matt-gold / **single-spa-angular-cli**

Single-spa starter project in an Angular CLI container application.

MIT License

☆ 47 stars 29 forks

☆ Star | Watch ▾

Code | Issues 6 | Pull requests 34 | Actions | Projects | Wiki | Secu

master ▾ ...

matt-gold remove invocation of deprecated single-spa-angular inte… ... on Nov 13, 2020 21

View code

README.md

single-spa-angular-cli

Angular micro-frontend starter using single-spa.

Description

Updated for Angular 10! Single-spa starter project in an Angular CLI container application. This has not been optimized for production, it is recommended to use webpack externals to avoid reloading shared dependencies, and to use resource hints to preload child applications, although your specific use case will ultimately determine the optimal load strategy.

Considerations

Before committing to a micro-frontend architecture, consider the following:

- Do you have the necessary infrastructure to share code across your apps? One way to achieve this while staying DRY is to publish shared code as npm packages in an npm registry internal to your organization. Another option - you could use a monorepo.

- Do you have a component library? Your applications will likely need to look and feel the same. A design system makes this much easier, if your organization does not have one of it's own consider using Google's material or creating a shared component library with something like Storybook.
- Do you have a mature CI/CD pipeline? Micro-frontends can make releases faster and less risky, and give dev teams full autonomy of the SDLC of smaller sections of a large application. However, each micro-app needs to be built and deployed independently, so make sure the overhead of setting up new CI/CD pipelines for each app is taken into account.

Disclaimer

The authors of single-spa generally discourage a parent-child application setup like this because it could lead to tighter coupling between the parent and child applications if you're not careful, and also because it does extra work to manage application life-cycles that single-spa already does for you.

The key difference is that the container app in this repository manages the lifecycles of the single-spa-angular micro-apps manually *using Angular's router* (by making use of single-spa "parcels"), rather than letting single-spa control the application life-cycles and routing (by registering the single-spa-angular micro-apps as single-spa "applications").

Reasons you may prefer the approach implemented here:

- You're an all-Angular shop and want to keep everything within the Angular CLI platform (maybe to reduce the mental overhead for your developers). This approach encapsulates single-spa calls into an Angular service, making single-spa more of an implementation detail.
- You are incrementally adding or experimenting with micro-frontends in a large Angular application, and are not necessarily ready to run everything with single-spa.
- You are migrating an existing Angular application to a micro-frontend whose app-to-app hand-off is complex and simply needs more control than single-spa's activity function provides.

Other resources

This example was originally created as part of this presentation: youtube link

Here are the associated slides: slides