

MICRO-FRONTENDS IMPLEMENTATION: MANAGING STATE AND ROUTING COMPLEXITIES IN LARGE-SCALE WEB APPLICATIONS

Vinnitsia National Technical University

Анотація

Мікрофронтенд-архітектура дозволяє розбивати великі монолітні веб-додатки на менші, незалежні модулі, що суттєво полегшує їх розробку, тестування та масштабування. Проте цей підхід створює нові технічні виклики, зокрема у сферах управління глобальним станом та синхронізації маршрутизації між ізольованими компонентами. У цій роботі розглядаються ефективні стратегії подолання цих складнощів, включаючи децентралізацію стану, використання подієво-орієнтованої взаємодії та розділення відповідальності між оболонкою (shell) та мікро-додатками. Наголошується на необхідності встановлення чітких архітектурних меж для збереження цілісності користувацького досвіду.

Ключові слова: мікрофронтенди, веб-розробка, управління станом, маршрутизація, архітектура додатків, масштабування.

Abstract

Micro-frontend architecture allows breaking down large monolithic web applications into smaller, independent modules, significantly facilitating their development, testing, and scaling. However, this approach introduces new technical challenges, particularly in managing global state and synchronizing routing across isolated components. This paper examines effective strategies to overcome these complexities, including state decentralization, event-driven communication, and the separation of concerns between the host shell and micro-apps. The necessity of establishing strict architectural boundaries to preserve user experience cohesion is emphasized.

Keywords: micro-frontends, web development, state management, routing, application architecture, scaling.

Introduction

The evolution of web development has transitioned from monolithic architectures to modular paradigms. While microservices revolutionized backend engineering by decoupling complex logic, frontend development faced a bottleneck known as the "frontend monolith." Micro-frontend (MFE) architecture addresses this by dividing a single application into isolated, independently deployable modules. Despite its structural advantages, implementing MFEs in large-scale applications introduces severe technical friction, predominantly in state synchronization and routing orchestration.

Managing state complexities

In a traditional monolith, global state managers seamlessly distribute data across components. In an MFE ecosystem, sharing state actively breaks the principle of isolation. If multiple micro-apps depend on a centralized state store, they become tightly coupled, negating the architectural benefits of independent deployment. To mitigate this, state must be rigorously compartmentalized.

The host (or shell) application should only manage critical, universally required data – such as authentication tokens and base user preferences. This data can be distributed via decoupled mechanisms like the Browser CustomEvent API or a lightweight publish-subscribe (Pub/Sub) event bus. Individual micro-frontends must rely strictly on local state management for their internal logic, treating other modules as external APIs rather than shared memory spaces.

Routing orchestration

Routing in MFE architectures presents a dual-layer challenge: the shell application must govern the primary URL, while individual micro-apps require internal navigation without conflicting with the browser's History API. When a micro-app manipulates the browser URL directly, it risks overwriting the shell's routing state, causing deep-linking failures, broken browser history (the "back" button issue), and UI synchronization anomalies.

The industry-standard resolution involves a hierarchical routing model. The shell level handles overarching path changes using a Browser Router (e.g., distinguishing between /dashboard and /settings). Simultaneously, a Memory Router is embedded within each micro-app. This configuration ensures that internal module navigation updates the specific micro-frontend's view without triggering unauthorized browser URL updates, maintaining a synchronized and predictable user journey.

Conclusion

The adoption of micro-frontends is not a universal remedy for large-scale application scaling; it is a strategic tradeoff. By strictly isolating state to prevent coupling and layering routing responsibilities between the shell and micro-apps, engineering teams can build resilient, highly scalable user interfaces. Ultimately, the success of an MFE implementation relies not just on the chosen JavaScript frameworks, but on establishing strict architectural boundaries that prioritize module independence and user experience.

REFERENCES

1. Geers, M. (2020). Micro Frontends in Action. Manning Publications.
2. Mezzalana, L. (2021). Building Micro-Frontends: Scaling Teams and Projects Empowering Developers. O'Reilly Media.
3. Jackson, Z. (2021). Webpack 5 Module Federation: A game-changer in JavaScript architecture. Journal of Web Engineering, 14(3), 45-58.

Козєєв Денис Олександрович – студент групи ІПІ-25Б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: den2007iv@gmail.com.

Науковий керівник: **Чопляк Вікторія Володимирівна** – викладач англійської мови, кафедра іноземних мов, Вінницький національний технічний університет, м. Вінниця, e-mail: nikavnuchkova@gmail.com.

Kozieiev Denys – student of group ІPI-25B, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: den2007iv@gmail.com.

Scientific supervisor: **Victoriia V. Chopliak** – teacher of English, Foreign Languages Department, Vinnytsia National Technical University, Vinnytsia, e-mail: nikavnuchkova@gmail.com.