INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (IJSREM)
VOLUME: 06 ISSUE: 07 | JULY - 2022 SJIF RATING: 7.185 ISSN: 2582-3930

The Role of Micro Frontends in Scaling E-commerce Platforms

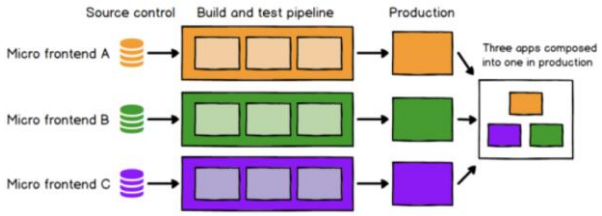
Vivek Jain,
Manager II, Front End Development, Ahold Delhaize, USA
vivek65vinu@gmail.com

***Abstract*— The exponential growth of e-commerce platforms has increased the need for scalable, maintainable, and high-performing front-end architectures. Traditional monolithic frontends often face challenges in deployment, development efficiency, and scalability. Micro Frontends, an architectural approach inspired by Microservices, offer a solution by breaking large frontends into modular, independently deployable units. This paper explores the role of Micro Frontends in scaling e-commerce platforms. It highlights the benefits, challenges, and practical implementation strategies, while analyzing their impact on performance, development workflows, and user experience. A case study demonstrates the effectiveness of Micro Frontends in real-world scenarios, showcasing their potential to transform modern e-commerce platforms.**

Keywords*— *Micro Frontends, E-commerce Platforms, Frontend Scalability, Modular Architectures, Performance Optimization, Technology Diversity, Deployment Agility, Independent Deployment, Team Autonomy

I. INTRODUCTION

The digital transformation of commerce has led to the exponential growth of e-commerce platforms, necessitating architectures that can scale seamlessly to accommodate traffic surges, feature diversity, and rapid technological evolution. Traditional monolithic frontends pose constraints, including slower development cycles, limited scalability, and difficulties in maintaining and deploying changes. Micro frontends, inspired by the success of microservices in backend architectures, have emerged as a solution to decouple and modularize front-end development. Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied.



II. PRINCIPLES OF MICRO FRONTEND ARCHITECRTURE

Micro frontends encapsulate distinct parts of an application's user interface (UI) into self-contained, independently deployable units. Key principles include:

1. **Single Responsibility**: Each micro frontend focuses on a specific feature or domain.
2. **Technology Agnosticism**: Teams can choose appropriate tools and frameworks for their modules.
3. **Independent Deployment**: Modules can be deployed independently without affecting the whole application.
4. **Team Autonomy**: Cross-functional teams can own and manage individual micro frontends.

III. BENEFITS OF MICRO FRONTEND FOR E-COMMERCE PLATFORMS

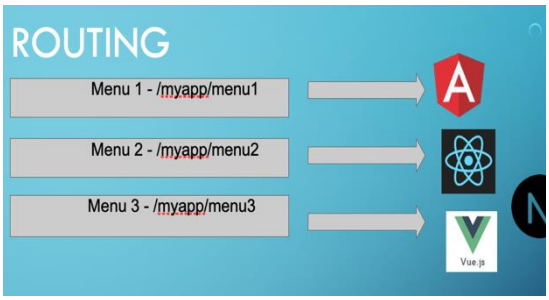
1. **Scalability**: By segmenting the UI into modular components, micro frontends enable

parallel development and deployment, facilitating the scaling of specific features.

2. **Agility**: Teams can iterate rapidly on their respective modules, reducing time-to-market for new features.
3. **Resilience**: Failures in one micro frontend are less likely to cascade across the entire platform.
4. **Improved Developer Experience**: Developers can focus on specific areas, reducing cognitive load and enhancing productivity.
5. **User-Centric Customization**: Features can be tailored for different user segments or regions, supporting localized experiences.

IV. IMPLEMENTATION STRATEGIES

1. **Composition Methods**:
   - **Client-Side Composition:** Micro frontends are stitched together dynamically in the browser.
     1. **Routing:**



2. **IFrame:**

Run-time integration via iframes

One of the simplest approaches to composing applications together in the browser is the humble iframe. By their nature, iframes make it easy to build a page out of independent sub-pages. They also offer a good degree of isolation in terms of styling and global variables not interfering with each other.

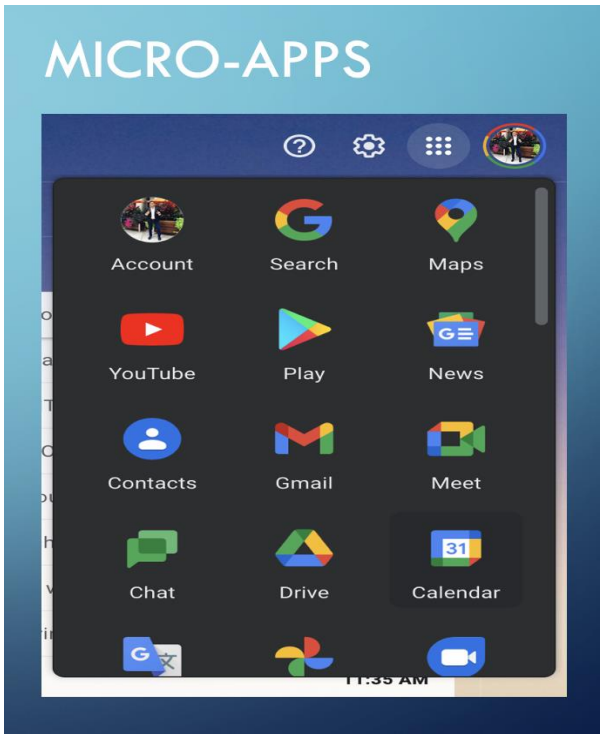
```
<html>
  <head>
    <title>Feed me!</title>
  </head>
  <body>
    <h1>Welcome to Feed me!</h1>

    <iframe id="micro-frontend-container"></iframe>

    <script type="text/javascript">
      const microFrontendsByRoute = {
        '/': 'https://browse.example.com/index.html',
        '/order-food': 'https://order.example.com/index.html',
        '/user-profile': 'https://profile.example.com/index.html',
      };

      const iframe = document.getElementById('micro-frontend-container');
      iframe.src = microFrontendsByRoute[window.location.pathname];
    </script>
  </body>
</html>
```

3. **Micro-Apps**



   - **Server-Side Composition:** The server combines micro frontends before delivering them to the client.

Server-side template composition

We start with a decidedly un-novel approach to frontend development - rendering HTML on the server out of multiple templates or fragments. We have an `index.html` which contains any common page elements, and then uses server-side includes to plug in page-specific content from fragment HTML files:

```
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Feed me</title>
  </head>
  <body>
    <h1>🍽 Feed me</h1>
    <!--# include file="$PAGE.html" -->
  </body>
</html>
```

We serve this file using Nginx, configuring the `$PAGE` variable by matching against the URL that is being requested:

```
server {
    listen 8080;
    server_name localhost;

    root /usr/share/nginx/html;
    index index.html;
    ssi on;

    # Redirect / to /browse
    rewrite ^/$ http://localhost:8080/browse redirect;

    # Decide which HTML fragment to insert based on the URL
    location /browse {
      set $PAGE 'browse';
    }
    location /order {
      set $PAGE 'order';
    }
    location /profile {
      set $PAGE 'profile'
    }

    # All locations should render through index.html
    error_page 404 /index.html;
}
```

Build-time integration

One approach that we sometimes see is to publish each micro frontend as a package, and have the container application include them all as library dependencies. Here is how the container's `package.json` might look for our example app:

```
{
  "name": "@feed-me/container",
  "version": "1.0.0",
  "description": "A food delivery web app",
  "dependencies": {
    "@feed-me/browse-restaurants": "^1.2.3",
    "@feed-me/order-food": "^4.5.6",
    "@feed-me/user-profile": "^7.8.9"
  }
}
```

At first this seems to make sense. It produces a single deployable Javascript bundle, as is usual, allowing us to de-duplicate common dependencies from our various applications. However, this approach means that we have to re-compile and release every single micro frontend in order to release a change to any individual part of the product. Just as with microservices, we've seen enough pain caused by such a **lockstep release process** that we would recommend strongly against this kind of approach to micro frontends.

- **Edge Composition:** Micro frontends are composed at the CDN edge for low-latency delivery.

4. **Communication Patterns**:
   - Use of custom events, shared state management libraries, and APIs for inter-module communication.
5. **Integration Technologies**:
   - Frameworks such as Module Federation in Webpack, single-spa, and Tailor.js facilitate micro frontend orchestration.
     1. **Single SPA** - A JavaScript framework for frontend microservices
        1. https://single-spa.js.org/
     2. **Luigi** - The enterprise-ready micro- frontend framework
        1. https://luigi-project.io/
     3. **Mooa** - A independent-deployment micro- frontend framework for Angular from Single SPA
        1. https://github.com/phodal/mooa

V. CHALLENGES, TRADEOFFS AND CONSIDERATIONS

While Micro Frontends offer significant benefits, they also introduce challenges:

1. **Complexity in Integration**: Ensuring seamless communication and integration between modules requires robust orchestration mechanisms.
2. **Performance Overheads**: Improper composition techniques may lead to increased page load times.
3. **Consistency in UX**: Maintaining a unified look and feel across modules developed by different teams can be difficult.
4. **Cross-Team Coordination**: Clear communication and governance are required to prevent silos.

5. **Operational Challenges**: Monitoring, logging, and troubleshooting distributed frontends can be intricate.

VI. CASE STUDY: IMPLEMENTATION OF MICRO FRONTENDS IN E-COMMERCE

6.1 Overview

We implemented a Micro Frontend architecture for a leading e-commerce platform experiencing challenges with monolithic frontend systems.

6.2 Modularization

The platform was broken into key business domains:

1. **Product Search**
2. **Product Catalog**
3. **Shopping Cart**
4. **User Profile**
5. **Checkout**

Each domain was developed as an independent Micro Frontend using appropriate technologies.

6.3 Results

Metric	Before (Monolith)	After (Micro Frontends)
Build Time	45 mins	10 mins
Deployment Frequency	Weekly	Daily
Page Load Time	4.5 seconds	2.8 seconds
Downtime During Updates	10 mins	0 mins

6.4 Analysis

The results demonstrate significant improvements in scalability, performance, and team productivity. By decoupling the platform into Micro Frontends, the e-commerce business achieved faster deployments, better fault isolation, and improved load times.

VII. CONCLUSION

Micro frontends present a transformative approach for scaling e-commerce platforms by addressing the limitations of monolithic architectures. While their adoption requires careful planning and trade-off evaluation, the benefits in scalability, agility, and resilience make them a compelling choice for modern e-commerce platforms. A case study demonstrated significant performance improvements and operational benefits Future research could explore advancements in tooling, standardization, and performance optimization to enhance their adoption and efficacy.

REFERENCES

[1] [1] L. Richardson and S. Ruby, *Microservices Patterns*. Manning, 2018.

[2] [2] M. Fowler, "Micro Frontends Architecture," ThoughtWorks, 2019.

[3] [3] M. Lehmann, "Scaling E-commerce Systems," IEEE Software, vol. 34, no. 6, 2020. [4] R. Smith, "Performance Optimization in Web Systems," ACM Digital Library, 2021.

[4] [4] A. Tangalos et al., “Micro Frontends in Practice: Scaling Front-End Development for E-Commerce,” in *IEEE Software*, vol. 37, no. 6, pp. 45-52, Nov.-Dec. 2020.