Mindtree
A Larsen & Toubro Group Company

Micro Frontend Architecture with **Single-SPA**

Introduction

Micro Frontend is a design pattern to split large and complex web applications or modules into light-weight web apps that can be deployed independently. Micro frontends are “An architectural style where independently deliverable frontend applications are composed into a greater whole.”[1] These loosely-coupled web apps are integrated into a single container or user interface to create a seamless experience. This concept is very similar to Micro Service pattern or Domain-driven Design pattern.

Why Micro Frontend?

Plenty of and single page applications or isomorphic web apps were developed and deployed over the years. The size and complexity of these web/apps grew and posed similar challenges to that of a monolithic applicatiions. Now, these applications face multiple challenges and issues like performance degradation with each release, non-scalablity, and difficulty in making modifications and ensuring maintainance. Micro frontend architecture is the answer to all these issues.

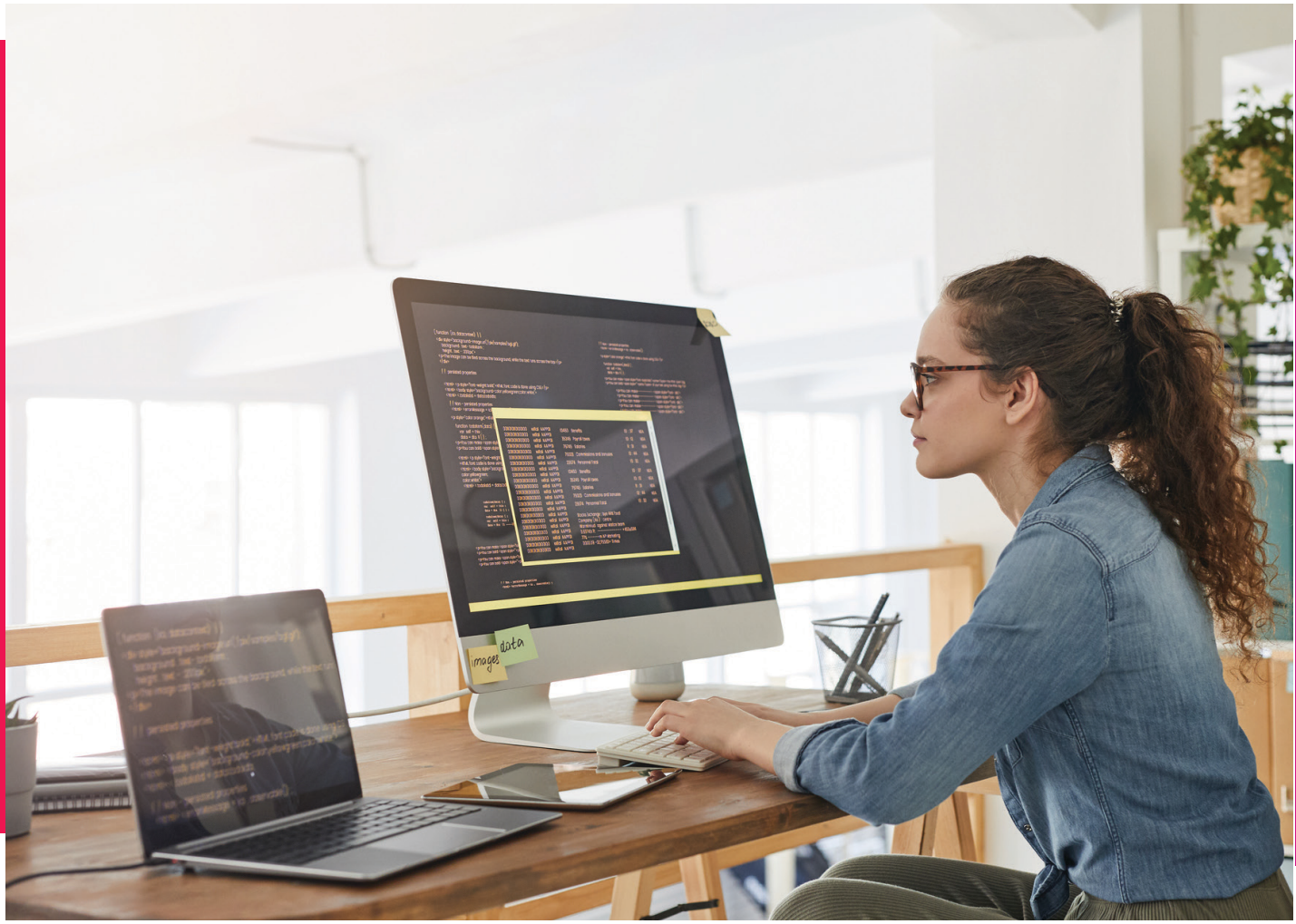
Different ways to implement Micro Frontend

While there are many ways to implement micro frontend, the below are most popular and widely implemented strategies:

- iFrame wherein we embed other web applications into the parent application
- Monorepo wherein many projecs share the same repository
- Single-SPA wherein we use multiple JavaScript libraries for the application

Single-SPA?

Single-SPA is a JavaScript framework that enables the design and development of micro frontend architecture developed using various JavaScript libraries like Angularjs, Angular, React, Vue, Svelte etc.



Micro Frontend Reference Architecture

We have depicted a reference micro front end architecture in Figure 1.

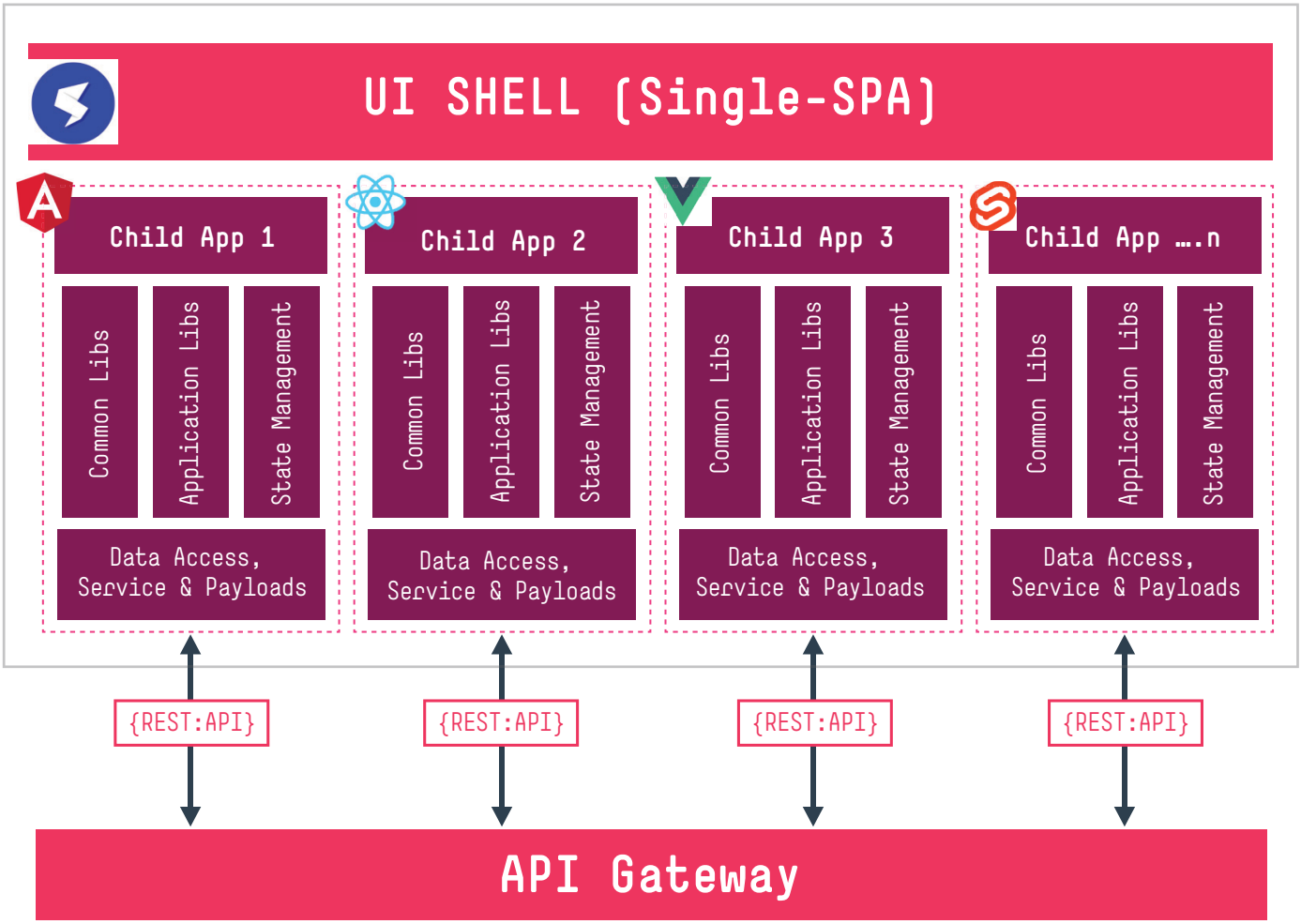
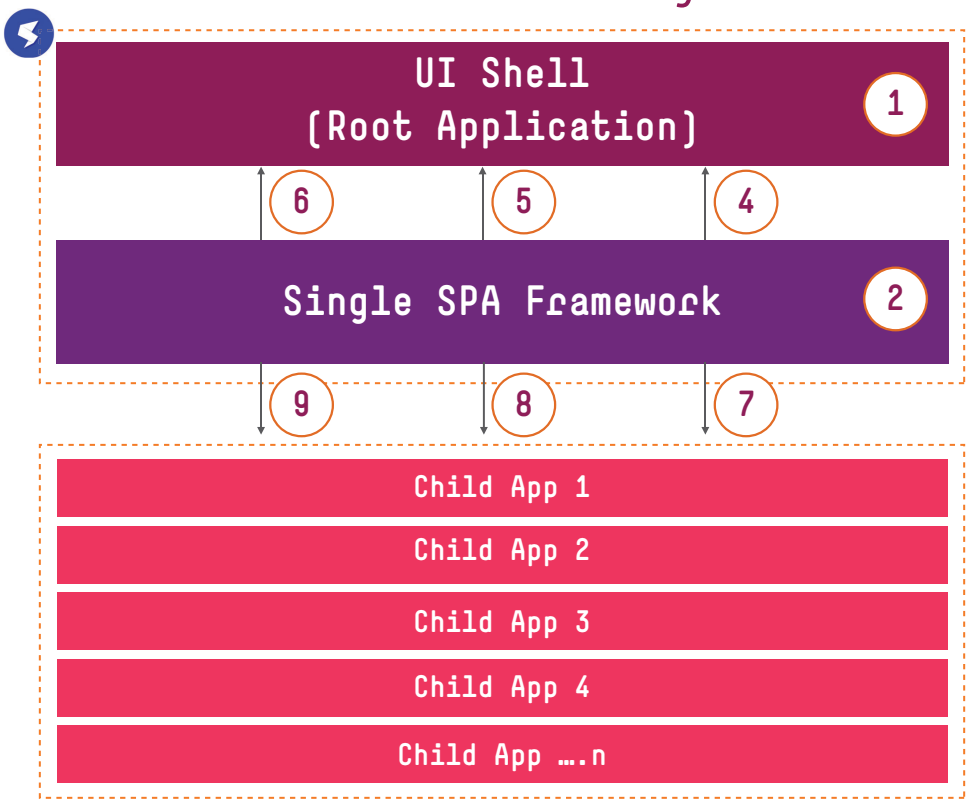


Figure 1 Micro frontend architecture

- Independently deployable and small web applications can be developed using different JavaScript frameworks or libraries like React, Angular, Vue, Svelte etc. These apps are called child apps.
- The main or parent web application works as a container (UI Shell) where all child apps mount and unmount based on user action.
- Each child app has the same or different End Points/API Gateways and use REST API to send and receive data.
- Enterprise applications are the most suitable and common examples of Micro frontend. Every department or business unit can have their own applications based on the business requirement. These applications can be builton different tech stack and have unique URLs, and are micro frontends that are independently developed and deployed at the enterprise level. In this scenario, enterprise user don't have a single entry point to access all enterprise applications. The user needs to visit respective web sites/apps to perform the task. The implementation of iFrame or Single-SPA could provide a single entry point to all enterprise users where all enterprise applications can gain acess from this container.

Single-SPA – UI Shell Life Cycle



1. UI Shell: Multiple micro frontends are integrated into a container/shell by injecting the respective scripts. The container application determines which micro frontend to be mounted and calls the appropriate function to tell the micro frontend when and where to load. The container has its own lifecycle methods to load the applications.

2. Single SPA Framework: Light-weight JavaScript encapsulation framework, which loads the relevant micro apps based on the path that is requested. It has its own lifecycle methods for bootstrapping, mounting and unmounting of the micro apps.

3. Child Application: Independent feature or module-based web app. Each child app has its own code base, which means that each child app can have its own common Lib, intra app state management module and inter apps communication module. Child apps can be developed by a single or multiple teams. Each app can have an independent CI/CD pipeline and DevOps processes.

4. Declare/Register Child Application: Each child app needs to register to the UI shell

5. Active Application/Route: It represents the current active app/route

6. Load Child App: Loads child app based on configured routes

7. Bootstrap: Life cycle function called right before the registered app is mounted for the first time

8. Mount: This lifecycle function will be called whenever the registered application is not mounted. This function should look at the URL to determine the active route and then create DOM elements, DOM event listeners, etc. to render content to the user.

9. Unmount: This lifecycle function is called whenever the registered application is mounted. This function should clean up all DOM elements, DOM event listeners, leaked memory, global and observable subscriptions, etc.

Inter App communication techniques

In this section, we discuss the inter app communication methods.

UI State

- UI state can be shared using the event emitter method
    - Observables (rxjs) - One micro frontend emits values and other consumes it. It exports the observable from its in-browser module and other modules needs to consume the exported values.
    - Custom events implementation using built-in event emitter system of browser. The source micro frontend first implements and fires the custom events, while another micro-frontend needs to subscribe to this event listener The source micro frontend emits the custom events, and other micro frontends subscribe to the events.

API Data

- Since each micro frontend is built on specific domain-driven design or micro services, the user journey will be different for every user, apart from having different sets of data. This means that sharing API data across the micro frontend would be a rare requirement.
- However, this can be achieved by using in-memory JavaScript cache of API object.

Functions, components, logic, and environment variables

Functions, components, logic, data, event emitters, and environment variables can be shared between micro frontends using the export and import feature. First, each micro frontend needs to define one entry file, which works as a public interface for the others. This public interface works as an abstraction layer to control what is exposed outside of the micro frontend. Micro frontends should not have more than one entry file. This module needs to export as external to make sure this is treated as an in-browser module. Other micro frontends need to import this external module.

Best practices to consider while developing micro frontends using Single-SPA

- One of the most important principles to consider while developing micro frontends is user interface consistency. Styling across all micro frontends must be consistent. A strong and detailed style guide would be required to maintain a unified user experience across all web apps.
- If it is a green field development, it is mandatory to ensure technology/framework choices, identification of dependencies, folder structure and coding guidelines before starting development.
- Select only those frameworks which support micro frontend architecture.
- Split the micro frontends in such a manner that data or state sharing would be minimal between them.

- It is good to merge multiple micro frontends into a single app if they are sharing data or state frequently between each other.
- Avoid global state management for all micro frontends.
- Automation is key for successful micro frontend development. Everything should be automated from unit testing to QA testing.
- Aa robust CI/CD pipeline should be built for continuous integration and deployment as multiple teams would work simultaneously.

Advantages of micro frontend

- Web app has a single entry point through the micro frontend
- Modular, reusable, cohesive and manageable code base
- We can create technology agnostic solutions that can be developed by independent teams.
- Web App will be extremely contextual, intuitive and responsive
- We can leverage lazy loading to improve the overall frontend performance.
- Ability to independently upgrade the sections of the frontend independently
- Multiple micro frontends could be loaded on the same page without refreshing the page
- Each micro app could be developed using a different frontend JavaScript framework like AngularJs, Angular, React, Vue etc.
- Easy to share a common access token between child apps
- Better control in terms of authentication and authorization
- Separation of concerns as each micro frontend's design is based on a specific use case or business need
- Small bundle size of each micro frontend delivers better performance
- Provides great flexibility to each micro frontend develop, deploy and test independently
- Multiple development teams can work on a single web app. However, each micro frontend can be developed in silos.
- An independent CI/CD pipeline can be built for each micro frontend and can be deployed without affecting the others
- It is a highly scalable architecture
- Small and structured code base which gives more control on refactor, change and maintainance of the code
- Faster development leads to quick and early roll out to market
- One micro frontend can pass down initialization information, like the rendering target, to another
- Common event bus reference can be passed between micro frontends to enable them to talk to each other
- It supports monorepo architecture as well, meaning all micro frontends can built on top of common reusable libraries/components

Challenges with Micro Frontend

- Sometimes, debugging across multiple microfrontends is challenging
- As each team is working independently, there are high chances of deviation in folder structure and coding standards
- Since teams are working in silos, there is a possibility of redundant code, duplication of dependencies or references
- There is high possiblity that the end user may have diverged user experience
- Micro frontends developed using different JavaScript frameworks increase maintainability
- Not recommended for SEO requirement

Approach for migrating monolith or legacy applications to the modern web

Figure 2 depictes how monolith or legacy applications can be migrated to modern web with a mobile first approach

Figure 2 Monolith to Modern web migration

Main use cases for micro frontend

1. Migrating a large and complex monolith website/app into a modern single page application or decoupled web app
2. Feature/module/user journey-based rapid development and faster roll out
3. Progressive upgrade where new and old web apps coexist in production
4. Step by step graceful decommission of the old web app
5. A collaboration platform that needs to develop the frontend in different technologies

References

1. https://martinfowler.com/articles/micro-frontends.html

About the authors

Bhuvan Amin
Full Stack Architect

Bhuvan Amin is a Full Stack Architect worked on multiple Desktop, Web and Mobile projects. Having work experience in Angular, React, Node, Native Mobile and .Net technologies. Micro Frontend architecture proposed to one of the Mindtree client post detail analysis of existing monolithic multipage server side application during discovery phase.

Dr. Shailesh Kumar Shivakumar
Solution Architect

Dr. Shailesh Kumar Shivakumar has 19+ years of experience in a wide spectrum of digital technologies including, enterprise portals, content management systems, lean portals and microservices. Dr. Shailesh holds a PhD degree in computer science and has authored eight technical books published by the world's top academic publishers such as Elsevier Science, Taylor and Franscis, Wiley/IEEE Press and Apress. Dr. Shailesh has authored more than 14 technical white papers, five blogs, twelve textbook chapters for various under-graduate and post graduate programs and has contributed multiple articles. He has published 20+ research papers in reputed international journals. Dr. Shailesh holds two granted US patents, apart from ten patent applications. Dr. Shailesh has presented multiple research papers in international conferences. Dr. Shailesh's Google Knowledge Graph can be accessed at https://g.co/kgs/4YoaiN . He has successfully led several large scale digital engagements for Fortune 500 clients. Shailesh can be reached at Shaileshkumar.Shivakumarasetty@mindtree.com

About Mindtree

Mindtree [NSE: MINDTREE] is a global technology consulting and services company, helping enterprises marry scale with agility to achieve competitive advantage. "Born digital," in 1999 and now a Larsen & Toubro Group Company, Mindtree applies its deep domain knowledge to 260 enterprise client engagements to break down silos, make sense of digital complexity and bring new initiatives to market faster. We enable IT to move at the speed of business, leveraging emerging technologies and the efficiencies of Continuous Delivery to spur business innovation. Operating in 24 countries across the world, we're consistently regarded as one of the best places to work, embodied every day by our winning culture made up of over 27,000 entrepreneurial, collaborative and dedicated "Mindtree Minds."