DOI: https://doi.org/10.23939/csn2024.02.219
УДК 004.8, 004.932

МЕТОДОЛОГІЯ ВПРОВАДЖЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ІЗ ВИКОРИСТАННЯМ МІКРОІНТЕРФЕЙСІВ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ ТА ШВИДКОСТІ ЇХ РОЗРОБКИ

О.В. Степанов, Г.І. Клим

Національний університет «Львівська політехніка»,
Кафедра спеціалізованих комп'ютерних систем
E-mail: oleksandr.v.stepanov@lpnu.ua, halyna.i.klym@lpnu.ua

Мікросервіси являють собою підхід до розроблення програмного забезпечення, варіацію сервіс-орієнтованої архітектури, яка структурує додаток як набір слабо пов'язаних сервісів. У даній роботі досліджено методологію проєктування та впровадження інформаційних систем із використанням мікро-інтерфейсів для підвищення якості та швидкості розробки, водночас спрощуючи їх використання. У роботі запропоновано метод переходу від монолітної архітектури програмного забезпечення до мікросервісної архітектури. Наводиться короткий огляд існуючих досліджень щодо реінжинірингу архітектур та виділяються переваги впровадження мікросервісного підходу. Експеримент із типовим зовнішнім односторінковим додатком ілюструє порівняння ефективності та продуктивності запропонованих архітектур. Розглянуто можливий майбутній напрямок розвитку мікрофронтендної архітектури у поєднанні з WebAssembly та проаналізовано переваги та недоліки даного поєднання технологій.

Ключові слова: архітектура, інтерфейс, мікроінтерфейс, мікросервіси, монолітна структура, програмні додатки, WebAssembly.

1. Вступ

До недавнього часу більшість програмного забезпечення не була особливо складною з точки зору розробки чи змісту. Однак за останні роки складність та вартість розробки програмного забезпечення значно зросла, що негативно вплинуло на загальну структуру програмного забезпечення. Це зростання складності впливає на користувацькі інтерфейси та вимагає використання складніших інструментів розробки [1].

При зростанні складності додатків збільшився й обсяг JavaScript-коду, що призводить до довших термінів розробки, триваліших періодів тестування та затримок між релізами. Великі додатки, що стикаються з цими проблемами, часто називають монолітами через їхню архітектурну побудову.

Монолітна архітектура характеризується об'єднанням усієї основної логіки в один єдиний компонент, де різні елементи комбінуються в одному шарі [1,2]. Ця тенденція породила потребу в новій концепції, яка могла б як покращити функціональність додатків, так і спростити процес їх розробки. Концепція мікрофронтендної архітектури, орієнтованих на фронтенд-розробку, та мікросервісів, націлених на бекенд, виникла для вирішення цієї проблеми. Архітектура мікрофронтендів передбачає розбиття великих монолітних додатків на менші, незалежно впроваджувані одиниці, відомі як мікро-інтерфейси. Кожен мікро-інтерфейс виконує конкретну функцію, але залишається частиною більшої системи. Це розділення дозволяє командам розробників працювати над різними частинами додатка незалежно, прискорюючи цикли розробки та спрощуючи обслуговування [3-5].

Це дослідження зосереджено на впровадженні концепції мікрофронтендів у проєктування та розробку інформаційних систем. Ці системи, які є по суті електронними мережами взаємопов'язаних компонентів для збору, оброблення, зберігання та поширення даних, можуть значно виграти від технік

розподілу програмних компонентів. Даний підхід може призвести до ефективнішого оброблення та керування даними [6].

2. Огляд літературних джерел

Критичною проблемою в архітектурах мікрофронтендів є ситуація, коли внутрішня комунікація між мікроінтерфейсами (МІ) стає надмірною і неконтрольованою. Замість сприяння незалежності, ці МІ стають надмірно залежними від постійного обміну даними, що може призвести до вузьких місць у комунікації. Така надмірна комунікація підриває автономію кожного МІ, що в кінцевому підсумку сповільнює загальну продуктивність додатка. Проблема ще більше ускладнюється, коли шаблони комунікації не регулюються належним чином, через що система починає нагадувати жорстко пов'язану монолітну архітектуру, а не розподілену, модульну систему [7, 8].

Цей збій у контролі за комунікацією підриває основну мету МІ: створити незалежні, самодостатні одиниці, здатні функціонувати автономно. Коли ці фронтенди надмірно залежать від внутрішніх повідомлень, це зменшує їх здатність до незалежного впровадження та масштабування, що нівелює очікувані переваги архітектури.

Надмірно пов'язані МІ не лише створюють проблеми комунікації, але й призводять до значних проблем із продуктивністю. Коли МІ діляться даними та функціями неконтрольованим чином, виникають операційні неефективності. Таке поєднання уповільнює процеси впровадження, ускладнює масштабування та часто призводить до постійного зниження продуктивності всієї системи.

Крім того, відсутність контролю над внутрішньою комунікацією призводить до зайвих витрат на обробку, оскільки кілька МІ постійно обмінюються даними, що збільшує затримку та споживання ресурсів. Відсутність добре структурованого механізму комунікації в архітектурах МІ може безпосередньо вплинути на чутливість додатка, порушуючи принципи архітектури, спрямовані на підвищення продуктивності та масштабованості [9-11].

3. Постановка задачі

Мета цього дослідження — вивчити методології проєктування та впровадження інформаційних систем з використанням мікроінтерфейсів з метою підвищення якості та ефективності їх розробки та шаблонів комунікації, а також покращення загального досвіду користувачів.

4. Методи комунікації мікроінтерфейсів

Переваги та недоліки комунікації мікроінтерфейсів

У монолітних архітектурах усі домени знаходяться в межах одного додатка, що забезпечує простий обмін даними між доменами. Різні фреймворки пропонують механізми для передачі даних через ці доменні межі, причому одним із загальних підходів є збереження даних на вищому рівні, після чого вони передаються вниз до різних доменів за допомогою props(параметрів).

Однак цей процес ускладнюється, коли додаток розбивається на кілька невеликих, незалежно працюючих додатків, що вбудовані в один контейнерний додаток, відомий як МІ. Хоча цей архітектурний підхід має численні переваги, він також вносить певні складнощі, однією з яких є комунікація між МІ.

Використання props

Використання пропсів представляє найосновніший метод забезпечення комунікації між МІ, коли контейнерний додаток керує станом і передає його до відповідних МІ. Цей підхід є особливо ефективним у сценаріях із використанням шаблону інтеграції МІ на етапі збірки, оскільки він дозволяє додатку рендерити МІ як компоненти та передавати необхідні props до них.

Цей метод широко визнаний у компоненто-орієнтованих архітектурах та має широку підтримку в різних фреймворках. Він використовує структури, специфічні для фреймворків, як-от Context API в React, щоб мінімізувати проблеми, такі як глибоке передавання props, тим самим спрощуючи керування станом у додатку.

Однак цей підхід створює значну залежність між МІ та контейнерним додатком, що може бути проблематичним. Це особливо складно, коли МІ використовують різні фреймворки, оскільки цей підхід вимагає спільного фреймворка для ефективної комунікації. Крім того, продуктивність додатка може бути знижена, оскільки часті зміни стану можуть призводити до перерендерингу кількох непотрібних шарів, що викликає неефективність.

Платформенні інтерфейси для зберігання даних додатків

У цьому підході використовуються специфічні для платформи вбудовані інтерфейси зберігання, такі як Local Storage у веб-браузерах і Async Storage у кросплатформених фреймворках, таких як React Native, для керування станом у мобільних МІ. Просту бібліотеку збереження можна розробити для абстрагування цих інтерфейсів зберігання, надаючи єдиний інтерфейс із функціями зчитування та запису. Це дозволяє МІ взаємодіяти з даними напряму, обходячи необхідність комунікації через контейнерний додаток.

Цей метод застосовний як для інтеграції МІ на етапі збірки, так і під час виконання. Однак, оскільки інтерфейси зберігання не підтримують вбудовану модель підписки на зміни, МІ не зможуть автоматично виявляти та реагувати на оновлення даних, здійснені іншими МІ. Для подолання цих обмеження необхідно впровадити асинхронну модель публікації та підписки. Ця техніка особливо корисна, коли МІ рендеряться на окремих екранах, оскільки читач даних може отримати останню інформацію під час завантаження.

Цей підхід сумісний як із веб-браузерами, так і з мобільними пристроями, використовуючи Local Storage для браузерів і Async Storage для мобільних додатків. Він створює менше зв'язку між контейнерним додатком і МІ, порівняно з методом передачі props. Однак може виникати складність у відстеженні, який саме МІ відповідає за зміну певних даних.

Цей метод ефективний для керування невеликими наборами даних, але не масштабований для великих додатків. Щоб уникнути плутанини, доцільно іменувати простори даних у сховищі відповідно до назви додатка. Проте цей метод не підходить для зберігання чутливих або захищених даних через відсутність вбудованих заходів безпеки.

Користувацькі події

Ця техніка особливо добре підходить для веб-базованих МІ і надає більш масштабоване рішення для архітектур МІ на етапі виконання. Основна ідея полягає у використанні вбудованого інтерфейсу браузера для Custom Events. що дозволяє забезпечити подієво-орієнтовану комунікацію між МІ. У цьому підході один МІ публікує події з даними, а інші МІ підписуються на ці події, щоб отримати дані.

Ця методологія тісно нагадує подієво-орієнтовану архітектуру, що зазвичай використовується в мікросервісах. Ця техніка застосовна як для інтеграції на етапі збірки, так і для виконання МІ, з оптимальною продуктивністю, коли перехресна комунікація відбувається на одній сторінці, оскільки події мають бути підписані до їх публікації.

Цей метод використовує вбудовані можливості платформи браузера, тісно повторюючи асинхронну подієво-орієнтовану архітектуру, що часто використовується в мікросервісах. Хоча початкові витрати на налаштування можуть бути значними, цей метод пропонує чудову масштабованість і сприяє встановленню стандартизованого механізму, який може бути прийнятий усіма командами, що працюють над мікрофронтендами. Проте, він не підходить для мобільних МІ.

Використання post-меседжів у iframe

Ця техніка здебільшого застосовна при розробці МІ на етапі виконання з використанням iframe. Хоча iframe можуть не бути оптимальним шаблоном інтеграції для створення МІ, цей підхід може бути особливо ефективним, коли необхідно вбудувати МІ у наявний, діючий додаток.

Запропонований шаблон комунікації для МІ

У цій роботі пропонується гібридна стратегія комунікації, яка поєднує сильні сторони як подієво-орієнтованих методів, так і інтерфейсів збереження даних платформи для оптимізації комунікації між різними компонентами МІ.

Гібридна модель комунікації розроблена для адаптації до різних випадків використання в архітектурі МІ (рис. 1). Вона дозволяє розробникам вибирати між двома шляхами комунікації в залежності від вимог до даних і особливостей платформи фронтенду:

1. Подієво-орієнтована комунікація: Ідеально підходить для реального часу, асинхронної комунікації між МІ. Цей підхід може бути реалізований з використанням браузерних інтерфейсів, таких як Custom Events, або шляхом впровадження власної системи повідомлень.

2. Інтерфейси збереження платформи: Придатний для постійних даних, які не потребують оновлень у реальному часі. Цей метод використовує сховища браузера або мобільної платформи (такі

як Local Storage або Async Storage) і є особливо ефективним, коли МІ не потребують частого спілкування.

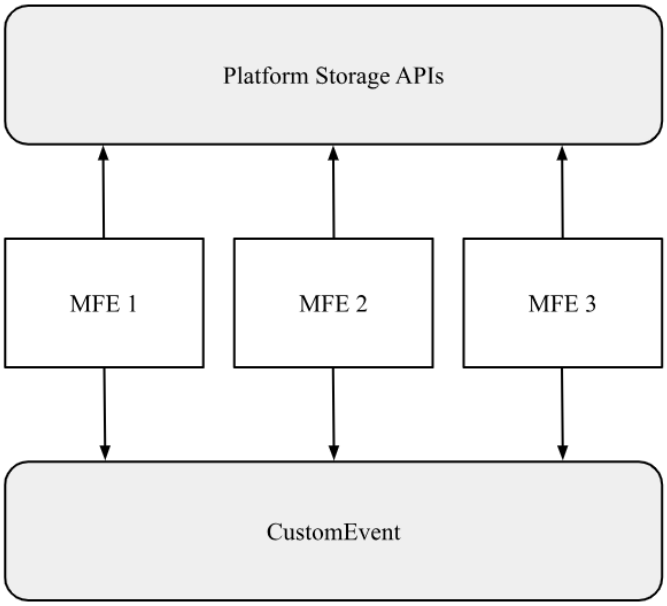

Рис. 1. Методи передачі даних між МІ сутностями

Ця гібридна стратегія комунікації дозволяє розробникам вибирати найбільш відповідний метод комунікації залежно від конкретних умов. Завдяки інтеграції як подієво-орієнтованих, так і платформних підходів збереження, ця модель може підвищити продуктивність, зменшити непотрібне перерендеринг стану і забезпечити ефективне управління даними в великих архітектурах МІ.

Порівняльні моделі різних архітектурних підходів

У швидко змінюваному ландшафті веб-розробки постійно з'являються нові технології та методології. Одним з найбільш помітних підходів у фронтенд-розробці сьогодні є концепція МІ. У цьому дослідженні буде проведено ґрунтовний аналіз різних стратегій впровадження МІ та представлено практичне порівняння на основі критичних показників продуктивності для фронтенд-додатків.

МІ представляють архітектурний підхід, який передбачає розбиття великого монолітного фронтенд-додатка на менші, більш керовані компоненти. Кожен компонент розробляється, впроваджується і, в деяких випадках, керується незалежними командами, що дозволяє досягти більшої гнучкості та масштабованості. Можна розглянути кілька підходів до впровадження МІ:

- МІ на основі iframe: Один із методів побудови масштабованих веб-додатків передбачає розбиття монолітного додатка на МІ, кожен з яких завантажується в основний додаток як iframe. Ця техніка дозволяє кожному МІ зберігати власне керування станом, маршрутизацією та іншими функціями, забезпечуючи певний ступінь незалежності в межах більшого додатка.

- Module Federation: Альтернативний і більш гнучкий підхід до побудови масштабованих веб-додатків передбачає використання Module Federation (табл. 1). Цей метод також передбачає розбиття великого монолітного додатка на менші, незалежно завантажувані модулі під час виконання. На відміну від iframe, Module Federation забезпечують більшу гнучкість та інтеграцію, оскільки кожен модуль може впроваджуватися незалежно, забезпечуючи безпечне розкриття свого інтерфейсу для використання іншими модулями.

- Vite — це сучасний інструмент для збірки фронтенду, розроблений для забезпечення швидкого процесу розробки. Завдяки використанню нативної підтримки ES-модулів у браузерах, Vite усуває необхідність у попередній збірці під час розробки. Він забезпечує миттєвий запуск сервера, надзвичайно швидку заміну модулів на льоту (HMR) та оптимізовані збірки для продакшену.

Результати досліджень

Для проведення дослідження створимо три прості веб додатки з інтерактивною кнопкою – лічильником базуючись на трьох різних підходах: Module Federation (React), Vite (React), Iframe. Обчислення результатів проведено за допомогою автоматичних тестів написаних на Python(Selenium) наведено у таблиці 1 та таблиці 2.

Таблиця 1.

Набір даних з порівняння продуктивності iframe і Module Federation

Метрики продуктивності	Iframe T1	Module Federation T2	Висновки (T1/T2), %
Час завантаження сторінки	0.020034 с	0.006509 с	308
Час для взаємодії	0.037650 с	0.022013 с	171
Час до першого байта	0.001999 с	0.001999 с	100
Використання пам'яті	47512114 байт	31535899 байт	151

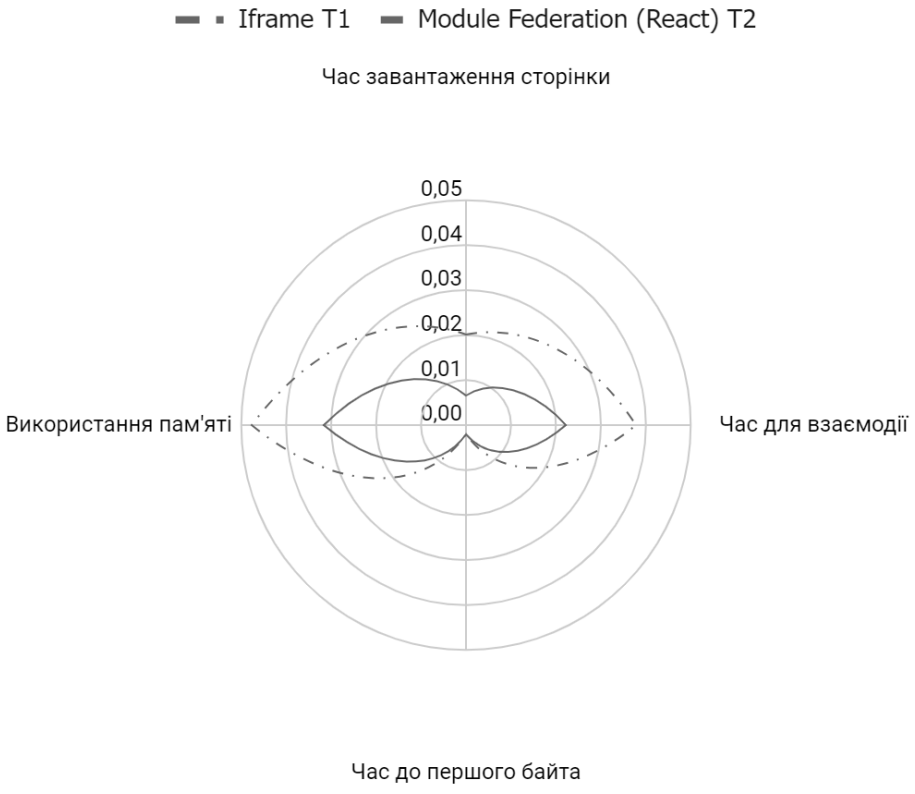

Рис. 2. Діаграма порівняння продуктивності iframe і Module Federation

Таблиця 2.

Набір даних з порівняння продуктивності iframe і Vite

Метрики продуктивності	Iframe T1	Vite T3	Висновки (T1/T3), %
Час завантаження сторінки	0,020034 с	0,01388 с	144
Час для взаємодії	0,03765 с	0,050056 с	75
Час до першого байта	0,001999 с	0,004669 с	43
Використання пам'яті	47512114 байт	4854963 байт	979

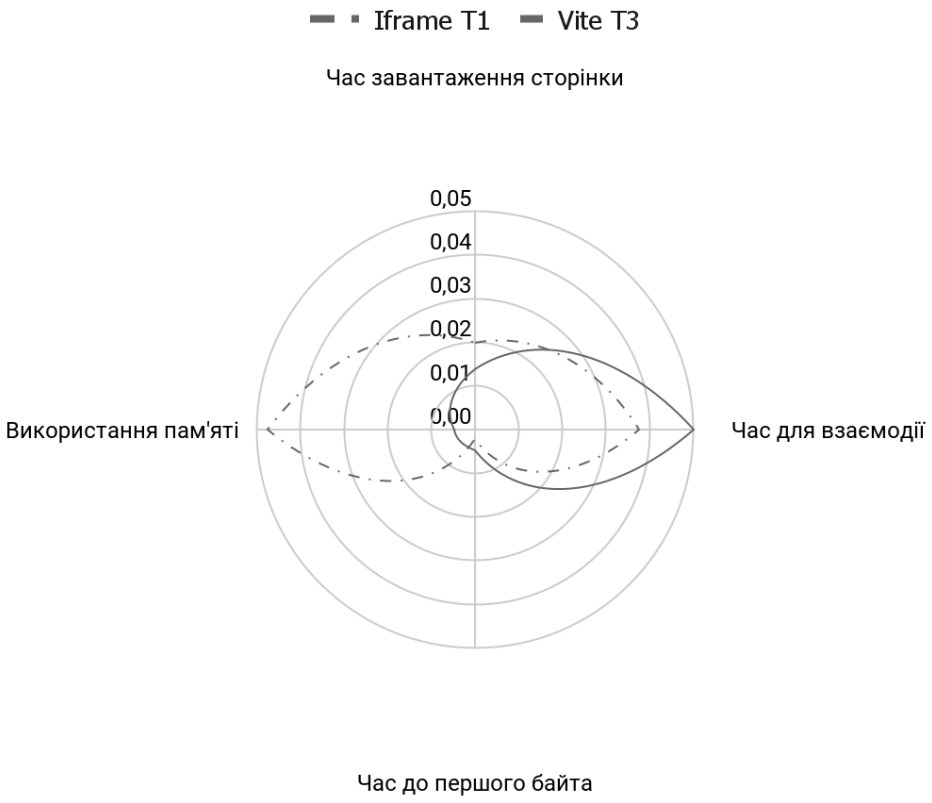

Рис. 3. Діаграма порівняння продуктивності iframe і Vite

В цілому, кожен підхід має свої унікальні переваги та недоліки в залежності від конкретного випадку використання. MI, побудовані за допомогою iframe, забезпечують повну ізоляцію, але можуть страждати від проблем із продуктивністю. Федеративні модулі забезпечують кращу інтеграцію та можливості спільного використання коду, але можуть вимагати більш масштабної початкової конфігурації та планування.

Найкраща загальна продуктивність (час завантаження сторінки, інтерактивність, використання пам'яті): Module Federation (рис. 2) є найбільш збалансованою та ефективною архітектурою. Вона пропонує найшвидший час завантаження, швидку інтерактивність і розумне використання пам'яті, що робить її ідеальним вибором для масштабованих і продуктивних додатків. Vite (табл. 2) відзначається економним використанням пам'яті, що робить його найкращим вибором для додатків, де критичне управління ресурсами. Однак він поступається в часі до першого байта і часу до інтерактивності, що може негативно вплинути на користувацький досвід. Iframe залишаються середнім рішенням, пропонуючи кращий час інтерактивності, ніж Vite (рис. 3), але поступаючись і в часі завантаження сторінки, і у використанні пам'яті в порівнянні з Module Federation.

5. Майбутні напрямки в архітектурі MI

Так само як мікросервіси революціонізували архітектури бекенду, MI пропонують модульний підхід до заміни монолітних фронтендів. Зростаюче впровадження цієї передової технології в останні роки є добре виправданим. Однак із подальшою еволюцією фронтенд-архітектур їхня зростаюча складність створює виклики для ефективного масштабування модульних веб-додатків.

WebAssembly виступає як ефективний інструмент для оптимізації модульних веб-додатків, особливо в поєднанні з MI. Його вроджена модульність гармонійно поєднується з MI та забезпечує високу продуктивність для ресурсомістких додатків. Однак, це інтегрування не обходиться без ускладнень.

Першим значним викликом є сумісність мов програмування. Модулі WebAssembly зазвичай пишуться на низькорівневих мовах, таких як C++ або Rust, тоді як MI складаються з JavaScript, HTML та CSS, які часто використовують JavaScript-фреймворки. Ці середовища відрізняються структурами даних, типовими системами та поведінкою під час виконання, що вимагає значних зусиль для забезпечення їх сумісності. Такі питання, як невідповідність типів даних і необхідність частої конверсії між WebAssembly і JavaScript, є поширеними. Цей процес конверсії створює затримки, що може призвести до проблем з продуктивністю та потребувати додаткового кодування.

Більше того, WebAssembly не має прямого доступу до моделі DOM у JavaScript, що змушує розробників шукати ефективніші стратегії взаємодії між модулями WebAssembly та компонентами JavaScript. На щастя, існує кілька підходів для зниження цих проблем. Наприклад, wasm-bindgen може автоматично генерувати допоміжний код для Rust і JavaScript. Використовуючи макрос #[wasm_bindgen] для анотації функцій Rust, розробники можуть експортувати їх у JavaScript, полегшуючи інтеграцію. Більш універсальним рішенням є WebAssembly System Interface (WASI), який стандартизує системні виклики і абстрагує загальні операції, такі як ввід-вивід файлів та мережеві операції. Хоча WASI не був розроблений спеціально для мікрофронтендів, він надає елегантний спосіб обробки системних операцій, підвищуючи портативність модулів.

Альтернативно, розробники можуть вирішити цю проблему з боку JavaScript, використовуючи ArrayBuffer або SharedArrayBuffer для безпосереднього доступу до пам'яті WebAssembly, мінімізуючи затримки через конверсію даних. Наприклад, створюючи типізований перегляд буфера. WebAssembly може безпосередньо записувати в цей загальний простір пам'яті, що дозволяє ефективний обмін даними без непотрібного копіювання.

Другий значний виклик — це керування станом. WebAssembly працює в ізольованому середовищі, що підвищує безпеку, але ускладнює спільне використання стану між модулями. Це питання стає ще більш вираженим, коли в одному проєкті використовуються різні JavaScript-фреймворки для різних мікрофронтендів. На відміну від JavaScript-додатків, які користуються централізованими рішеннями для керування станом, такими як Redux або Context API у React, модулі WebAssembly зазвичай керують станом внутрішньо. Це призводить до фрагментації даних стану та ускладнює синхронізацію між модулями. Крім того, це ускладнення ще більше погіршується через додаткові витрати, пов'язані з серіалізацією та десеріалізацією між двома окремими моделями пам'яті.

Для вирішення складнощів із керуванням станом у таких середовищах доцільно впровадити єдине глобальне сховище стану за допомогою інструментів, таких як Redux або MobX, що може забезпечити більш ефективну синхронізацію стану між мікрофронтендами та модулями WebAssembly.

Для досягнення ефективного керування станом при використанні MI архітектурі рекомендується інтегрувати кілька підходів. Зберігайте локальний стан у кожному MI для збереження модульності та незалежності. Впроваджуйте подієво-орієнтовану стратегію для керування спільним станом і комунікації між фронтендами. Крім того, використовуйте Module Federation для спільного використання загальних утиліт і логіки керування станом між MI.

Module Federation є оптимальним рішенням для великих додатків, що потребують високої продуктивності, балансу швидкості, інтерактивності та ефективного використання ресурсів. Vite є найкращим вибором для середовищ, де критичне управління пам'яттю, хоча його повільніший час до інтерактивності та час до першого байта повинні бути враховані. Iframe все ще можуть бути корисними для простіших додатків, де пріоритетом є негайна інтерактивність, але загалом вони поступаються як у продуктивності, так і в ефективності порівняно з іншими двома рішеннями.

Поєднання WebAssembly з персоналізацією, керованою штучним інтелектом, у архітектурі MI дозволить створити надзвичайно динамічну та чутливу систему, яка покращить користувацький досвід, одночасно підтримуючи продуктивність і безпеку.

6. Перспективи майбутніх досліджень

Розробка та впровадження МІ є актуальним завданням для сучасної індустрії програмного забезпечення, і майбутні дослідження можуть зосередитися на подоланні існуючих проблем, пов'язаних з масштабованістю, безпекою та інтеграцією з іншими технологіями.

Однією з перспективних напрямків є більш глибока інтеграція WebAssembly у МІ архітектури, особливо у поєднанні з штучним інтелектом (AI) та машинним навчанням (ML). Такі рішення можуть значно підвищити продуктивність додатків, зокрема для завдань із великою обчислювальною складністю, як-от обробка даних у реальному часі або аналітика на стороні клієнта. WebAssembly дозволить досягати більш високої швидкості виконання в порівнянні з традиційними веб-технологіями.

Іншим важливим напрямком досліджень є забезпечення безпеки у розподілених мікрофронтендних додатках. Оскільки кожен мікрофронтенд працює незалежно, питання автентифікації, авторизації та збереження конфіденційних даних є критично важливими для розробників. Майбутні дослідження повинні розробити стандартизовані підходи до захисту комунікацій між мікрофронтендами, особливо в контексті різних фреймворків і технологій, які можуть використовуватися у великомасштабних додатках.

Також варто приділити увагу питанням переносимості та інтероперабельності мікрофронтендів. У великих організаціях, де використовуються різні технологічні стеки для різних частин додатка, необхідно розробляти інструменти, що полегшують обмін даними та функціональністю між компонентами, написаними на різних фреймворках та мовах програмування.

Інноваційні дослідження у сфері AI та автоматизації розробки МІ також можуть покращити управління складністю та процесами інтеграції. Наприклад, системи на основі AI можуть автоматично пропонувати оптимальні стратегії для організації комунікацій між МІ або автоматизувати тестування та оптимізацію продуктивності різних компонентів додатка.

7. Висновки

МІ пропонують революційний підхід до побудови сучасних веб-додатків, дозволяючи розробникам розбивати великі монолітні системи на менші, більш керовані компоненти, які можна розробляти та впроваджувати незалежно. Проте впровадження цієї архітектури пов'язане з багатьма викликами, зокрема з питань міжфронтендної комунікації, продуктивності та керування станом.

У цій роботі було запропоновано кілька методів вирішення проблем комунікації між МІ, зокрема використання подієво-орієнтованих підходів та платформних інтерфейсів для збереження даних. Запропоновано гібридну модель комунікації, яка поєднує сильні сторони обох підходів для забезпечення більшої продуктивності та керованості додатків.

Крім того, порівняння різних архітектурних підходів, таких як iframe, Module Federation і Vite, показало, що федерація модулів є найбільш збалансованим і ефективним рішенням для великих масштабованих додатків, тоді як Vite відзначається кращим управлінням пам'яттю, але має певні недоліки щодо часу до першої взаємодії.

Інтеграція WebAssembly з МІ представляє великий потенціал для покращення продуктивності додатків, особливо в поєднанні з AI та ML для обробки складних завдань. Однак це інтегрування вимагає подолання викликів, пов'язаних з сумісністю мов і керуванням станом.

Майбутні дослідження у цій галузі повинні бути спрямовані на вирішення проблем безпеки, масштабованості та перенесення мікрофронтендів між різними технологічними стеками, а також на пошук нових шляхів інтеграції з новими технологіями.

Список літератури

*1. Blinowski, G., Ojdowska, A., & Przybylek, A. (2022). Monolithic vs. Microservice Architecture: A performance and scalability evaluation. *IEEE Access*, 10, 20357–20374. https://doi.org/10.1109/access.2022.3152803*

*2. Terdal, S. (2022). Microservices enabled e-commerce web application. *International Journal of Research in Applied Science and Engineering Technology*, 10(7), 3548–3555. https://doi.org/10.22214/ijraset.2022.45791*

*3. Zhou, J., Yang, L., & Wu, J. (2023). Micro-frontend architecture base. In *Proceedings of the Sixth International Conference on Computer Information Science and Application Technology (CISAT)*. https://doi.org/10.1117/12.3003818*

*4. Pontarolli, R. P., Bigheti, J. A., de Sá, L. B. R., & Godoy, E. P. L. (2023). Microservice-Oriented Architecture for Industry 4.0. *Engineering*, 4, 1179–1197. https://doi.org/10.3390/eng4020069*

*5. Perlin, R., Ebling, D., Maran, V., Descovi, G., & Machado, A. (2023). An approach to follow microservices principles in frontend. In *Proceedings of the IEEE 17th International Conference on Application Information and Communication Technology (AICT)*. https://doi.org/10.1109/aict59525.2023.10313208*

*6. Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From Monolithic Systems to Microservices: An assessment framework. *Information and Software Technology*, 137, 106600. https://doi.org/10.1016/j.infsof.2021.106600*

*7. Homay, A., Zoitl, A., de Sousa, M., & Wollschlaeger, M. (2019). A survey: Microservices Architecture in Advanced Manufacturing Systems. In *Proceedings of the IEEE 17th International Conference on Industrial Informatics (INDIN)*. https://doi.org/10.1109/indin41052.2019.8972079*

*8. Marco, V., & Farias, K. (2024). Exploring the technologies and architectures used to develop micro-frontend applications: A systematic mapping and emerging perspectives. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.475066*

*9. Abdellatif, M., Shatnawi, A., Mili, H., Moha, N., Boussaidi, G. E., Hecht, G., Privat, J., & Guéhéneuc, Y.-G. (2021). A taxonomy of Service Identification Approaches for Legacy Software Systems Modernization. *Journal of Systems and Software*, 173, 110868. https://doi.org/10.1016/j.jss.2020.110868*

*10. Nikulina, O., & Khatsko, K. (2023). Method of converting the monolithic architecture of a front-end application to microfrontends. *Bulletin of National Technical University KhPI Series System Analysis Control Information Technologies*, 2(10), 79–84. https://doi.org/10.20998/2079-0023.2023.02.12.*

*11. Stepanov, O., & Klym, H. (2024). Features of the implementation of micro-interfaces in information systems. *Advances in Cyber-Physical Systems (ACPS)*, 9(1), 54–60. https://doi.org/10.23939/acps2024.01.054*

METHODOLOGY OF IMPLEMENTATION OF INFORMATION SYSTEMS USING MICRO INTERFACES TO INCREASE THE QUALITY AND SPEED OF THEIR DEVELOPMENT

O.V. Stepanov, H.I. Klym

Lviv Polytechnic National University
Department of Specialized Computer Systems
E-mail: oleksandr.v.stepanov@lpnu.ua, halyna.i.klym@lpnu.ua

Abstract: Microservices represent a software development approach, a variation of service-oriented architecture, that structures an application as a collection of loosely connected services. The aim of this work is to explore the design and implementation methodology for information systems using micro-interfaces to enhance development quality and speed while simplifying their usage. This work proposes a method for transitioning from a monolithic software architecture to a microservice architecture. A brief review of existing research on architecture reengineering is provided, and the advantages of adopting a microservice approach are highlighted. An experiment with a typical external single-page application illustrates the comparison of the effectiveness and performance of the proposed architectures. The potential future direction of micro-frontend architecture development in combination with WebAssembly is examined, and the advantages and disadvantages of this technology integration are analyzed.

Keywords: architecture, interface, micro-interface, microservices, monolithic structure, software applications, WebAssembly.