Microfrontend Example: Create React App Example / Rsbuild

Requirements

1. **Node.js**: Ensure Node.js is installed on your computer. Download it from Node.js.
2. **pnpm**: Install pnpm globally by running `npm install -g pnpm`.

Project Overview

This project demonstrates a basic host application loading a remote component.

- **host**: The host application (CRA-based).
- **remote**: A standalone application (CRA-based) that exposes a `Button` component.

Installing the Application

1. Run `pnpm install` to install all required dependencies and node modules. This may take some time.
2. Run `pnpm install --save-dev jest` to install Jest for testing.
3. Run `pnpm install cypress --save-dev` to install Cypress for end-to-end testing.

Building and Running the Demo

1. Run `pnpm run start`. This will build and serve both `host` and `remote` applications on ports 3001 and 3002 respectively. The command uses the global package.json start scripts: `"start": "pnpm --filter cra_* start"`.
   - Host Dashboard Admin: localhost:3001
   - Standalone Remote - Dashboard Client: localhost:3002

Running Cypress E2E Tests

1. To run tests in interactive mode, execute `npm run cypress:debug` from the root directory. This will open the Cypress Test Runner for interactive testing. More information can be found here.
2. To build the app and run tests in headless mode, execute `yarn e2e:ci`. This command will build the app and execute tests in headless mode. If tests fail, Cypress will create a `cypress` directory in the root folder with screenshots and videos.

3. Additional resources: "Best Practices, Rules, and More".

Important Notes

- Always use `pnpm` instead of `yarn` or `npm` for package management.

Submitting Your Project

- Send your code as a `.zip` file or provide a link to your GitHub repository.
- Include a Notion export as a `.pdf`.

Next Steps: Create a Sports Association Website

1. **Documentation**: Create a Notion (or Confluence or Obsidian) page to document your project like a true feature team. Ideally, create one short page with screenshots for each step. Add me as a contributor at idodgedit@gmail.com.

2. **Shared Component App**

   - Develop a shared library of React components.
   - Ensure these components are reusable and modular.
   - Create a Footer and a Header in the shared component app. Use these in both the host and remote landing pages (app.js).
   - Ideally, expose Footer and Header similarly to how the button is exported in the remote layer: `cra`
     `emote\modulefederationConfig.js`.

3. **Remote App Development (Client's View)**

   - Develop a landing page for both Remote and Host, using the Footer and Header from the Shared Component App.
   - Create a unique component for Remote and another for Host, ideally parameterized (e.g., a component with a customizable title).
   - Integrate Vitest for testing and write basic tests for the components in Remote and Host.

4. **Host App Development (Client View)**

   - Implement a mock API call to fetch and display data. You can use JSONPlaceholder or create your own mock data.
   - Example APIs: MockAPI or Pokemon API.

5. **File Upload (Admin View)**

   - Use the AWS S3 API to upload files (PDFs, images, etc.) to OVH public cloud.

   - Create an account and use the free credits offer: OVH Cloud Storage Trial.
   - Set up a standard object storage: OVH Public Cloud Object Storage.

6. **Read Files from AWS S3 Bucket (Client View)**

   - Display all uploaded files in a dashboard or an array, fetched from OVH Public Cloud.

Bonus Tasks

1. **Deploy the App on a Free Platform as a Service (PaaS)**

   - Options include Vercel, Fly.io, Dokku, or Qovery.

2. **Dockerize the App**

   - Create a Dockerfile and run the app through Docker.
   - Deploy the app using a more complex PaaS that supports Docker, such as Dokku or Qovery.