Micro Frontends in React.js: A Comprehensive Analysis and Implementation Guide

Prof. Nidhi Upadhyay

Assistant professor

(SAGE University Indore

Prof. Nidhi Singh

Assistant professor

SAGE University Indore

Abstract: Micro frontends is an architectural approach that involves breaking down a monolithic frontend application into smaller, more manageable and independently deployable units. This research paper explores the concept of micro frontends and delves into its practical application within the context of the React.js framework. the benefits and challenges of adopting micro frontends in React.js, has been analyzed in research study and provide a step-by-step implementation guidelines for developers interested in harnessing the potential of this innovative architectural pattern.

Keywords:- Application development, REACTjs, Front-end development, Routing, Navigation, Decoupling, E-commerce, Monolithic, Static website ,frameworks.

Introduction

1.1 Background The rise of complex web applications has led to challenges in maintaining and scaling large monolithic frontend codebases. Micro frontends offer a potential solution to these challenges by enabling the development of smaller, self-contained units that can be composed to form a cohesive user experience.

1.2 **Objectives of research paper:**

- To provide a comprehensive understanding of micro frontends and its relevance in the context of React.js.
- Examine the benefits and drawbacks of implementing micro frontends in a React.js-based application.
- Present a practical guide for implementing micro frontends in React.js, including best practices and design patterns.
- Micro Frontends: Concepts and Principles

2.1 **Definition and core concepts**

Microfrontends are the concepts which are prolong version of micro services which are in use of frontend development. frontend is nothing but our user interface(UI) which is seen at the user end, it is not specific to a certain language. **Microfrontend** is a frontend web development design in which a single application may be built front utterly different builds. It can be seen over our UI in our Application. It can be seen everywhere at every section of our UI, which nowadays we normally call as MFEs. We can divide our UI in every possible small part which is reusable and that MFE can be in use kept on different posts whenever required. for example; An online food delivery app is full of different services like searching of food section, cart section, order and bill section etc. that is managed at the backend. But in the frontend which totally resembles our UI like how does the search bar looks, images of food or any frontend smallest part of UI can be created as MFEs, later on can be used in any other application.

2.2 Comparison with monolithic and other frontend architectural patterns

Monolithic architecture is traditional model of the software which is a complete single unit or we can say a single codebase. Previously the monolithic pattern of our Application used to be complex as the whole application used to be the combination of interrelated data and has to deploy all at once and in case of failure or if any changes required, we had to change the other threads of the application.

Whereas micro services are an architectural pattern to software development in which it contain small individual but independent services which are then combined for a single software. Micro services strategies are used to design and deploy modern applications, but mostly all at the backend level.

<u>Front end architectural patterns:</u>

frontend architectural patterns are:

i. **Single page application(SPAs):** works in a single Browser and only goes to one part to another part to another part of our application without Reloading the entire page.

ii. **Jamstack:** It is an approach to front end web development, which allow us to develop an efficient website which is static for the users.

iii. **Static website creation:** this front end creation is generally based on HTML,CSS or in some point uses JS also. As the name resembles static, its content is always same for all the users unlike the website shows different content related to the users who is using it, just like any company's website is static as it shows info. of the company, every content is related to the company so anyone who reach to this website will interact with the static website i.e., the data or the content will be same for all the users.

iv. **Microfrontends:** this architectural pattern divide web front ends just like micro services where every front end components developed as MFEs or microfrontends. This is helpful in developing modular, scalable and maintainable web applications.

How micro frontends promote better collaboration and scalability

Different teams can work on different parts of the software and this decoupling allows teams to work parallel without interfering the other parts of the software. MFE allows the teams to work according to their choice of technology or language which give freedom to make decisions and work accordingly without waiting for approval. It also allows applications to scale horizontally by dividing the work to different teams and services. It makes easy to solve or deploy large applications by this scalability over time. Also it reuses the code and composition by breaking the UI into smaller components. Also support Isolation and resilience which beneficial at the time of making changes in a component, we do not need to disturb other parts or components of an application. Hence make easier maintenance with such merits.

2.2 **Principles of Micro Frontends**

2.2.1 Decoupling and isolation of frontend modules

Decoupling and Isolation of frontend modules are the core principles in MFEs architecture. As Decoupling implies Independent Development from different teams on different components of front end of an application .It also loose coupling that is components can communicate with each other by APIs or contracts, which enables teams to make changes to one component to another component without imparting to other. Developers are free to choose their technology as decoupling also promote Technology Agnosticism, based on the performance, familiarity or any other requirements.

Whereas isolation maintains Boundary Enforcement which ensures the minimum risk of unintended side effects of other code base. Coders are allowed for independent Deployment due to which they do not need to wait for other teams for updation or fixing bugs etc. e.g with MFEs, if it fails, it will only affect within its limited boundary not to other MFEs or components.

2.2.2 Communication strategies between micro frontends

Commonly used communication strategies between MFEs include:

Pub/Sub patterns which is a messaging pattern where components can publish messages to a central message broker.

Custom Events: this allows simple and direct communication between components without the need for a centralized event bus.

Also Shared Management which trigger updates in other microfrontends subscribing to that state by using Redux or MobX.

The other strategies like Cross- Origin communication, iFrame communication, URL Parameters and Hash Routing, GraphQL, WebSocket communication, APIs Gateway and Shared Libraries.

2.2.3 Potential use of different technologies and frameworks for each micro frontend

In a MFEs architecture, different technologies and frameworks can be used for each micro frontend based on various factors such as specific requirements, team expertise and the nature of the functionality each microfrontend serves.

Here how these technologies and frameworks applied:

Component based architecture and its rich libraries React microfrontend is one of the best choice. This can be used for dynamics and interactive UI. With this we can use technologies like Material-UI for styling, Redux for state management, React Router for Routing etc.

Vue.js microfrontend Provide framework for working with ease , simplicity and productivity. Here for state management it provides Vuex, Vue Router for routing and libraries like bootstarpVue styline.

There are other technologies and Frameworks such as Angular Microfrontend, Svelte Microfrontend, web components, Server-Side Rendering(SSR) microfrontend and Static Site Generation(SSG) microfrontend work potentially for microfronends.

Perks and issues of Micro Frontends in React.js

3.1 **Perks**

Improved team autonomy and efficiency: with microfrontends teams can work on different parts of the applications independently, they focus on specific components, due to which they move faster in their production. Microfrontends empowers teams to make autonomous decisions regarding technologies.

Independent deployment and scalability: Each microfrontend can developed, tested and deployed independently, also allowing for faster and more autonomous updates. Also microfrontends can be scaled on individual basis, according to their specific needs, optimizing resource usage and performance.

Flexible technology stack choices for each micro frontend: every MFE can use its own technology, allowing teams to choose the well suited tools for the specific need.

Easier code maintenance and debugging: Due to smaller and isolated code for every microfrontend give ease for maintenance and debugging efforts

3.2 **Issues**

Complexity in inter-micro frontend communication: Due to the need of better interfaces and protocols it is very difficult to manage communication between microfrontends. Consistent data flow and management of state of microfrontends leads to the complexity. to revolve these challenges proper coordination and interaction strategies are needed.

Shared state management between micro frontends: Require careful coordination among microfrontends as sharing of state among microfrontends is challenging, centralized state management can be used for it.

Potential performance implications and optimization techniques: microfrontend introduces latency and more resource usage, Techniques which can be used for optimization are cashing, lazy loading and efficient communication protocols to resolve these issues.

Implementing Micro Frontends in React.js: It involves several steps an strategies:

Steps to implement:

i. choosing a framework: This includes a webpack which allows dynamic loading of microfrontends, single page application for integrating multiple frameworks and bits for sharing and managing components of React.

ii. Setting up microfrontends:

- for each microfrontend, separate react application must be created.

- Every application must be configured with 'package.json', dependencies and build configuration.

iii. In each microfrontend's configuration setup module federation Plugin:

```
// Example webpack.config.js
module.exports = {
  plugins: [
    new ModuleFederationPlugin({
      name: 'microfrontendName',
      filename: 'remoteEntry.js',
      exposes: {
        './Component': './src/Component',
      },
      shared: { react: { singleton: true }, 'react-dom': { singleton: true } },
    }),
  ],
};
```

iv.Host Applicaion Setup:

- Create Host Application that willload and display the microfrontends.
- Use dynamics imports to load microfrontends

```
// Example host application code
import React, { Suspense, lazy } from 'react';
const Microfrontend = lazy(() => import('microfrontendName/Component'));

const App = () => (
  <Suspense fallback="Loading...">
    <Microfrontend />
  </Suspense>
);
```

export default App;

v. Routing and Navigation: Use React Router for navigation within microfrontends

vi. Implement a Global State Management solution example: ContextAPI for sharing the state.

4.1 **Project Setup and Architecture Design**

Designing the micro frontend structure: Organize microfrontends folder structure, utilize Git for version control and use package manager like npm to manage dependencies.

Architecture design:

i. Identify responsibilities and function for microfrontends

ii. choose integration approach such as web components.

iii. Routing mechanism for navigation among microfrontends.

iv. setting up CI/CD pipelines for automated build,test and deployment.

v. use testing and monitoring for ensuring the performance of microfrontends.

Establishing communication channels:

- must use events to notify other MFEs of state change.
- creation of shared services or APIs to facilitates interaction among MFEs

Handling shared state and data: choose a suitable data management solution like context API, for shared state between microfrontends. With the help of props drilling or context provides for sharing data between MFEs.

4.2 **Routing and Navigation**

Handling navigation across micro frontends: for handling navigation across microfrontends we must use a consistent routing structure with React Router. Integrate navigation components in individual MFEs. Use shared state for cross-microfrontend navigation. Ensure accessibility and error handling for a seamless UI.

Implementating a unified Routing Strategies: First we must Define routes consistently across MFEs using React Router. This centralize routing configuration. Use Browser events or a shared state for Seamless navigation between MFEs.

4.3 **Communication Strategies**

In MFEs, communication strategies are essential for interaction and sharing of data. Common method include custom events, which can be hard to manage with different events, they are simple and decoupled. Redux works for shared state, provide centralized, predictable state management with increased complexity and tighter coupling. URL or query parameter offer a stateless, easily sharable options. Local storage allows for persistent data across sessions. The post message API facilitates secure communication. Hence choosing the right strategy depends on real time need and data complexity.

4.4 **Styling and Theming**

Approaches to managing styles and theming in a micro frontend architecture.

- To encapsulate styles in each MFEs using CSS modules.
- Define global variables for themes to ensure consistent look of MFEs.
- Utilization of common design system for standardizing styles.
- While allowing customization and maintaining uniformity, implementation of a shared CSS framework like Bootstrap.
- Dynamic theming allows themes to be changed at the runtime using CSS variable.
- Avoidance of style conflicts by using unique prefixes for class names.

4.5 **Testing and Quality Assurance**

Testing and Quality Assurance in MFEs ensures functionality and integration across the system, which involves unit testing by using tool called Jest for each component, integration testing by using Cypress for the verification of communication among MFEs. Consistent interfaces between MFEs using Tool Pact in contract testing. Detection of avoidable UI changes by using visual Regression Testing with tool called Percy. Also performance and compliance with standards like WCAG BY performance Testing and accessibility Testing Respectively. Consistent Integration or continuous utilization pipelines automates testing and deployment and vigilance tools like Promethers and Grafrana helps maintain quality in production.

Case Study : E-commerce Application

Microfrontends in React.js enhance the flexibility and maintainability of an e-commerce platform, which can be demonstrated.

We can divide our e-commerce platform into smaller independent modules, in which every module is responsible for a particular task. For example, we can have separate MFE for search panel, cart, payment etc. Also different teams can use their own workflow and work independently on different MFEs. This will increase speed of development and also if any changes required only those MFEs should be change, which will not impact the other parts of the application. Also this allows horizontal scaling which ensures optimal performance and resource utilization.

Hence the microfrontends works as a pro in e commerce applications.

6.1 **Summary of Findings**

Microfrontends have number of benefits like better scalability, independent development, faster in marketing by allowing each team to work independently on different parts of an application. They also benefit in terms of Technology diversity, allowing every team to select best tool for certain tasks.

However some challenges arise in maintaining consistency between MFEs, managing communication and sharing of data between them. Beside these challenges the merits of MFEs overcome the drawbacks by implementing with proper planning, vigorous communication strategies, effective testing and improved quality.

6.2 **Future Outlook**

Potential areas of improvement and research in the micro frontend approach within the React.js ecosystem:

This include performance optimization through reduced bundle size and effective component loading, advance state management for better scalability and consistency, Robust communication for interaction between MFEs.

Research into security, error management and debugging is also needed. Research of user experience of accessing microfrontends architecture on usability.

References :-

- **https://readmedium.com/the-ultimate-guide-to-micro-frontends-development-417b552a56c0**
- **https://www.addwebsolution.com/blog/angular-micro-frontends-architecture**
- **https://sematext.com/glossary/micro-frontend/**
- **https://blog.openreplay.com/what-are-microfrontends/**
- **https://www.ramotion.com/blog/micro-frontends/**
- **https://dev.to/scofieldidehen/micro-**