DOI 10.26886/2414-634X.8(52)2021.1

UDC: 004.514

JUSTIFICATION OF MICROSERVICE APPROACH TO WEB DEVELOPMENT

Yura Abharian, Specialist of the Department Transport Technologies

https://orcid.org/0000-0001-8519-2539

e-mail: yuraabharian@gmail.com

Zaporizhia National Technical University, Software Engineer SoftServe, Zaporozhye, Ukraine

The article substantiates the microservice approach to web development. It is emphasized that the main principle of microservice architecture is to create a program by dividing the components of business logic into small services that can be deployed and operated independently, tasks are distributed between services, which are separated by so-called service boundaries. It is substantiated that the boundaries of services are closely related to business requirements and the boundaries of the organizational structure. Individual services can be linked to individual teams, budgets and action plans.

Key words: website, server, client, microservice, interface, Microfrontend, architecture.

спеціаліст, розробник програмного забезпечення, Абгарян Юра Серьожайович. Обгрунтування мікросервісного підходу до веб-розробки / Запорізький національний технічний університет, SoftServe Україна, Україна, Запоріжжя

У статті обґрунтовано мікросервісний підхід до веб-розробки. Підкреслено, що основним принципом мікросервісної архітектури є створення програми шляхом поділу компонентів бізнес-логіки на невеликі сервіси, які можуть розгортатися та експлуатуватися

незалежно один від одного, завдання розподіляються між сервісами, які розділені між собою так званими межами сервісів. Обґрунтовано, що кордони сервісів тісно пов'язані з бізнес-вимогами та межами організаційної структури. Індивідуальні послуги можуть бути пов'язані з окремими командами, бюджетами та планами дій. У якості прикладів меж сервісів зазначено, що можна навести послуги обробки платежів та аутентифікації користувачів.

Ключові слова: веб-сайт, сервер, клієнт, мікросервіс, інтерфейс, Microfrontend, архітектура.

Вступ. Стандартні веб-додатки, за своєю структурою, традиційно поділяються на дві частини: бек-енд (сервер), який у більшості випадків відповідає за обробку даних, та інтерфейс (клієнт), який необхідний для забезпечення зручного інтерфейсу взаємодії між користувачами та системами.

Внутрішні компоненти можуть бути розроблені з використанням різних архітектурних підходів, одним з яких виступають мікросервіси, що найбільше підходить для масштабованих систем. Ідея підходу базується на концепції відокремлення логічно незалежних частин системи. Він використовується замість монолітних архітектурних стилів, тому що їх важко масштабувати (і неможливо масштабувати лише окремі його частини) і розвивати різні його частини одночасно, тоді як ці питання вирішуються в мікросервісах.Якщо говорити про світ клієнтських додатків, то їх розмір, а також складність зростає (особливо у веб-розробці). Тому архітектура починає відігравати все більшу роль. Спільнота Фрондтенд прийшла до ідеї Microfrontend, яка схожа на концепцію мікросервісів і передбачає поділ однієї великої програми на менші частини, що призводить до можливості одночасної розробки, швидшого завантаження та більшої продуктивності.

Аналіз останніх досліджень і публікацій. Публікації стосовно застосування мікросервісного підходу до веб-розробки є популярним напрямком досліджень на протязі останніх 30-ти років.

К. Р. Колос, А. І. Баранов та Р. В. Петросян [1] провели аналіз побудов клієнтських частин веб-додатків на основі Microfrontend підходу. Автори наголошують, що перед переходом до Microfrontend архітектури необхідно враховувати: наявність ресурсів для достатнього рівня автоматизації та забезпечення управління додатковою необхідною інфраструктурою; зміни в процесі розробки, тестування та випуску у великій кількості компонентів; зростання складності, пов'язаної з використанням більшої кількості інструментів та підходів до розробки; забезпечення достатнього рівня якості, узгодженості та управління великою кількістю кодових баз. Отже, обираючи архітектуру Microfrontend слід зважити і проаналізувати наявність технічної та організаційної доцільності для прийняття такого підходу. Низка авторів О. Н. Котенко, Т. О. Жирова, В. І. Чубаєвський та А. М. Десятко описали основні складові front-end розробки, здійснили детальний огляд текстового редактора Sublime Text, як одного з найпопулярніших текстових редакторів з широким набором зручних інструментів для виділення, маркування та обробки текстових фрагментів коду.

Із зарубіжних авторів варто відзначити такі роботи як: M.Baumann, Nascimento, Charlles & Sotto, Eder, Newman S., Malavalli, Divyanand & Sathappan, Sivakumar, A. Kumar, Yang, Caifang & Liu, Chuanchang & Su, Zhiyuan, Kuepper R., Bartkov M., Peltonen, Severi & Mezzalira, Luca & Taibi, Davide, Peltonen, Severi & Mezzalira, Luca & Taibi, Davide , Noppadol, Nattaporn & Limpiyakorn, Yachai, Pölöskei, István & Bub, Udo, Yang, Caifang & Liu, Chuanchang & Su, Zhiyuan та інші.

Проте, враховуючи описані наукові набутки, за темою, питання обґрунтування мікросервісного підходу до веб-розробки залишається відкритим та потребує детального опрацювання.

Постановка завдання. Здійснити обґрунтування мікросервісного підходу до веб-розробки. Описати підхід Microfrontend у контексті односторінкових додатків. Розкрити принцип застосування архітектури мікросервісів у проектуванні та розробці інтерфейсу веб-додатку.

Викладення основного матеріалу дослідження. На сьогодні, в умовах стрімкого розвитку інформаційних технологій, ринок програмного забезпечення, інструментів та підходів до проектування стрімко зростає. Розробник має можливість обрати інструментальну складову на свій розсуд.

Масштабність альтернативних варіантів розробки інтерфейсних програм вражає, розглянемо найбільш актуальні з них. Так, JAMstack означає JavaScript, API і Markup, і є найбільш популярною філософією веб-розробки, яка спрямована на прискорення як процесу веб-розробки, так і часу завантаження веб-сторінки [16].

JAMstack – це архітектура для створення сучасних веб-сторінок і програм на основі передових інструментів і робочих процесів для швидшої, простішої та безпечнішої мережі. Дана архітектура забезпечує швидкість і простоту попередньо відтворених статичних сайтів із динамічними можливостями за допомогою JavaScript, API та без серверних функцій. Ці попередньо відтворені сайти розгортаються безпосередньо в мережі доставки контенту (МДК) без необхідності керувати, масштабувати або виправляти будь-які веб-сервери. Для JAMstack взагалі немає серверного середовища. Тому програми JAMstack є менш дорогими, оскільки розміщення статичних файлів дешеве. Це також означає, що розробники інтерфейсу можуть зосередитися лише на розробці та налагодженні інтерфейсу, що

зазвичай означає більш цілеспрямований підхід до кінцевого результату.

На рис. 1 показані основні відмінності між традиційним цілісним веб-додатком і додатком на основі архітектури JAMstack. У цій роботі багато дискусій про руйнування моноліту, тобто розбиття інтерфейсного додатку на менші частини, як у випадку з MicroFrontends. Щодо архітектури JAMstack, з іншого боку, вона більше зосереджена на повному відокремленні інтерфейсу та бек-енду один від одного.

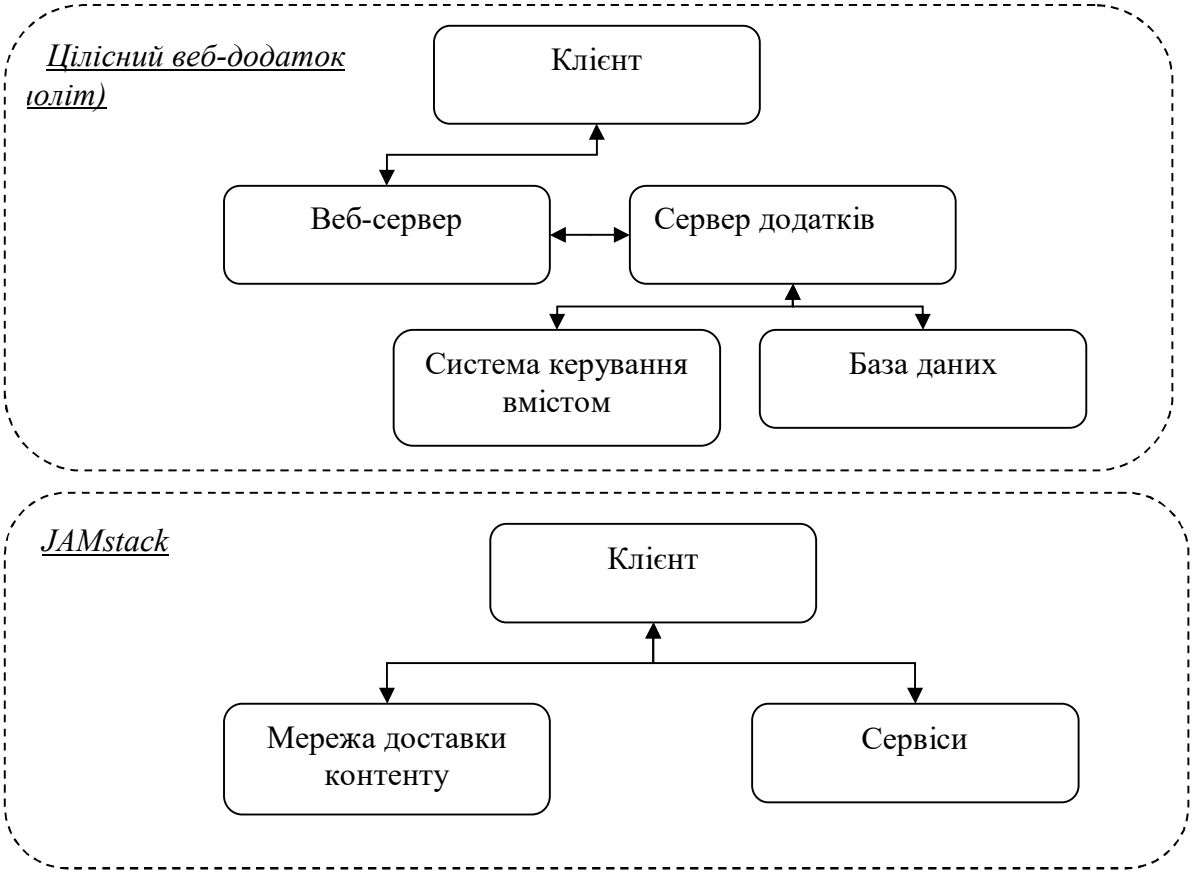

Рис. 1. Основні відмінності між традиційним цілісним веб-додатком і додатком на основі архітектури JAMstack

Фронтенд-додаток також може бути клієнтським, де клієнт (браузер) отримує файли HTML і JavaScript, які управляють HTML-документом і DOM. Це призводить до створення інтерактивних програм, у яких взаємодія користувача призводить до анімації або

зміни графічного інтерфейсу, без необхідності оновлення сторінки. Щоб полегшити це, браузер надає API, який називається Document Object Model (DOM), який дозволяє мовам сценаріїв отримувати доступ і управляти HTML-документами.

У односторінкових програмах (Single-page Applications (SPA)) у браузер завантажується лише один файл HTML, сторінку не потрібно оновлювати, коли користувач взаємодіє зі сторінкою, це дає користувачеві більш плавний користувальницький досвід. Щоб мати можливість змінювати вміст сторінки та надавати додаткову інформацію для користувача, DOM необхідно динамічно оновлювати за допомогою JavaScript та HTTP-запитів для отримання інформації від сервера [16]. Існує велика кількість фреймворків, бібліотек та інструментів, наприклад, Angular, React і Vue, які призначені для створення шару абстракції для процесу маніпулювання DOM. Сутність реалізації фреймворків односторінкових програм наведено на рис. 2.

Оскільки один HTML-файл завантажується браузером, HTML забезпечує точку вкорінення для програми JavaScript, яка також завантажується разом із зображенням, CSS, файлами скриптів та іншими зовнішніми ресурсами [14]. Точка вкорінення – це в основному один елемент HTML, найчастіше це блоковий елемент. Коли для програми JavaScript надається точка кореневої системи, програми знають, з чого почати компіляцію вмісту HTML у документу. Усі фреймворки, розроблені для створення односторінкових додатків, мають методи життєвого циклу програми зі стандартизованими іменами, які дають розробникам можливість визначати, що і коли буде змінено в DOM протягом життєвого циклу програми, це одна з переваг SPA. Таким чином, існує метод, який визначає, перебіг подій, коли компонент програми буде відображатися для користувача, і інший метод, який визначає, перебіг подій, коли компонент програми буде

видалено з подання. Оскільки SPA уникає множинних мережевих викликів для завантаження додаткової логіки програми та миттєво відтворює правильний вміст протягом життєвого циклу програми, покращується користувальницький досвід, і програма може моделювати додатки.

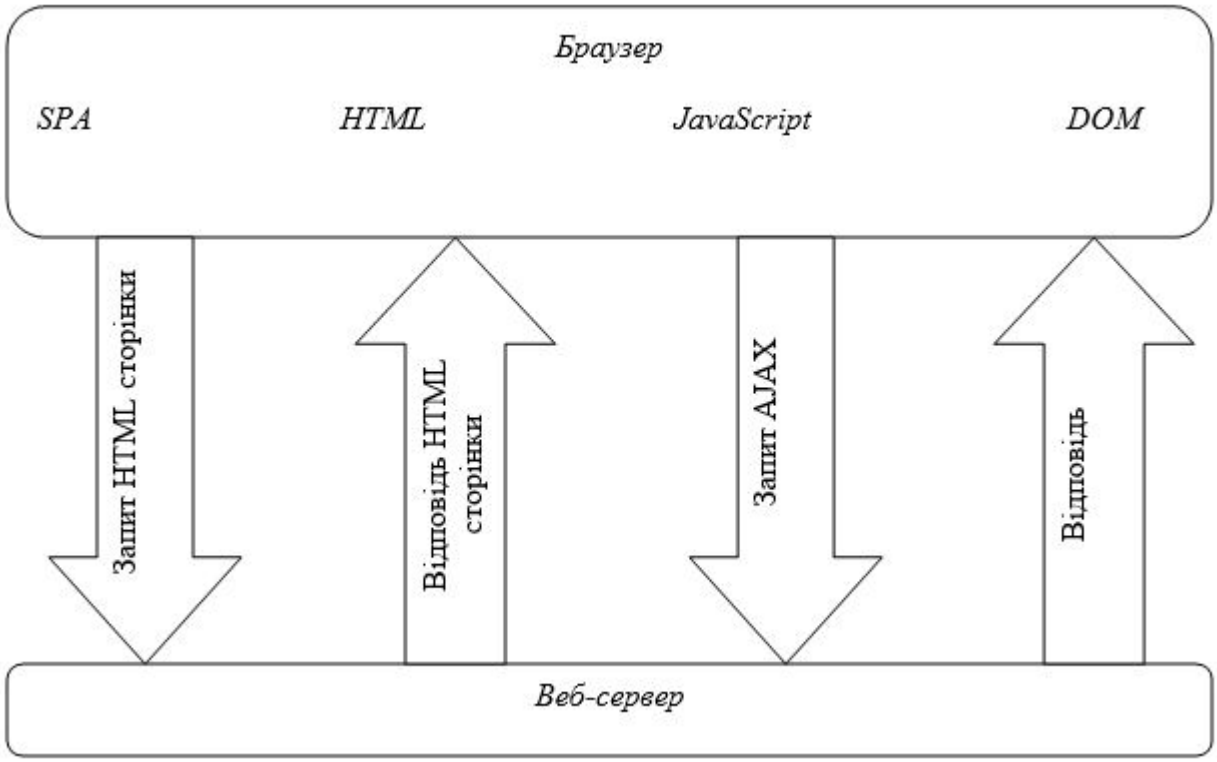

Рис. 2. Сутність реалізації фреймворків односторінкових програм

Незважаючи на те, що сьогодні SPA є популярним способом створення веб-додатків, він має деякі недоліки для певного типу застосувань. Початковий додаток завантажується під час запуску програми, і це зазвичай займає більше часу, ніж з іншими архітектурами, оскільки всю програму потрібно завантажити до браузера, а не лише те, що користувач має бачити в цей момент.

Різні методи інтерфейсних програм можна змішувати, а різні частини HTML-документа можна відтворювати за допомогою методів на стороні клієнта або сервера. Основна ідея ізоморфних веб-додатків, або універсальних додатків, полягає в написанні програм JavaScript,

розроблених для веб-браузера, але в той же час програма повинна працювати на сервері для створення файлів розмітки HTML. Таким чином, код між сервером і клієнтом є спільним і може виконуватися в обох контекстах. Зазначений підхід приносить певні переваги при правильному використанні. Це особливо зручно, коли час взаємодії, A/B-тестування та SEO є важливими характеристиками для кінцевої програми. Ізоморфний додаток може бути розроблений різними способами, але головна концепція полягає в тому, що веб-сторінка відображається двічі одним і тим же додатком. Оскільки веб-додаток ділиться кодом між клієнтом і сервером, сервер, наприклад, може виконувати частину візуалізації для сторінки, яку запитує браузер, отримувати дані для відображення з бази даних або з одного або кількох API, компілювати дані разом і потім попередньо відтворить його за допомогою системи шаблонів, яка використовується для створення представлення, щоб надати клієнту сторінку, яка не потребує додаткових мережевих викликів для запиту додаткових даних для відображення. Перевага порівняно з традиційним SPA полягає в тому, що веб-сторінка завантажується швидше, оскільки початковий файл містить весь необхідний HTML для відображення сторінки. Хоча з ізоморфною програмою час взаємодії більший, оскільки користувач побачить графічні елементи на екрані, які здаються інтерактивними, але не є такими.

Оскільки універсальна програма використовує як інтерфейс, так і бек-енд для створення представлень для користувача, ці програми можуть постраждати від проблем із масштабованістю, якщо веб-сторінку відвідують мільйони користувачів, оскільки програма може попередньо відтворювати HTML-сторінки на сервері. Microfrontends також може страждати від цієї проблеми, якщо використовується композиція на стороні сервера, оскільки на сервері потрібно з’єднати

багато різних мікропрограм. Зрештою, все виходить у вигляді HTML на інтерфейсі. Веб-сторінки написані на HTML, декларативній мові веб-програмування, яка вказує веб-браузерам, як структурувати та представляти вміст на веб-сторінці. Іншими словами, HTML забезпечує основні будівельні блоки для Інтернету. І довгий час ці будівельні блоки були досить простими і статичними: рядки тексту, посилання та зображення. HTML-сайт все ще є досить поширеним і може бути правильним рішенням для невеликого сайту або цільової сторінки для запуску бізнесу. Статичний HTML-сайт не призначений для довготривалого використання або, принаймні, розробник не може створити на його основі повноцінний додаток. Працюючи з інтерфейсом програмного забезпечення, розробники програм і архітектори програмного забезпечення мають кілька варіантів архітектури на вибір, наприклад, односторінкові програми, SPA, коротше кажучи, програми для відтворення на стороні сервера або програми, складені зі статичних файлів HTML. Згодом ці архітектури можуть призвести до того, що проект стане єдиним цілим. Це збільшує складність інтерфейсної програми, і внесення змін до частини системи може мати непотрібні або небажані наслідки для інших частин. Бази коду стають величезними, програма має багато залежностей і стає тісно пов’язаною, координація між командами розробників стає все складнішою і повільнішою, що призводить до закону спадної віддачі. Збільшення кількості розробників у фронтенд-командах не вплине на швидкість виробництва, оскільки обрана архітектура встановлює межі для розробників.

Microfrontends розширює вплив мікросервісів на програму. Він перетворює цілісні веб-додатки з архітектури програми на основі одного коду в програму, яка об’єднує кілька невеликих інтерфейсних програм в одне ціле. Кожен з цих незалежних додатків може

запускатися, розроблятися та розгортатися незалежно. Можливість незалежної розробки та розгортання дозволяє командам розробників створювати ізольовані та слабо пов'язані служби. Ідея Microfrontends полягає в тому, щоб обробляти веб-додаток як комбінацію функцій або бізнес-піддоменів. Кожна команда повинна мати лише один домен для обробки.

Цілісні структури інтерфейсу вводять горизонтальні шари до програми, але Microfrontends прагне розділити додаток по вертикалі, як показано на рис. 3.

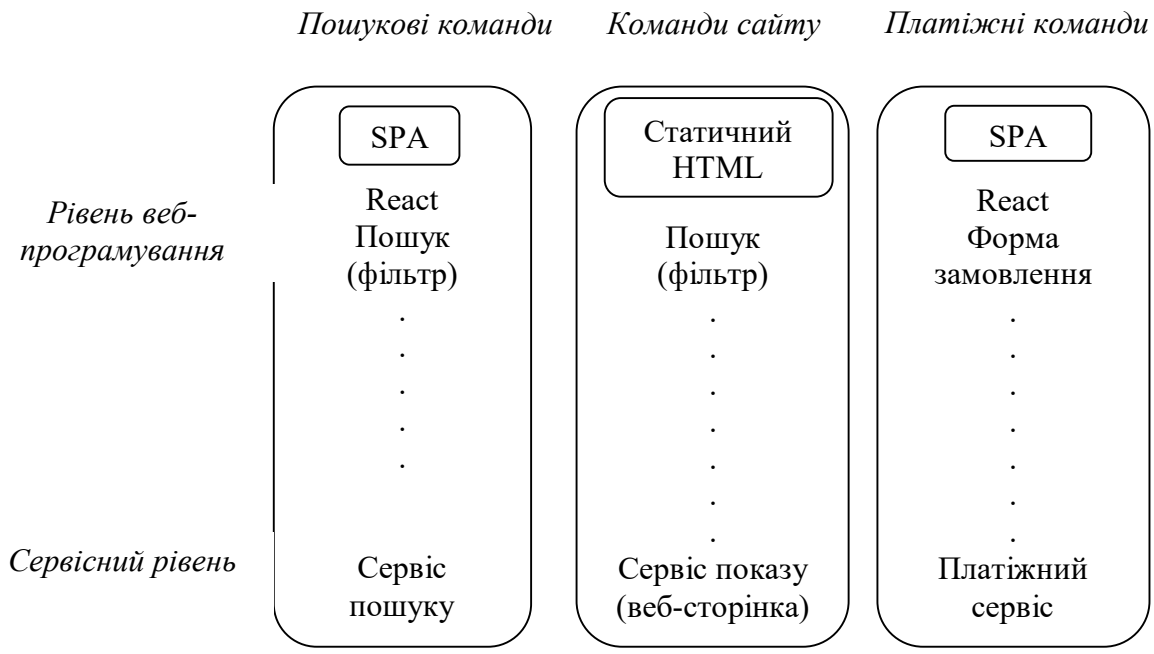

Рис. 3.Архітектура Microfrontends

Кожен з цих вертикальних фрагментів обслуговує певний бізнес-домен або функцію і повністю побудований від низу до верху. Завдяки Microfrontends кожна команда розробників має технологічно-агностичний підхід і вирішує, який стек технологій використовувати при розробці. Команди можуть оновлювати або навіть змінювати стек, не співпрацюючи з іншими командами.

З архітектурою Microfrontends деякі архітектурні рішення повинні бути прийняті заздалегідь, оскільки ці рішення будуть формувати

майбутні кроки, які будуть прийняті разом з проектом. Щоб визначити Microfrontends, ключове рішення, яке необхідно прийняти, - це визначити, як розглядати Microfrontends з технічної точки зору. Для цього є два варіанти: горизонтальне розділення: кілька мікро-фронтендів на сторінці; вертикальний розподіл: один мікро-фронтед за раз. При горизонтальному розподілі кілька менших додатків завантажуються на одну сторінку, і це вимагає, одночасної координації кількох команд, оскільки кожна команда відповідає за частину представлення.

У сценарії вертикального розділення кожна команда відповідає за бізнес-домен, наприклад, досвід автентифікації або оплати. При вертикальному розподілі буде застосовуватися дизайн, керований доменом (DDD). У разі горизонтального розділення важливим кроком є визначення того, як Microfrontends взаємодіють один з одним. Однин із методів це використання активатора подій, введеного у кожен мікро-інтерфейс. Це робить кожен Microfrontend повністю незалежним. Коли Microfrontend випускає подію, інші Microfrontends, підписані на цю конкретну подію, реагують належним чином.

У разі вертикального розділення важливо розуміти, як обмінюватися інформацією між мікро-інтерфейсами. Як для горизонтального, так і для вертикального підходів необхідно підходити до питань взаємодії у разі змін. Цілком можливо, що змінні можуть передаватися через рядок запиту або за допомогою URL-адреси для передачі невеликої кількості даних (і змушуючи нове представлення отримати деяку інформацію з сервера). Крім того, можна використовувати веб-сховище для тимчасового (сховище сеансів) або постійного (локальне сховище) зберігання інформації. Мікросервісний підхід до веб-розробки ґрунтується на трьох варіантах: композиція на стороні клієнта; бокова композиція; композиція на стороні сервера. У

композиції на стороні клієнта оболонка програми завантажує Microfrontends всередині себе. Microfrontends повинен мати як точку входу файл JavaScript або HTML, щоб оболонка програми могла динамічно додавати вузли DOM у випадку файлу HTML або ініціалізувати програму JavaScript, коли точкою входу є файл JavaScript. Іншим можливим підходом є використання механізму трансклюзії, який можна використовувати на стороні клієнта за допомогою техніки включення на стороні клієнта. Саме тут оболонка програми завантажує компоненти всередині контейнера, використовуючи тег-заповнювач, і аналізує всі заповнювачі, замінюючи їх відповідним компонентом. Такий підхід надає безліч варіантів.

Висновки. У роботі обґрунтовано мікросервісний підхід до веб-розробки. Визначення перспективності використання Microfrontends підходу до поєднання віджетів або сторінок, реалізованих різними командами з використанням різноманітних фреймворків, є беззаперечним, за рахунок таких переваг як: модульна архітектура, швидкість тестування, паралельні деплойменти. Однак мікросервісний підхід до веб-розробки вимагає масштабованості, чим більше проект застосування з розподіленими командами тим більше переваг застосування.

Перспективним напрямком подальших досліджень у даній сфері є розробка веб-додатку з використанням Microfrontends підходу та порівняння його за швидкодією з монолітним аналогом.

References:

1. Kotenko, N., Zhyrova, T., Chybaievskyi, V., & Desiatko, A. (2019). Research of main trends of modern web sites development. *Cybersecurity: Education, Science, Technique*, (5), 6–15. https://doi.org/10.28925/2663-4023.2019.5.615

2. Zelenyuk, O. (2019). Web design in the context of the formation of a visual culture of the virtual environment. *Young Scientist*, *1*(65), 23–26. https://doi.org/10.32839/2304-5809/2019-1-65-6
3. Baumann, M. (2019). Micro-Frontends: Studienarbeit. *HSR Hochschule fur Technik Rapperswil*.
4. Nascimento, C. P., & Sotto, E. C. S. (2020). MICROFRONTEND. *Revista Interface Tecnológica*, *17*(1), 153–165. https://doi.org/10.31510/infa.v17i1.798
5. Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. USA : O'Reilly Media.
6. Malavalli, D., & Sathappan, S. (2015). Scalable microservice based architecture for enabling DMTF profiles. In *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE. https://doi.org/10.1109/cnsm.2015.7367395
7. Kumar, A. (2019). *Micro Frontends Architecture: Introduction, Design, Techniques & Technology*. USA : KdpPrint Us.
8. 11. Yang, C., Liu, C., & Su, Z. (2019). Research and Application of Micro Frontends. *IOP Conference Series: Materials Science and Engineering*, *490*, 062082. https://doi.org/10.1088/1757-899x/490/6/062082
9. Nelson T., Kuepper R. (2020). *Hands-On Swift 5 Microservices Development: Build Microservices for Mobile and Web Applications Using Swift 5 and Vapor 4*. Packt Publishing, Limited.
10. Bartkov, M. (2021). Graal as a multilingual platform. *Innovative Solutions in Modern Science,* 3(47). doi: 10.26886/2414-634X.3(47)2021.9 (date of access: 23.12.2021).
11. Peltonen, S., Mezzalira, L., & Taibi, D. (2021). Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature

Review. *Information and Software Technology*, *136*, 106571. https://doi.org/10.1016/j.infsof.2021.106571

12. Nguyen, T.-P. N., & Tran, N.-T. (2021). Contour: Penalty and Spotlight Mask for Abstractive Summarization. In *Recent Challenges in Intelligent Information and Database Systems* (pp. 174–187). Springer Singapore. https://doi.org/10.1007/978-981-16-1685-3_15
13. Search Engine Optimization. (2021). *Pro Web 2.0 Application Development with GWT*. Berkeley, CA, 333–355. URL: https://doi.org/10.1007/978-1-4302-0637-8_12 (date of access: 23.12.2021).
14. Pölöskei, I., & Bub, U. (2021). Enterprise-Level Migration to Micro Frontends in a Multi-Vendor Environment. *Acta Polytechnica Hungarica*, *18*(8), 7–25. https://doi.org/10.12700/aph.18.8.2021.8.1
15. Yang, C., Liu, C., & Su, Z. (2019). Research and Application of Micro Frontends. *IOP Conference Series: Materials Science and Engineering*, *490*, 062082. https://doi.org/10.1088/1757-899x/490/6/062082
16. Cardoza, C. (2020). *Jamstack brings front-end development back into focus*. 2020. https://sdtimes.com/webdev/jamstack-brings-front-end-development-back-into-focus/ (date of access: 23.12.2021).

Citation: Yura Abharian (2021). JUSTIFICATION OF MICROSERVICE APPROACH TO WEB DEVELOPMENT. New York. TK Meganom LLC. Innovative Solutions in Modern Science. 8(52). doi: 10.26886/2414-634X.8(52)2021.1