

## Conclusion

You've successfully built a microfrontend architecture with a Shell App and a UI App using React, TypeScript, TailwindCSS, and Webpack 5 with Module Federation. This setup allows for independent development and deployment of frontend modules, enhancing scalability and maintainability.

### Additional Tips

- \*\*Styling Consistency\*\*: Ensure both apps use the same TailwindCSS configuration to maintain a consistent look and feel.

- \*\*State Management\*\*: For shared state, consider using a state management library (e.g., Redux) or React's Context API.

- \*\*Error Handling\*\*: Implement proper error boundaries to gracefully handle failed remote module loads.

- \*\*Deployment\*\*: For production, host the remote modules on a CDN or a dedicated server to optimize load times.

- \*\*Version Management\*\*: Keep shared dependencies (like React) as singletons to prevent multiple instances and potential conflicts.

- \*\*Security\*\*: Validate and sanitize any data passed between microfrontends to maintain application security.

Feel free to extend this setup with more microfrontends, advanced routing, and other features as your application grows!