**Micro-Frontends Application Tutorial**

This tutorial will guide you through the process of building a micro-frontends application in three main sections: setting up the project with Webpack, implementing micro-frontends with React Ionic, and containerizing the application with Docker.

**Setting up the project with Webpack**

1. **Install Node.js and npm**

Ensure you have Node.js and npm installed on your system. You can download them from the official website: <https://nodejs.org>

1. **Create a new React Ionic project**

**2.1** Open your terminal or command prompt and run the following commands to install the Ionic CLI globally and create a new Ionic React project:

* **npm install -g @ionic/cli**
* **ionic start app1 blank --type=react**

**2.2** Install Webpack and webpack-cli:

* **npm install --save-dev webpack webpack-cli**

**2.3** Create a Webpack configuration file like the one in the Github repository:

* **webpack.config.js**

**2.4** Install Babel and necessary presets to transpile React and Ionic code:

* **npm install --save-dev @babel/core @babel/preset-env @babel/preset-react @babel/preset-typescript babel-loader**

**2.5** Start the development server by running:

* **cd app1**
* **ionic serve**

**Implementing Micro-frontends with React Ionic**

Now that we have the app1 application up and running, the next step is to create another application app2 that follows the same steps as above. The app1 application will use a button from the app2 application using micro-frontends. The main difference would be that app1 will use port 3001 and app2 will use port 3002. Both app1 and app2 will be added to the same basic-host-remote project.

1. **Create the Remote component**

**3.1** Implement the app2 button:

import React from "react";

const Button = () =>

{

return (

<button onClick={handleClick}>GO TO LOGIN</button>

);

};

const handleClick = () => {

window.location.href = 'http://localhost:3000/';

};

export default Button;

**3.2** Use the Local button in your app2 application:

import LocalButton from "./Button";

import React from "react";

const App = () => (

<div>

<h2>App 2</h2>

<LocalButton />

</div>

);

export default App;

* 1. Import in app1 the remote button from app2:

import React from "react";

const RemoteButton = React.lazy(() => import("app2/Button"));

const App = () => {

return (

<div

<h2>App 1</h2>

<React.Suspense fallback="Loading Button">

<RemoteButton />

</React.Suspense>

</div>

);

}

export default App;

**Containerizing the application with Docker**

1. **Install Docker:** [**https://docs.docker.com/engine/install/**](https://docs.docker.com/engine/install/)
2. **Configure the docker files for each app:**

**5.1** The docker files for app1 and app2 using different ports and paths:

FROM node:16

WORKDIR /app1

COPY package\*.json ./

COPY src/assets/projects.jpg ./

COPY . .

RUN npm install --save-dev webpack-cli@latest

RUN npm install file-loader --save-dev

RUN npm install

EXPOSE 3001

CMD ["npm", "start"]

**5.2** The docker-compose file for the basic-host-remote project:

version: '3'

services:

app1:

build:

context: ./app1

dockerfile: Dockerfile

ports:

- "3001:3001"

container\_name: app1

app2:

build:

context: './app2'

dockerfile: Dockerfile

ports:

- "3002:3002"

container\_name: app2

**5.3** Build and run Docker Containers:

* **docker compose up**

Visit <http://localhost:3001> and <http://localhost:3002> in your browser to see the micro-frontends in action.

Congratulations! You've created a micro-frontends architecture using React, Ionic, Webpack, and Docker.