НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК 004.42

До захисту допущено:
В.О. Завідувача кафедри
Михайло НОВОТАРСЬКИЙ
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Мікрофронтенди: архітектурні підходи та їх вплив на масштабованість веб-застосунків»

Виконав:
студент ІІ курсу, групи ІО-41мп
Грицюк Максим Юрійович _____

Керівник:
к.т.н., доцент
Верба Олександр Андрійович _____

Консультант з нормоконтролю:
проф., д.т.н.
Кулаков Юрій Олексійович _____

Рецензент:
к.т.н., доцент
Шимкович Володимир Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Освітньо-кваліфікаційний ступінь *магістр*
(назва ОКР)

Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Спеціалізація *123. Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
В.О. Завідувача кафедри
Михайло НОВОТАРСЬКИЙ
(підпис) (ініціали, прізвище)
« » 2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Грицюку Максиму Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Мікрофронтенди: архітектурні підходи та їх вплив на масштабованість веб-застосунків»
Науковий керівник дисертації Верба Олександр Андрійович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «06» 11 2025 р. № 4841-с

2. Строк подання студентом дисертації 01.12.2025

3. Об'єкт дослідження робота та продуктивність веб-додатків на архітектурі мікрофронтедів проти стандатного монолітного підходу

4. Предмет дослідження методи та програмні засоби розробки веб-додатків використовуючи архітектуру мікрофронтендів.

5. Перелік завдань, які потрібно розробити:

1. Огляд теоретичних основ мікрофронтендів.
2. Аналіз архітектурних підходів до розробки мікрофронтендів.
3. Інструменти та технології впровадження мікрофронтендів, проєктування та реалізація застосунку на основі мікрофронтендів.

4. Оцінювання масштабованості та ефективності.
5. Розроблення стартап-проєкту.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	проф. каф. ОТ д.т.н., Кулаков Ю.О.		
2	проф. каф. ОТ д.т.н., Кулаков Ю.О.		
3	проф. каф. ОТ д.т.н., Кулаков Ю.О.		
4	проф. каф. ОТ д.т.н., Кулаков Ю.О.		

7. Дата видачі завдання 01 вересня 2025

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	Затвердження теми роботи	01.09.25 – 07.09.25	
2.	Розроблення структури системи, розроблення графічних елементів	08.09.25 – 21.09.25	
3.	Програмна реалізація системи. Тестування системи	22.09.25 – 02.10.25	
4.	Підготовка матеріалів першого, другого та третього розділів дипломного проєкту	03.10.25 – 07.11.25	
5.	Тестування функціональності та оцінювання ефективності роботи системи	08.11.25 – 14.11.25	
6.	Підготовка та оформлення матеріалів магістерської дисертації	15.11.25 – 30.11.25	

Студент _____ Максим ГРИЦЮК
(підпис)

Науковий керівник дисертації _____ Олександр ВЕРБА
(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Мікрофронтенди: архітектурні підходи та їх вплив на масштабованість веб-застосунків»

студентом: Грицюком Максимом

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 115 аркушів основного тексту, 50 ілюстрації, 31 таблиця. При підготовці використовувалася література з 38 джерел.

Актуальність теми. Сьогодення диктує нові правила у процесі розвитку веб-застосунків. Розробка сучасних веб-систем потребує нових підходів, оскільки змінюються вимоги складності, масштабування та продуктивності. Варто зазначити, що традиційний тип архітектури фронтенду – монолітний, уже не задовольняє виконання всіх поставлених вимог до застосунків, які включають швидке масштабування та підтримку роботи системи завдяки паралельній роботі великих команд. Оскільки веб-застосунки розробляються для значної кількості користувачів, то розробники мусять постійно додавати нові функціональні потреби, а отже потребують кращої архітектури системи.

Описані умови заставляють задуматись над тим, що концепція «мікрофронтенд» – це достатньо хороше та сучасне рішення для застосування мікросервісної архітектури на рівні фронтеду (інтерфейсному). Даний підхід дозволить забезпечити поділ великої фронтенд-частини на незалежні сервіси, з якими можна буде працювати окремими командами розробників.

Ефективність дослідження за цією темою є доведена, оскільки провадження такого архітектурного шаблону є затребуваною для забезпечення розробки великих, гнучких, масштабованих та технологічно продуманих веб-застосунків, а також для прискореного випуску та подальшої надійної підтримки системи.

Мета і завдання дослідження. Метою цього дослідження є аналіз підходу проєктування та реалізація застосунку на основі мікрофронтендів, архітектура якої буде забезпечена за рахунок таких технологій як Vue, React, а також платформи Single-SPA. Завданням дослідження є не тільки розробки демонстраційного веб-застосунку, а також оцінка його ефективності.

Для досягнення поставленої мети потрібно виконати ряд завдань:

- проаналізувати еволюційну складову в архітектурних підходах до розробки фронтенд складової;
- дослідити поняття «монолітний» фронтенд та його обмеження;
- розглянути концепцію мікросервісів та їх відмінність від мікрофронтенду;
- дослідити основні підходи (архітектурні шаблони) для побудови мікрофронтендів;
- проаналізувати інструменти, які забезпечують реалізацію та провадження дослідженого архітектурного підходу;
- спроєктувати, розробити та дослідити демонстраційний веб-застосунок за допомогою сучасних фреймворків Vue та React;
- оцінити ефективність проведеного дослідження;
- розробити стартап проєкт.

Об’єкт дослідження – архітектура мікрофронтендів та її вплив на масштабованість веб-застосунків.

Предмет дослідження – архітектурні підходи при розробці мікрофрондендів, а також інструменти та технології їх впровадження.

Методи дослідження. Досягнення поставленої мети забезпечуватиметься за допомогою методів системного аналізу та експериментальної реалізації веб-застосунку.

Наукова новизна результатів полягає в наступному:

- оцінка впливу розробленого демонстраційного веб-застосунку за допомогою архітектури мікрофронтендів на продуктивність,

швидкість реалізації та розгортання;

– дослідження ефективної можливості впровадження даної архітектури у масштабних проєктах.

Практична цінність. Результати, які отримані у процесі дослідження, можуть бути використані для проєктування масштабних застосунків в IT-компаніях. Також проведене дослідження можна використовувати в навчальних курсах фронтенд-інженерії.

МІКРОФРОНТЕНДИ, АРХІТЕКТУРА, ВЕБ-ЗАСТОСУНКИ, WEBPACK MODULE FEDERATION, VUE.JS, REACT, SINGLE-SPA.

ABSTRACT

The work consists of an introduction and four chapters. The total volume of the work: 115 pages of main text, 50 illustrations, 31 tables. In preparation, literature from 38 sources was used.

Relevance of the topic. The present day dictates new rules in the process of developing web applications. The development of modern web systems requires new approaches, as the requirements for complexity, scalability, and performance are changing. It should be noted that the traditional type of frontend architecture—monolithic—no longer satisfies the fulfillment of all the stated requirements for applications, which include rapid scaling and system support through the parallel work of large teams. Since web applications are developed for a significant number of users, developers must constantly add new functional needs, and thus require a better system architecture.

The described conditions force us to consider that the concept of "micro-frontend" is a sufficiently good and modern solution for applying microservices architecture at the frontend level (interface). This approach will allow for the division of a large frontend part into independent services, which can be worked on by separate development teams.

The effectiveness of research on this topic is proven, as the implementation of such an architectural pattern is in demand for ensuring the development of large, flexible, scalable, and technologically sound web applications, as well as for accelerated release and further reliable system support.

Goal and objectives of the research. The goal of this research is to analyze the design approach and implement an application based on micro-frontends, whose architecture will be provided through technologies such as Vue, React, and the Single-SPA platform. The task of the research is not only the development of a web application but also the evaluation of its effectiveness.

To achieve the stated goal, it is necessary to accomplish a series of tasks:

Analyze the evolutionary component in architectural approaches to the development of the frontend component;

- Study the concept of "monolithic" frontend and its limitations;
- Consider the concept of microservices and their differences from micro-frontends;
- Investigate the main approaches (architectural patterns) for building micro-frontends;
- Analyze the tools that ensure the implementation and adoption of the studied architectural approach;
- Design, develop, and investigate a demonstration web application using modern frameworks Vue and React;
- Evaluate the effectiveness of the conducted research;
- Develop a startup project.

The object of research is the micro-frontend architecture and its impact on the scalability of web applications.

The subject of research is architectural approaches in the development of micro-frontends, as well as the tools and technologies for their implementation.

Research methods. Achieving the stated goal will be ensured through methods of systems analysis and experimental implementation of a web application.

The scientific novelty of the results lies in the following:

- Assessment of the impact of the developed demonstration web application using micro-frontend architecture on performance, implementation speed, and deployment;
- Investigation of the effective possibility of implementing this architecture in large-scale projects.

Practical value. The results obtained in the course of the research can be used for designing large-scale applications in IT companies. The conducted research can also be used in frontend engineering training courses.

MICRO-FRONTENDS, ARCHITECTURE, WEB APPLICATIONS, WEBPACK MODULE FEDERATION, VUE.JS, REACT, SINGLE-SPA.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 4
ВСТУП 5
РОЗДІЛ 1 ОГЛЯД ТЕОРЕТИЧНИХ ОСНОВ МІКРОФРОНТЕНДІВ 6
1.1 Еволюція технологій фронтенду 6
1.2 Поняття монолітного фронтенду та його обмеження 8
1.3 Концепція мікросервісів у веб-розробці 10
1.4 Визначення мікрофронтедів та їх відмінності від мікросервісів 15
1.5 Переваги та недоліки мікрофронтендів 17
1.6 Аналіз архітектурних підходів до розробки мікрофронтендів 19
1.6.1 Мікрофронтенди через маршрутизацію 19
1.6.2 Мікрофронтенди через Web Components 20
1.6.3 Архітектура Module Federation 22
1.6.4 Server-side composition 23
1.6.5 Performance-патерни та кращі практики 25
1.6.6 Порівняльний аналіз варіантів 26
Висновки до розділу 1 28
РОЗДІЛ 2 ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ ВПРОВАДЖЕННЯ МІКРОФРОНТЕНДІВ 30
2.1 Дослідження платформ: Single-SPA, Luigi, Frint.js, Piral 30
2.2 Підхід інтеграції мікрофронтендів за допомогою Webpack та маршрутизації в Single-SPA 37
2.3 Інтеграція різних фреймворків: Vue + React 39
2.4 Механізми комунікації між мікрофронтендами 40
Висновки до розділу 2 43
РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ НА ОСНОВІ МІКРОФРОНТЕНДІВ 45
3.1 Формування вимог та бізнес-процесів 45
3.2 Моделювання процесів за допомогою BPMN, IDEF0 46

3.3 Проєктування архітектури додатка 51
3.4 Реалізація з використанням Single-SPA 54
3.5 Інтеграція модулів на Vue та React 56
3.6 Тестування та результати впровадження 59
3.7 Оцінювання масштабованості та ефективності 68
3.7.1 Метрики продуктивності фронтенду 68
3.7.2 Аналіз продуктивності моноліту vs мікрофронтендів 73
3.7.3 Метрики продуктивності фронтенду 79
Висновки до розділу 3 81
РОЗДІЛ 4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ 83
4.1 Загальна характеристика стартап-проєкту 83
4.2 Технологічний аудит ідеї проєкту 86
4.3 Аналіз ринкових можливостей запуску стартап-проєкту 87
4.4 Розроблення ринкової стратегії проєкту 97
4.5 Розроблення маркетингової програми стартап-проєкту 100
Висновки до розділу 4 107
ВИСНОВКИ 109
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 112
ДОДАТКИ 116

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

HTML (HyperText Markup Language) – мова розмітки для створення структури веб-сторінок, визначає елементи як заголовки, параграфи та посилання.

CSS (Cascading Style Sheets) – мова стилів для оформлення HTML, контролює кольори, шрифти, макет і анімації

DOM (Document Object Model, об'єктна модель документа) - деревооб'єктне представлення HTML/XML у браузері, яке дозволяє динамічно змінювати вміст через JavaScript.

API (Application Programming Interface) – набір правил і протоколів для взаємодії між програмними компонентами, наприклад, REST API для обміну даними між фронтендом і бекендом.

Single-SPA – фреймворк для мікрофронтендів, що дозволяє інтегрувати кілька незалежних SPA в єдиний веб-додаток з централізованою маршрутизацією.

ВСТУП

Здавалося ще зовсім недавно використовували мову програмування JavaScript для примітивних задач по зміні кольору тексту на веб-сторінці. А далі веб технології почали стрімко розвиватись, а внаслідок цього – стрімко збільшувалась складність розроблений веб-застосунків.

Сьогодні веб-застосунки перетворились на швидкі, динамічні та інтерактивні платформи, які своєю роботою повинні забезпечити виконання низки бізнес-процесів для різного роду установ та підприємств. Оскільки постійне збільшення користувачів та затребуваного ними нового функціоналу потребує частих та швидких оновлень, то значну увагу слід приділяти сучасним та гнучким архітектурним рішенням. Для проєктування фронтенд частини веб-застосунків традиційно використовувалась монолітна архітектура, однак при масштабуванні проєкту такий архітектурний шаблон почав більшою мірою заважати, ніж допомагати. Це пояснюється тим, що в даному шаблоні програмний код (логіка роботи клієнтської частини) зв’язаний між собою та сповільнює розробку, оскільки будь-яка зміна та додавання нових функціональних потреб – змушує розробників глибоко тестувати весь клієнтський рівень. Цей підхід значно сповільнює процес розробки та ускладнює підтримку великих застосунків. Також використання монолітної архітектури в проєктах, які потребуватимуть частих оновлень, швидкого масштабування та довгої підтримки, збільшує ризики виникнення критичних помилок.

Описані проблеми спонукають веб-розробників переходити на розподілений тип архітектура. Як наслідок на бекенд-частині розробки дану проблему вирішено за допомогою переходу на мікросервісну архітектуру. Тривалий час даний підхід не поширювався на фронтенд-розробки, однак останні роки – сформувалась концепція мікрофронтенду. Цей підхід переносить досвід проєктування бекенду на клієнтський рівень.

РОЗДІЛ 1
ОГЛЯД ТЕОРЕТИЧНИХ ОСНОВ МІКРОФРОНТЕНДІВ

1.1 Еволюція технологій фронтенду

Сьогодні у світі важко знайти ще якусь галузь, яка б так само стрімко розвивалась як фронтенд-розробка. Останні три десятки років можна спостерігати трансформацію від простих тестових сторінок до масштабного, інтерактивного та динамічного веб-застосунку.

Кожен етап розвитку тісно пов’язаний з потребами користувачів. Було пройдено довгий шлях від напівпустої HTML-сторінки через період технології Flash до прогресивних веб-застосунків, які використовують JavaScript фреймворки.

На початку 90-х років веб-сторінка представляла собою надпростий текстовий документ HTML. Тоді ще не існувало стилізації, тому всі веб-сайти були схожі один на одний. У 1994 (96) році вперше було запропоновано CSS, який значною мірою покращив вигляд вмісту на сторінках, однак досі вигляд сторінок залишався статичним. Хоч це й були перші кроки у процесі фронтенд-розробки, однак вони залишаються актуальними до сих пір [1].

На кінці 90-х років з’явилась технологія JavaScript, яка «оживила» мертві веб-сторінки, що нагадували друковані версії газет. З появою даної технології уже можна було створити мінімально інтерактивний інтерфейс того часу. Важливим етапом розвитку фронтенду стала поява в 2005 році технології AJAX (для асинхронної взаємодії з бекендом), яка дозволила виконувати обмін даними з сервером без оновлення веб-сторінки.

У 2000-х роках з’явилась на світ бібліотека jQuery, яка значно спростила роботу з DOM (об’єктною моделлю документа). Але варто зазначити, що справжньою революцією стала поява на світ таких веб-фреймворків як React, Angular та Vue.js. Треба зазначити, що кожен з цих

інструментів приніс свій підхід у галузь веб-розробки. Фреймворк React запропоновав свій принцип віртуального DOM, а також компонентний підхід до розробки. Angular у свою чергу запропонував двосторонню прив'язку даних. Vue.js вирішив об'єднати кращі рішення конкурентів та представити це у своєму спрощеному вигляді.

На даний час можна побачити не просто певний період еволюції, а фундаментальну зміну – перехід від монолітного фронтенду до мікрофронтенду На рис. 1.1 представлено різницю між двома архітектурами веб-додатків: монолітною та мікрофронтендами. З рисунка видно, що кожен мікрофронтенд розробляє своя команда [2].

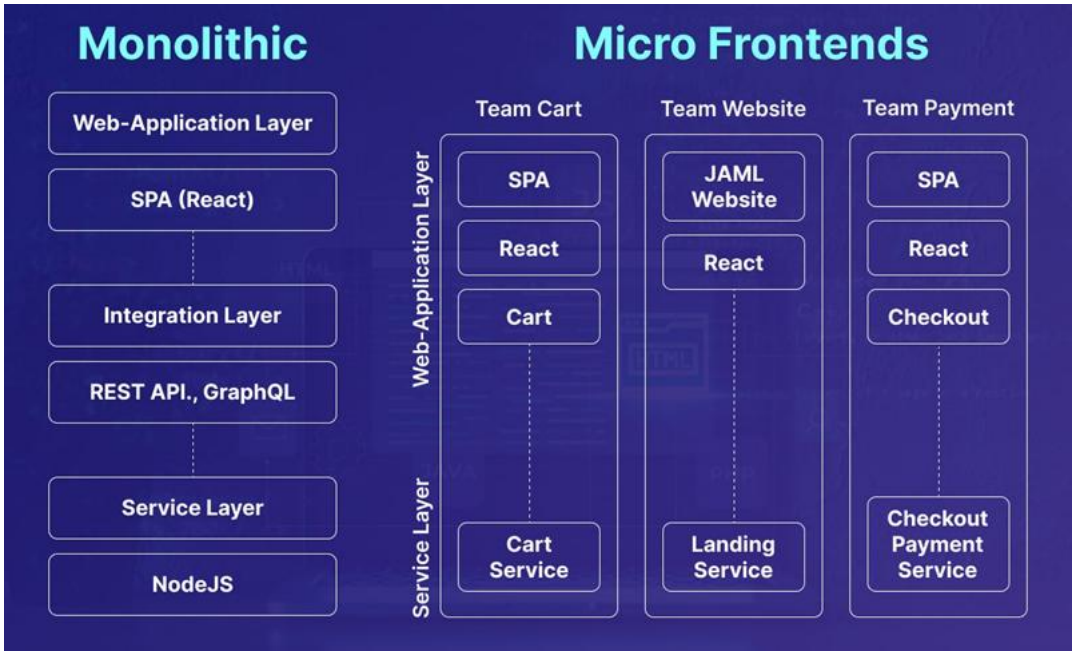


Рис. 1.1. Архітектура веб-додатків [2]

Деякі інші сучасні фреймворки все частіше переносять частину логіки робити ближче до кінцевого користувача. Також веб-фреймворки на кшталт Remix чи Next.js розмивають кордони між клієнтською та серверною частинами, а отже з'являються fullstack-рішення.

Сучасний підхід до розробки клієнтської частини за допомогою компонентів – повністю змінив принцип організації коду. Такий підхід дозволяє кожному компоненту стати незалежною частинкою та брати

участь в побудові динамічного інтерфейсу. Також у час, коли є різноманітна кількість екранів різних форм-факторів, – адаптивний дизайн більше не додаткова опція, а обов’язкова технічна вимога до проєктування веб-застосунків.

Застосунки з кожним разом стають більшими та масштабнішими, отже досить актуальною тенденцією є перехід на мікрофронтенд. Однак варто пам’ятати, що такий перехід – це не забаганка розробників, а в першу чергу покращення користувацького досвіду.

1.2 Поняття монолітного фронтенду та його обмеження

Монолітна архітектура – це підхід до розробки програмного забезпечення, в якому всі існуючі компоненти взаємодіють один з одним та знаходяться на одному сервері. Дана архітектура представляє собою єдиний модуль, який працює автономно. Щоб зробити оновлення такої системи – потрібно виконати оновлення всього коду при проєктуванні та розгортанні нової версії. Такий підхід ускладнює роботу з оновлення та затрачає велику кількість часу [3].

Тип архітектури моноліт зручно використовувати на початкових етапах розробки, оскільки проєктування такої архітектури є простішим та не потребує глибоких знань розробника. Для прикладу монолітного фронтенду можна навести веб-застосунок будь-якого середнього онлайн-магазину, де логіка додавання товарів в корзину чи обробка замовлень працює як одне ціле на одному сервері. У даному випадку логіка роботи системи знаходиться в одному місці та в одному програмному коді, а всі зміни виконуються в рамках цього коду. Варто зазначити, що у монолітній архітектурі легше тестувати безпеку, а також атаки на неї менші, оскільки даний тип архітектури має менше зовнішніх зв’язків (або мінімальну кількість).

Отже, використання монолітної архітектури теж має свої переваги,

оскільки дозволяє розробити програмне забезпечення швидше. Також до переваг даного типу архітектури відносять:

- використання одного виконуваного файлу спрощує розгортання системи;
- завдяки централізації можна добитись кращої продуктивності (тільки на початковому етапі життя масштабного веб-застосунку);
- полегшує пошук проблем у програмному коді;
- простіше у розробці.

Монолітний тип архітектури має значні переваги, але тільки до того часу, поки застосунок не стає занадто великим та важким. Громіздкий код важко масштабується, оскільки для того, щоб внести незначні зміни – потрібно повністю скомпілювати та протестувати всю систему. Такий підхід суперечить сучасним підходам (agile) програмістів до розробки. Обмеження монолітного типу архітектури представлені на рис. 1.2 [4].

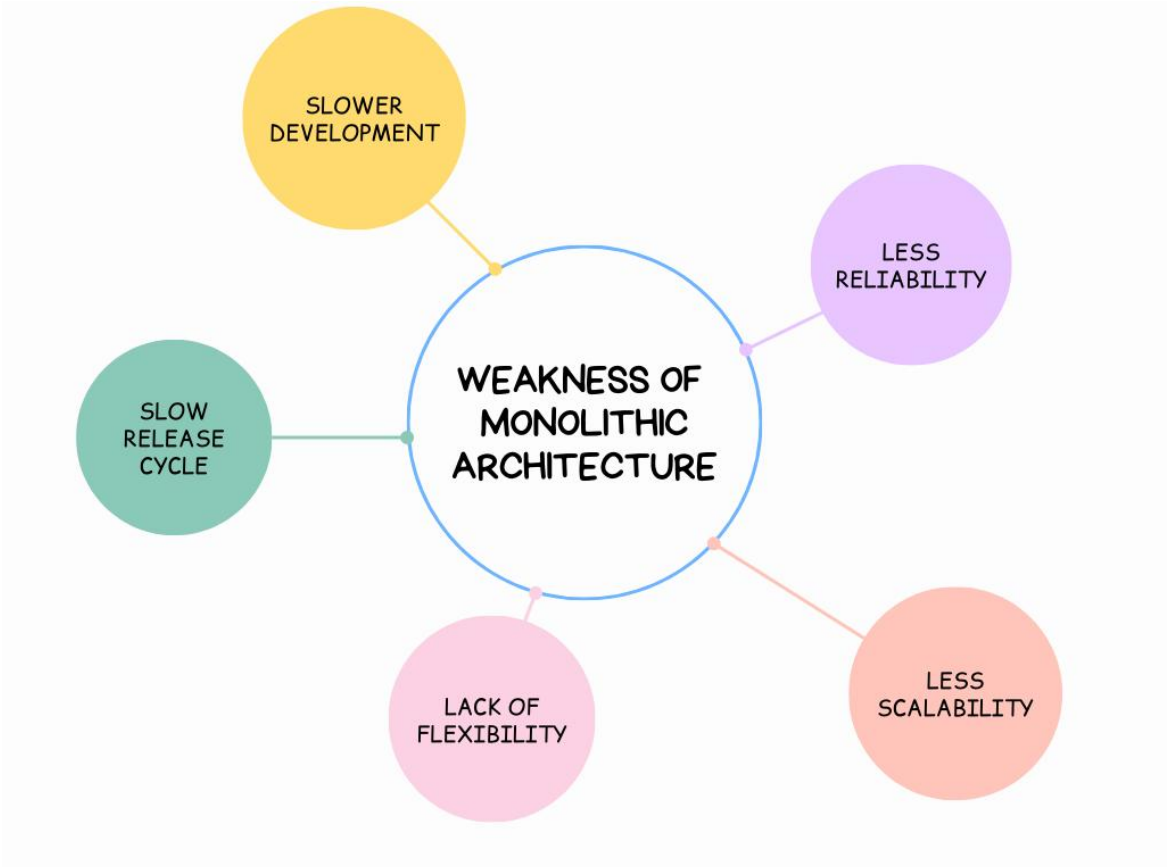


Рис. 1.2. Слабкі сторони моноліту [4]

З рис. 1.2 можна зазначати, що дані обмеження є недоліками і до них також відносять:

- ненадійність (помилка в одному компонентів може покласти крах всій системі);
- складність в повторному розгортанні (при додаванні нових змін потрібно повністю розгортати цілу систему);
- зниження швидкості розробки (при проєктування великих веб-застосунків);
- проблеми із гнучкістю (монолітні системи можуть мати обмеження для деяких технологій);
- великі затрати при масштабуванні;
- важкість при масштабуванні окремих компонентів.

Монолітний тип архітектури ідеально підходить для невеликих або середні проєктів, де всі функціональні потреби вже реалізовано і в майбутньому швидкий та значний ріст не прогнозується.

1.3 Концепція мікросервісів у веб-розробці

Концепція мікросервісів – це тип архітектури, який забезпечує створення веб-застосунків з набору невеликих сервісів, які взаємодіятимуть між собою. Мікросервісний тип архітектури відрізняється від монолітного наступними особливостями [5]:

- вільне зв’язування;
- масштабованість (мікросервіси можна розвивати та масштабувати незалежно один від одного, а в монолітному типі необхідно масштабувати весь застосунок);
- незалежне розгортання (можна розгорнути мікросервіс без простою всієї системи, однак зміни в монолітній архітектурі потребуватимуть зупинки застосунка);
- децентралізація (функціональність веб-застосунку з

мікросервісною архітектурою розділяється між сервісами, а в монолітному вся функціональність в одному програмному коді);

– ізоляція помилок (вихід з ладу одного сервісу в мікросервісах критично не впливає на роботу застосунку, однак в монолітному може спричинити вихід з ладу усієї системи).

За допомогою табл. 1.1 буде представлено комплексне порівняння монолітної та мікросервісної архітектур [6].

Таблиця 1.1

Порівняльна характеристика монолітної та мікросервісної архітектур

Ознака	Тип архітектури	
	Моноліт	Мікросервіси
Зв’язок між компонентами	Тісно пов’язані між собою, а також містять прямі залежності	Слабка зв’язність, однак присутній високий зв’язок за допомогою API
Архітектурний стиль	Вся функціональність містить в уніфікованій базі коду	Розподіл функціональності по невеликих та незалежних сервісах
Стек технологій	Для всієї системи використовується єдиний стек	Можна для різних сервісів використовувати різні технології
Етап впровадження	Систему можна розгорнути тільки як єдине ціле	Кожен сервіс можна назалежно розгорнути

Таблиця 1.1 (продовження)

Порівняльна характеристика монолітної та мікросервісної архітектур

Ознака	Тип архітектури	
	Моноліт	Мікросервіси
Стійкість до збоїв	При збоях в одному з компонентів може виникнути збій в усьому веб-застосунку	Виникнення збою в окремому мікросервісі ніяким чином не впливає на інші сервіси
Масштабування	Систему можна масштабувати як єдине ціле	Кожен серіс можна масштабувати незалежно один від одного
Командна робота	Конфлікти в коді у зв’язку з великою кількістю розробників	Розробники можуть працювати паралельно з різними сервісами
Тестування	Виконується простіше, оскільки всі тести в одному середовищі	Складніше та довше, легко проводити тільки одиничні тести. Також необхідна більша кількість інструментів для тестування
Продуктивність та мережа	Швидше завдяки меншим мережевим викликами	Виникають додаткові затримки в роботі через мережу. Також складніший моніторинг

Отже, до переваг мікросервісів відносять:

- можливість гнучко вибрати технологію до кожного сервісу;
- належний підхід до розробки;
- масштабування кожного сервісу відповідно до попиту;

- ізоляція помилок в межах певно мікросервісу;
- легша підтримка та краще розуміння окремого сервісу.

Незважаючи на ряд переваг, також слід виділити найбільші виклики, з якими стикнуться розробники при переході з монолітної на мікросервісну архітектуру:

- потреба в більш складному підході у процесі розробки;
- потрібно проводити складніші тестування, щоб забезпечити сумісність всіх мікросервісів;
- контроль між розподіленим управлінням;
- складність при проєктуванні безпеки мікросервісів;
- може викликати затримки при міжсервісному зв’язку при поганому проєктуванні.
- високі вимоги до інфраструктури, проблеми із стандартизацією.

В основному мікросервіси взаємодіють між собою через інтерфейси програмування додатків (API). Вони описують як дані будуть передаватись та в якому форматі. Існують наступні схеми взаємодії [6]:

- синхронний запит-відповідь (деякий сервіс надсилає запит до іншого та очікує на його відповідь. З одного боку все достатньо просто, однак якщо все не налаштовано коректно, то можуть виникнути проблеми з продуктивністю);
- асинхронний обмін повідомленнями (в даному випадку сервіси не очікують на відповідь, вони просто надсилають запит через посередника і виконують свою наступну роботу, а отримувач, як тільки-но буде мати змогу, надішле відповідь. Дана схема взаємодії є найефективнішою при обробці значною кількості даних);
- архітектура на основі подій (дана схема дещо схожа на підписку на розсилку, де потрібно реагувати, але не одразу).

За допомогою табл. 1.2 буде представлено наглядний порівняльний аналіз схем взаємодії.

Таблиця 1.2

Порівняльна характеристика схем взаємодії між мікросервісами

	Схема взаємодії		
	Запит-відповідь	Обмін повідомленнями	На основі подій
Тип зв’язку	Синхронний	Асинхронний	Синхронний
Приклад застосування	Перевірка ціни товару в магазині	Обробка замовлення	Повідомлення про платіж
Переваги	Швидкий для простих завдань	Легко масштабується	Гнучка схема, однак має слабкий зв’язок
Недоліки	Може гальмувати та викликати збої	Можуть виникати затримки при роботі	Важче в налагодженні

Мікросервіси – це не універсальні рішення, тому перед початком розробки веб-застосунку потрібно врахувати конкретні функціональні потреби, а вже потім за рахунок вимог здійснити вибір на користь певного архітектурного рішення. Варто згадати конкретні та успішні приклади застосування мікросервісної архітектури:

- Amazon (перевели сайт електронної комерції з монолітної архітектури на мікросервісну для покращення адаптації);
- Netflix (вважається першопрохідцем, оскільки їхній підхід використовується як об’єкт досліджень для іншим компаній. Використали мікросервісну архітектуру у своєму потоковому сервісі для керування значними обсягами даних);
- Uber (постійне зростання потребує застосування мікросервісів для управління запитати, платежами та іншим).

Успішні приклади, які було наведено вище, демонструють ефективність та вплив мікросервісних систем на кінцевого користувача.

1.4 Визначення мікрофронтедів та їх відмінності від мікросервісів

Веб-застосунки стали об’ємнішими та складнішими, а також почали довго розроблятись, повільніше розгортатись та важче підтримуватись за допомогою монолітної архітектури. Тому розробники прийшли до ідеї реалізувати мікросервісну архітектуру в процесі проєктування фрондент рівня та створили концепцію, яка називається – мікрофронтенди.

Мікрофронтенди – це мікросервісний підхід при проєктуванні клієнтського рівня веб-застосунка на подобі того, що існує на бекенд-частині. Отже, розробник ділить проєкт, який реалізована у монолітній архітектурі, на незалежні частинки. Найважливішим інструментом, який допомагає здійснити такий поділ, є Webpack Module Federation [7].

Зазвичай проєкт будують наступним чином: кожна сторінка деякого програмного продукту поділена на окремий мікрофронтенд і за нього відповідає своя команда розробників. З цього випливає, що для розробки таких мікрофронтедів потрібно значна кількість фронтенд-розробників.

При використанні мікрофрондендів процес розробки виглядає по-іншому, аніж при роботі з монолітною архітектурою. Менеджеру проєкту легше контролювати процес такої розробки, оскільки кожна фронтенд-команда сфокусована над вирішенням своїх нагальних потреб. Також використання мікрофронтедів дозволяє розробникам обирати фреймворки, бібліотеки та різноманітні інструменти на власний розсуд, так як це ніяким чином не вплине на інші фронтенди.

Варто зазначити, що використання мікрофронтендів, краще дозволяє угодити дії розробників та мінімізувати кількість конфліктів. Даний підхід покращую якість коду та усуває значну кількість багів. Щодо тестування, то тут теж є значна ефективність при розробці: кожна команда, незважаючи на іншу, тестує свої мікрофронтенди. Також хорошим рішення є на кожну команду виділити окремого розробника для проведення різного

роду тестувань.

Розгортання окремих мікрофронтендів теж проводять без будь-якого негативного впливу на сам проєкт. Розробникам не має потреби зв’язуватись з іншими командами чи встановлювати єдину дату для розгортання, а також не потрібно збиратись для виконання хотфіксів – всі ці процеси можна повноцінно зробити окремо. Даний підхід значною мірою додає зручності та гнучкості під час розробки веб-застосунку.

Слід звернути увагу, що хоч мікрофронтенди базуються на ідеї мікросервісів, однак між ними існують значні відмінності. У той час як мікросервіси працюють на стороні серверного рівня (бекенду) та відповідають за роботу бізнес-логіки, а також комунікацію між різноманітними сервісами за допомогою API, то мікрофронтенди сконцентровані на користувацькому рівні та досвіді користувача. Суттєві відмінності полягають у процесі розгортання: мікросервіси можна деплоїти без будь-якого впливу на веб-застосунок, однак для деплою мікрофронтендів слід сконцентрувати увагу над узгодженням зовнішнього вигляду та складової дизайну. За допомогою табл. 1.3 буде представлено основні відмінності між мікрофронтедами та мікросервісами [8].

Таблиця 1.3

Основні відмінності між мікрофронтедами та мікросервісами

Критерій	Тип архітектури	
	Мікрофронтенди	Мікросервіси
Застосування	Клієнтський рівень (для розробки користувацького інтерфейсу)	Серверний рівень (логіка роботи веб-застосунку)
Комунікація	Single-SPA, Web Components, Module Federation	API

Таблиця 1.3 (продовження)

Основні відмінності між мікрофронтедами та мікросервісами

Критерій	Тип архітектури	
	Мікрофронтенди	Мікросервіси
Розгортання	За допомогою мережі доставки контенту (CDN)	CI/CD для кожного сервісу, контейнери
У разі чого використовують	Коли проєктують масштабний інтерфейс користувача з великою кількістю розробників	При проєктування систем, які потребують високе навантаження на дані
Продуктивність	Залежить від браузера та проєктування системи	Залежить від мережі
Безпека	CORS, Shadow DOM	OAuth, JWT, ізоляція сервісів

Отже, з табл. 1.3 можна зрозуміти, що мікрофронтеди та мікросервіси мають значні відмінності, однак обидва підходи до проєктування частин веб-застосунку (клієнтської та серверної) спрямовані на масштабованість, підвищення гнучкості та швидкодії. Як мікросервіси, так і мікрофронтенди забезпечують одночасну роботу команд над системою, що у подальшому спрощує обслуговування та покращує якість оновлень.

1.5 Переваги та недоліки мікрофронтендів

Архітектура мікрофронтендів має як очевидні переваги, так і власні недоліки. Розглянуті нижче сильні та слабкі сторони потрібно врахувати у процесі проєктування сучасного веб-застосунку. Отже, до переваг архітектури мікрофронтедів відносять [8]:

– незалежне розгортання (кожен модуль можна розгортати окремо

один від одного, а також швидку виправити помилки. Використання мікрофронтедів значно зменшує ризика на етапі релізів);

- зручніша підтримка та тестування;
- дана архітектура дозволяє легко модернізувати застарілу веб-систему (можна переписувати частинки модулів не зламавши роботу застосунку);
- прискорення розробки шляхом застосування кількох незалежних команд (кожна група розробника відповідає за свій модуль);
- необмежений вибір технологій та інструментів;
- стабільна робота системи навіть при виході деякого модуля з ладу.

Недоліками такого підходу проєктування фронтенду називають [9]:

- можуть виникнути проблеми з продуктивністю (для запобігання подібних проблем слід застосовувати спільне кешування або оптимізацію);
- складність у процесі проєктування;
- велике фінансові навантаження на інфраструктуру;
- проблема узгодженості дизайну (оскільки різні команди програмістів працюють над своїми модулями, то виникають розбіжності в дизайні. Потрібно прикладати додаткові зусилля для створення централізованої дизайн-системи);
- високі вимоги для координації;
- ускладнене відлагодження та моніторинг (для вирішення цієї проблеми потрібно використовувати спеціалізовані рішення).

1.6 Аналіз архітектурних підходів до розробки мікрофронтендів

1.6.1 Мікрофронтенди через маршрутизацію

Мікрофронтенди на основі маршрутів – це один з найпростіших та найпоширеніших підходів при проєктуванні архітектури мікрофронтендів. Даний підхід засновано на маршрутизації, а суть цього підходу полягає в тому, що громіздкий веб-застосунок буде поділено на окремі частинки (модулі), де кожен відповідатиме за свій маршрут у користувацькому інтерфейсі.

До основної одиниці поділу мікрофронтендів через маршрутизацію належить сторінка або деякий логічний розділ. Цей розділ має свій унікальних шлях. Тобто, кожен із маршрутів може підвантажити окрему фронтенд-частинку, а вона у свою чергу матиме свою логіку роботу та окремий технологічний стек. Також у разі переходу користувача на таку сторінку – завантажується лише конкретний мікрофронтенд [10].

Для узгодження переходів між частинками (мікрофронтендами) існує центральний контейнер-додаток. Він відповідає за маршрутизацію та виконує роль головного шлюза на клієнтському рівні. Даний контейнер містить спільну навігацію, глобальні стилі, хедер та футер, отже він відповідає за вигляд каркасу інтерфейсу. На рис. 1.3 представлено схематичний вигляд архітектури мікрофронтендів через маршрутизацію.

Основною перевагою мікрофронтендів через маршрутизацію називають простоту реалізації. Даний підхід можна впровадити без складної перебудови програмного коду та додаткових інструментів, на відміну від Module Federation чи Server-side composition. В останньому випадку також необхідно використовувати спеціалізовані інструменти. Також до переваг відносять більшість сильних сторін мікрофронтедів, а саме: ізоляція між модулями, незалежне використання команд розробників та власних інструментів розробки.

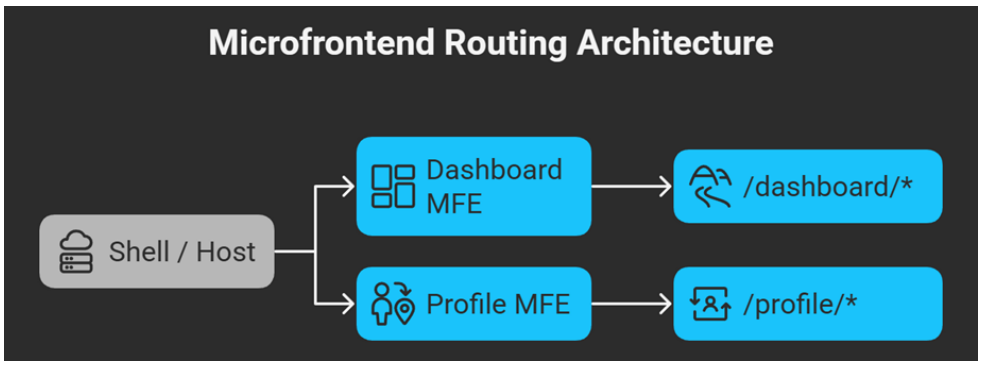


Рис. 1.3. Архітектурний підхід: мікрофронтенди на основі маршрутів [10]

Однак, важливу увагу слід приділити недоліками мікрофронтендів на основі маршрутів, до таких відносять:

– повторення програмного коду (негативно вливає на продуктивність. Мінімізують даний ефект за рахунок впровадження механізмів кешування);
– відсутність глибокої інтеграції між модулями (це найочевидніших недолік, оскільки кожна частинка-мікрофронтенд працює як окрема сторінка. Для обміну даними між ними потрібно централізований сховище на подобі API серверного рівня);
– складнощі із забезпеченням єдиного дизайну веб-застосунку (необхідно узгоджувати спільний дизайн системи).

Отже, підхід на основі маршрутизації є найпростішим рішенням, особливо при переході з монолітної на мікрофронтендну архітектуру. Хоч цей архітектурний метод має ряд недоліків, однак він добре підходить до систем, в яких кожен розділ повинен виконувати свою функцію.

1.6.2 Мікрофронтенди через Web Components

Web Components – це найбільш універсальних метод для реалізації архітектури мікрофронтендів. Дана технологія забезпечує створення ізольованих компонентів, які можна використовувати багаторазово, і це ніяким чином не залежить від фреймворку та програмного коду.

Web Components містить в собі ряд технологій: Custom Elements

(визначає як створювати та реєструвати нові елементи), Shadow DOM (запобігає конфліктам між різними частинами веб-застосунку), HTML Templates (підвищує ефективність роботи в браузері за рахунок того, що визначає шаблони, які не рендеряться зразу) та ES Modules (система, яка дозволяє виконувати експорт/імпорт модулів між частинами застосунку) [11].

Отже, завдяки розглянутому набору технологій, Web Components створює незалежні модулі, а також може виконувати їх підключення в застосунку будь-якої складності. До основних переваг підходу проєктування мікрофронтендів через Web Components належить:

- високий рівень сумісності (компоненти, які створені за допомогою Web Components можна застосувати у будь-якому середовищі);
- ізоляція (усування конфліктів між компонентами завдяки існуючому набору технологій);
- підтримується на нативномі рівні браузерами (значно покращує продуктивність, особливо помітно у великих корпоративний веб-застосунках).

Недоліками цього методу проєктування мікрофронтендів називають:

- відсутність зручних механізмів реалізації;
- відсутній централізований підхід до розробки дизайну;
- погано розвинена екосистема, яка у свою чергу ускладнюю розробку масштабних інтерфейсів користувача.

Web Components зазвичай використовують як основну бібліотеки UI-компонентів, в цьому випадку легко забезпечити єдиний стандарт у спроєктованому веб-застосунку. У загальному даний підхід до проєктування мікрофронтендів виглядає наступним чином: окремі команди розробників створюють свої компоненти у стилі Web Components, які потім розгортають у вигляді сервісів, що у свою чергу будуть підключені до основної веб-системи за допомогою HTML-тегів.

Якщо брати в порівняння підхід через маршрутизації та через Web Components, то другий забезпечить більше можливостей – дозволить окремі частинки-модулі вставляти у якості елементів інтерфейсу, а не тільки окремі сторінки. Однак, даний підхід не завжди зручний, особливо якщо йде мова про інструменти, які дозволяються ефективно керувати станом.

1.6.3 Архітектура Module Federation

Module Federation – це один з найефективніших підходів до проєктування мікрофронтедів, який представлено у Webpack 5. Даний підхід на відміну від двох попередніх (мікрофронтенди через маршрутизацію та мікрофронтенди через Web Components) дозволяє забезпечити глибоку інтеграцію, яка працює на рівні JavaScript. Це значить, що кілька частин застосунка можуть ділити між собою код у реальному часі. Отже концепція проєктування Module Federation забезпечить динамічний та ефективний імпорт модулів з одного застосунку до іншого коли це потрібно. Інструмент Module Federation використовують в наступних сценаріях [12]:

- для поділу веб-додатків на частини (даний поділ виконується для над великих фронтенд-застосунків);
- динамічного завантаження програмного коду (завантажується тільки та частина коду, яка необхідна в даний момент. Такий підхід значною мірою покращує продуктивність);
- спільне використання компонентів.

Підхід до проєктування Module Federation має свою архітектуру, яка складається наступним елементів:

- Host Application (головний елемент, відповідає за логіку, стан та маршрутизацію);
- Remote Applications (модулі, які зачасту розроблені командою розробників);

– Shared Dependencies (для запобігання дублювання коду використовуються спільні бібліотеки).

Отже, проаналізувавши даних підхід слід визначати переваги та недоліки, до переваг відносять:

– зручне та ефективне повторне використання програмного коду (відсутнє дублювання коду за рахунок використання спільних бібліотек);
– гучка технологія (незважаючи на те, що кожен застосунок-компонент має бути зібраний Webpack, однак всі вони можуть мати незалежну внутрішню структуру);
– незалежне розгортання та динамічне оновлення модулів веб-застосунку.

Основними недоліками та обмеженнями Module Federation називають: складність керування інструментом Webpack та конфігурацією, конфліктами несумісності різних версій бібліотек в модулях та підвищену складність у процесі тестування розробки. Також до недоліку відносять залежність Module Federation від Webpack 5, оскільки при потребі використати інші збирачі – необхідні додаткові рішення.

1.6.4 Server-side composition

Server-side composition (або серверний підхід) – архітектурне рішення, яке забезпечує збирання інтерфейсу користувача не у веб-браузері, а на сервері. Даний метод забезпечує остаточне об’єднання мікрофронтедів у єдину HTML-сторінку, яка у свою чергу (у вже готовому вигляді) надсилається користувачу веб-застосунку. Серверна композиція дозволяє скоротити час першого завантаження, оскільки процес генерації виконується ще до відправки сторінки клієнту. Для реалізації цього підходу найбільш популярними є інструменти Edge Side Includes (ESI) та фреймворк Tailor [13].

Технологія Edge Side Includes (ESI) використовується для композиції

сторінок: мікрофронтенд генерує фрагмент (навігацію, для прикладу), а edge-сервер паралельно виконує об’єднання за допомогою тегів та вставляє у загальну HTML. До переваг Edge Side Includes відносять високу продуктивність при формування статичних або напівдинамічних сторінок, а також відсутність навантаження на клієнт. Також слід зазначити недоліки цієї технології: ESI здатен працювати тільки з HTML-фрагментами та не може об’єднувати складні компоненти з логікою.

Tailor – відкритий Node.js-фреймворк, який призначений для серверної композиції мікрофронтендів. Він дозволяє зібрати фрагменти мікрофронтендів на сервері, а також паралельно виконувати рендеринг та посилання відповіді клієнту. Основні етапи роботи фреймворка Tailor:

- описує шаблон JSON-складом;
- паралельно виконує запити до мікрофронтендів, збирає потрібні фрагменти та вставляє в шаблон;
- готова сторінка потоково передається клієнту.

Даний фреймворк корисний завдяки низькій затримці при рендерингу та відсутності зупинки роботи при виникненні помилок. Також завдяки Tailor можна простіше масштабувати вже наявний веб-застосунок.

Отже, розглянуто серверний підхід до об’єднання мікрофронтендів, на основі виконаного аналізу можна виділити наступні переваги:

- краща індексування в пошукових система завдяки відрендереній сторінці;
- використання більшої кількості серверів спрощує масштабованість та підвищує відмовостійкість;
- користувачі швидше отримують результати сторінки;
- збирання сторінки на сервері покращує надійність веб-застосунку.

Недоліками серверної композиції називають зменшення інтерактивності сторінки, оскільки це підвищує складність реалізації, та високе серверне навантаження внаслідок потреби значної кількості ресурсів для проведення рендерингу [13].

1.6.5 Performance-патерни та кращі практики

Мікрофронтенди надають значні можливості для масштабування та швидкого розгортання клієнтської частини веб-застосунку, однак дана архітектура може нести свій вплив на продуктивність системи. Кожен фронтенд-модуль створений за допомогою власних бібліотек та інструментів, а це у свою чергу збільшує навантаження на сервер. Тому для мінімізації та уникнення проблем з продуктивність слід застосовувати перевірені шаблони оптимізації.

Достатньо ефективною технікою для оптимізації є lazy loading. Дана техніка дозволяє завантажувати контент тільки в тому разі, коли в ньому є необхідність. Такий тип динамічного завантаження особливо важливий для мікрофронтендів, що працюють через Web Components чи Module Federation. Також часто використовують техніку code splitting, яка забезпечує поділ коду на незалежні частинки. Ці частинки завантажуються браузером поступово, а отже зменшується час рендерингу.

Варто зазначити, що в кожній частинці мікрофронденду часто застосовуються одні і ті ж бібліотеки, а отже кожен модуль повинен виконати завантаження цієї бібліотеки, що в подальшому негативно вплине на сторінку веб-застосунка. Для усунення подібного негативного впливу використовують техніки повторного використання коду та знову ж таки оптимізації. Тому часто спільні бібліотеки розміщують в CDN та використовують singleton-залежності, які дозволяють використовувати бібліотеки спільно та завантажувати один раз.

Оскільки забезпечення продуктивності є досить затребуваним процесом, то слід використовувати інструменти, які допоможуть оцінити ефективність роботи та промоніторити продуктивність в режимі реального часу. До таких інструментів належать: Lighthouse, New Relic, Datadog та інші.

Варто звернути увагу, що достатньо зручно використовувати гібридні патерни, оскільки вони дозволяють активувати тільки потрібну в

даних момент частину інтерфейсу та забезпечуються економію ресурсів. Під час проєктування веб-застосунку з використанням мікрофронтендної архітектури важливо використовувати єдину систему дизайну, а також спільну бібліотеку стилів. Такий підхід значною мірою покращує процес кешування та забезпечує візуальну узгодженість.

1.6.6 Порівняльний аналіз варіантів

Аналіз архітектурних підходів до розробки мікрофронтендів включав дослідження популярних методів, їх переваги та недоліки. Тому для підведення підсумку побудовано порівняльну характеристику розглянутих архітектур (табл. 1.4).

Таблиця 1.4

Порівняльна таблиця характеристик архітектурних підходів

Підхід до проєктування	Характеристика	Застосування	Переваги/недоліки
На основі маршрутизації	Кожна сторінка – окремий мікрофронтенд	Корпоративні панелі та великі SPA	Легка реалізація / відсутність глибокої інтеграції між модулями
За допомогою Web Components	Компоненти для повторного використання	Технологічні системи, UI-бібліотеки	Незалежне використання багатьох технологій / складність при взаємодії
Патерн Module Federation	Застосунок дозволяє забезпечити глибоку інтеграцію на рівні JavaScript	Масштабні корпоративні веб-застосунки	Динамічне оновлення модулів, використання спільних бібліотек / складність керування інструментом Webpack, складність при тестуванні

Таблиця 1.4 (продовження)

Порівняльна таблиця характеристик архітектурних підходів

Підхід до проєктування	Характеристика	Застосування	Переваги/недоліки
Компонування на стороні сервера (Server-side composition)	Інтерфейс збирається на сервері та передається клієнту	Системи з великим навантаженням, які потребують швидкого завантаження веб-сторінок	Краще індексування в пошукових системах, швидкість завантаження / зменшення інтерактивності, високе навантаження на сервери

Проведене дослідження (табл. 1.4) показало, що універсального архітектурного підходу до розробки мікрофронтендів не існує, оскільки для великих команд розробників доцільно розглянути використання Web Components чи Module Federation, для високооптимізованих та швидкісних систем – компонування на стороні сервера, а для простої та надійної реалізації – підхід на основі маршрутизації. Тому вибір здійснено на користь проєктування мікрофронтендів через маршрутизацію та за допомогою технології Webpack.

Висновки до розділу 1

Під час написання першого розділу дипломної дисертації було розглянуто еволюцію технологій проєктування клієнтського рівня веб-застосунку. У ході аналізу досліджено монолітну архітектуру та розглянуто її обмеження, які включають складнощі при масштабуванні та розширенні функціональних потреб. Також проведений аналіз показав, що з такої архітектурою важко працювати командам розробників, оскільки часто виникають конфліктні ситуації в програмному коді системи.

Досліджено, що виявленні недоліки посприяли розвитку мікросервісної архітектура для розробки на серверному рівні. Принципи, на яких заснований мікросервісний підхід, лягли в основу архітектури мікрофронтендів. Мікрофронтенди у свою чергу забезпечили зручний поділ інтерфейсу користувача на окремі модулі, які допомогли прискорити проєктування, розгортанні та масштабування сучасних веб-застосунків.

У ході роботи над розділом визначено, що мікрофронтенди та мікросервіси хоч і мають спільній підхід у процесі розробки, однак вони значно відрізняються за своєю природою. Було розглянуто переваги мікрофрондів, такі як зручність при масштабуванні, можливість розробки веб-застосунку великою командою розробників та незалежному розгортанні. У той же час проаналізовано ряд недоліків: підвищені вимоги до архітектури, складність планування на початковому етапі та проблеми з єдиним стилем користувацького інтерфейсу. Однак варто зазначити, що дані недоліки не є критичними та вже існують напрацьовані рішення для їх вирішення.

Також у ході роботи проведено аналіз архітектурних підходів до розробки мікрофронтенду. Було досліджено такий архітектурний підхід як мікрофронтенди через маршрутизацію. Дане архітектурне рішення зарекомендувало себе як стабільне та безконфліктне для модулів, що робить його хороший варіантом у процесі розробки корпоративних систем.

Проведений аналіз показав, що мікрофронтенди через Web Components – одне з найпопулярніших рішень, оскільки забезпечує повторне використання компонентів у різних модулях та за допомогою різних фреймворків. Досліджено недоліки: складність керування станом та вплив значної кількісті компонентів на продуктивність.

Значний вплив на архітектурні рішення вніс патерн Module Federation завдяки проривній змозі динамічно завантажувати та застосовувати модулі в інших компонентах. Незважаючи на труднощі, такі як розроблення стратегія керування версіями та синхронізацією залежностей для уникнення конфліктів між модулями, Module Federation – незамінний інструмент для проєктування та впровадження мікрофронтендів.

Досліджено підхід компонування інтерфейсу на стороні сервера, який збільшує продуктивність та покращує користувацький досвід через швидке завантаження веб-сторінок. У той же час розглянуті недоліки: зменшення інтерактивності та високе навантаження на сервери – роблять цей підхід менш застосовуваним.

Отже, найкращим підходом для експериментальної розробки веб-застосунку з використанням мікрфронтендів буде його реалізація через маршрутизації за допомогою інтеграції через Single-SPA та збіркою Webpack. Це рішення обрано за рахунок простоти інтеграції, гнучкості та ефективного керування.

РОЗДІЛ 2
ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ ВПРОВАДЖЕННЯ МІКРОФРОНТЕНДІВ

2.1 Дослідження платформ: Single-SPA, Luigi, Frint.js, Piral

Під час розвитку архітектури веб-систем виникла потреба в інструментах, які здатні забезпечити та полегшити процес розробки та розгортання частин користувацького інтерфейсу – платформах для мікрофронтендів. Існуючі платформи повинні спростити інтеграцію кількох мікрофронтендів і при тому зберегти узгоджений дизайн та навігацію. Полегшити процес розробки застосунку з використанням мікрофронтендів можуть фреймворки Single-SPA, Luigi, Frint.js та Piral, які дозволяють краще керувати модульністю та ізоляцією компонентів. Для здійснення конкретного вибору потрібно розглянути кожен із них.

Single-SPA – це фреймворк, який використовують для об’єднання кількох JavaScript мікрофронтендів у великий та масштабний веб-застосунок. Він дозволяє комбінувати фреймворки React, Vue та інші на веб-сторінці. Типовий вигляд веб-застосунку з використанням фреймворку Single-SPA представлено на рис. 2.1 [14].

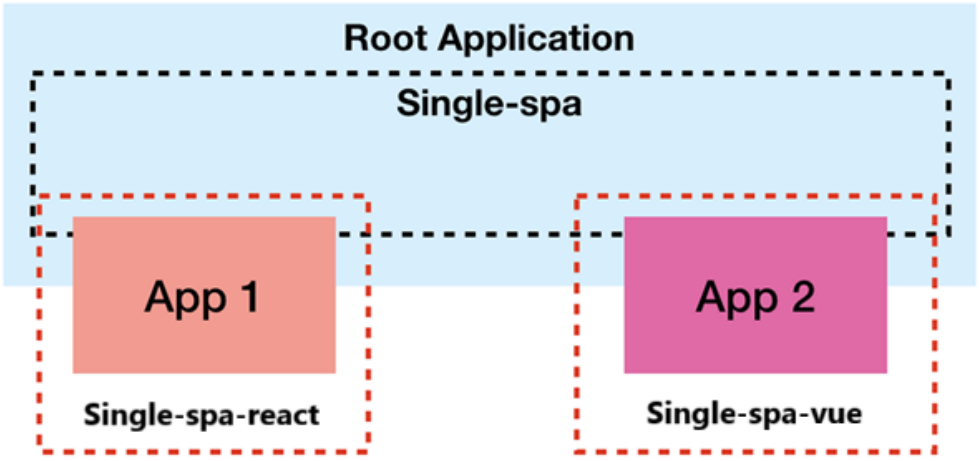


Рис. 2.1. Вигляд застосунку Single-SPA [14]

До переваг Single-SPA відносять:

- можливість підтримки кількох веб-фреймворків одночасно (наприклад, підтримує використання React та Vue в одному інтерфейсі без конфліктів);
- дозволяє поступовий перехід з монолітної до мікрофронтендної архітектури;
- забезпечує гнучку інтеграцію з Webpack чи іншими методами збірки;
- містить достатньо простий та зрозумілий механізм маршрутизації;
- забезпечує плавні переходи при навігації;
- можливість використання lazy loading.

Недоліками цієї фреймворк-платформи називають:

- збільшений час при освоєнні для нових розробників в команді;
- можуть виникати конфлікти версій бібліотек, а також стилів;
- не підходить для великих команд, оскільки потребує більшої комунікації;
- складність при інтеграції;
- для використання значною кількості мікрофронтендів потрібно використовувати оптимізаційні підходи.

Отже, Single-SPA це достатньо простий та ефективний фреймворк, оскільки він не нав’язує свою структуру для проєктування системи, а дає розробнику можливість вибрати інструменти, а також спосіб обміну даними між мікрофронтендами. Ці характеристики забезпечили використання Single-SPA у середніх командах розробників. Недоліки цієї платформи нівелюються, якщо використовувати її для інтеграції меншої кількості мікрофронтендів, а також з малою чи середньою командою розробників.

Luigi – відкрита платформа, яка розроблена компанією SAP, та використовується для проєктування мікрофронтендів. Дана платформа ідеально підходить для тих підприємств, де потрібно до старих систем

додавати нові компоненти без їх переписування. Luigi забезпечує керування автентифікацією, безпекою та навігацією між частинками фронтендів. Ця платформа має свою оболонку, за допомогою якої забезпечує маршрутизацію між мікрофронтендами та завантаження їх у потрібне місце на веб-сторінці. Використання Luigi спрощує адміністрування, оскільки використовує конфігураційний підхід, інтеграція забезпечується через YAML або JSON-файли. Архітектура додатку, який спроєктований за допомогою даної платформи представлено на рис. 2.2 [15].

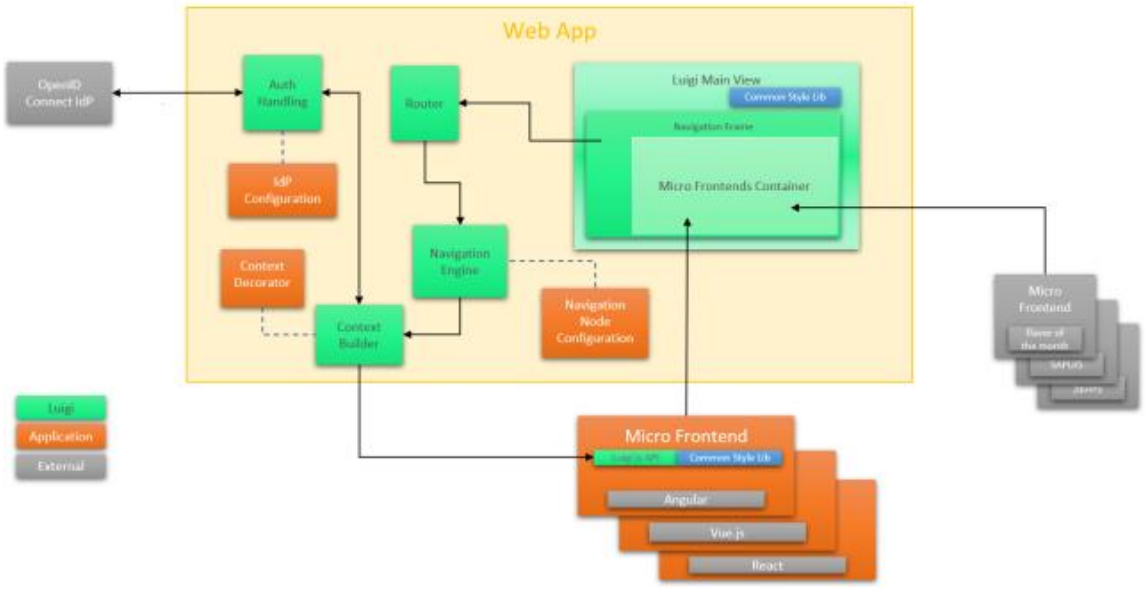


Рис. 2.2. Вигляд архітектури застосунку на платформі Luigi [15]

Перевагами платформи Luigi називають:

- забезпечує єдиний дизайн через централізовану навігацію та теми;
- вбудована авторизація та ізоляція обмежує виникнення конфліктів;
- легко додавати модульність не змінюючи ядро системи.

До недоліками цієї відкритої платформи відносять:

- менше можливостей у порівнянні з Single-SPA;
- потрібно використовувати значну кількість додаткових

інструментів;

- складний обмін даними між мікрофронтендами;
- слабка підтримка спільноти та важке навчання.

Отже, платформа Luigi є досить корисним інструментом розробки, оскільки дозволяє швидко осучаснити старий застосунок на монолітній архітектурі, однак вона більше орієнтується на застарілі корпоративні рішення.

Frint.js – це фреймворк, який використовується для розробки масштабованих та реактивних веб-застосунків. Даний інструмент відноситься до модульного JavaScript-фреймворку та забезпечує розбиття додатків на незалежні модулі, які в подальшому можна комбінувати. Незалежні модулі можна організувати в застосунки за допомогою API. Frint.js застосовуються у випадку проєктування великого застосунку, де потрібно керувати мікрофронтендами. Даний JavaScript-фреймворк не прив’язує розробника до конкретної бібліотеки фронтенд-проєктування, однак забезпечує хорошу інтеграцію з React. До переваг фреймворку Frint.js відносять [16]:

- реактивність та спрощена асинхронна комунікація;
- зручна робота з великою командою за рахунок модульності;
- полегшене тестування;
- спрощує розбиття бандлів на частини;
- значна економія ресурсів.

Недоліками цього JavaScript-фреймворку називають:

- складне навчання (потрібно знати RxJS на хорошому рівні);
- незначна спільнота та мала кількість документації;
- проєкт оновлюється рідко;
- не достатньо реалізована підтримка з сучасними фреймворками типу Vue.js та ін;
- не підходить для невеликих чи середніх проєктів.

Отже Frint.js – це достатньо хороший інструмент, який добре

підходить для забезпечення модульності та реактивності у веб-застосунках (особливо з застарілими додатками), однак для проєктування системи з нуля варто розглянути сучасніші альтернативи, оскільки в подальшому можуть виникнути проблеми з підтримкою. Також даний фреймворк не підійде, якщо мікрофронтенди будувати з використання Vue.js.

Piral – фреймворк з відкритим програмним кодом, який використовують для створення мікрофронтендів. Цей фреймворк дозволяє проєктувати складні та масштабовані веб-застосунки, в яких оболонка інтегрує pilets (незалежні модулі). Piral досить корисний для тих систем, де потрібно комбінувати React, Vue та навіть Angular. Це все можна інтегрувати в одному інтерфейсі і не виникне жодних конфліктів. Фреймворк Piral легкий, але здатний розширюватись через свою екосистему. На рис. 2.3 показано діаграму, яка демонструє вигляд архітектури фреймворку Piral [17].

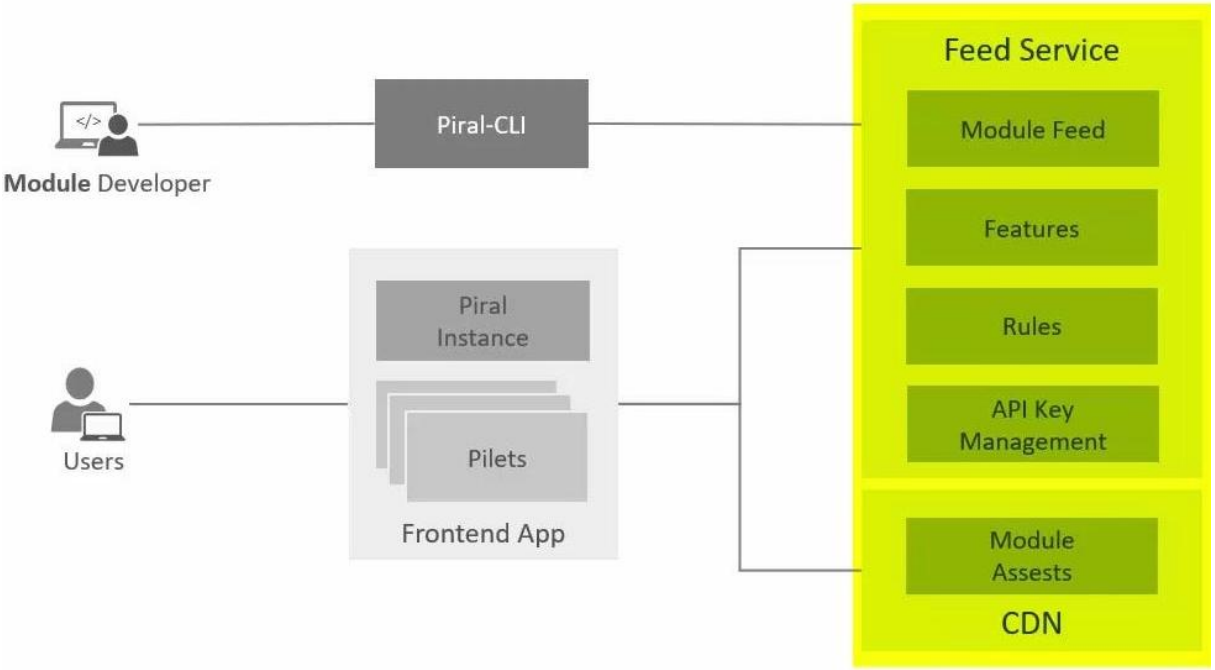


Рис. 2.3. Вигляд архітектури з використанням Piral [17]

До переваг фреймворку Piral відносять:

– можна використовувати велику кількість технологій в одному

застосунку (React/Vue/Angular);

- містить зручні інструменти для розробника (наприклад, для розгортання);
- підтримує значні можливості при розробці на React;
- дозволяє динамічно підвантажувати модулі завдяки підтримці lazy-loading;
- зручна інтеграція API;
- модулі ізольовані між собою, що позитивно впливає на безпеку веб-застосунку.

Також варто розглянути недоліки:

- складна конфігурація оболонки для застосунку;
- значна залежність від React (інші популярні фреймворки мають гіршу підтримку);
- підходить для великих тільки для великих застосунків;
- невелика спільнота розробників;

Отже, фреймворк Piral є одним із найкращий рішень для побудови сучасних мікрофронтендів та забезпечить хороший досвід у процесі розробки. Він дозволяє командам працювати автономно та продуктивно. Також даний фреймворк містить зручні та корисні інструменти, які значною мірою спрощують процес розробки. Однак такі недоліки як велика залежність від екосистеми React та складність конфігурації оболонки змушують здійснити вибір на користь іншої платформи.

Аналіз інструментів та технологій впровадження мікрофронтендів включав дослідження популярних платформ, їх переваги та недоліки. Тому для підведення підсумку побудовано порівняльну характеристику проаналізованих платформ (табл. 2.1).

Таблиця 2.1

Порівняльна таблиця проаналізованих платформ

Критерій, за яким порівнюється	Проаналізована платформа			
	Single-SPA	Luigi	Frint.js	Piral
Рівень простоти інтеграції	Середній	Високий	Складний	Середній
Рівень сучасних фреймворків	Високий	Обмежений	Частковий	Найбільша підтримка для React
Рівень гнучкості конфігурації	Високий	Середній	Середній	Середній
Рівень динамічного завантаження модулів	Високий	Частковий	Не реалізує	Високий
Рівень безпеки	Середній	Високий	Середній (в найближчому майбутньому може погіршитись)	Середній
Рівень документації та спільноти	Високий	Обмежений	Дуже слабкий	Середній

З наведеної табл. 2.1 можна побачити, що тільки платформа Single-SPA максимально відповідає всім критеріям, а отже здійснено вибір на її користь. Single-SPA – найоптимальніша платформа для проєктування веб-

застосунку з використанням мікрофронтендної архітектурі, оскільки забезпечить гнучку інтеграцію таких фреймворків як Vue + React, а також дає можливість експериментувати з різноманітними інструментами. Це значить, що в Single-SPA немає жорсткої прив'язки до певних технологій, що у свою чергу дає значні можливості розробникам для експериментів. Також обрана платформа надає хорошу документацію, а у глобальній мережі можна знайти велику кількість реалізованих завдань.

2.2 Підхід інтеграції мікрофронтендів за допомогою Webpack та маршрутизації в Single-SPA

Оскільки головною задачею мікрофронтендної розробки є незалежна інтеграція окремих частин, а також те, що кожен модуль має свою логіку роботи, стилі чи залежності, то для забезпечення виконання всіх цих умов – слід виконати правильне підключення та налаштування мікрофронтендів. Хорошим та ефективним рішення для виконання такої інтеграції є платформа Single-SPA разом з інструментом Webpack. Платформа відповідає за систему маршрутизації та керування мікрофронтендами, а інструмент Webpack – за збірку та оптимізацію. За допомогою збирача розробник має можливість розділити програмний код на окремі програмні незалежні частинки та виконувати їх динамічне завантаження у веб-браузер. У веб-застосунках, де впроваджені мікрофронтенди, Webpack має три основні функції [18]:

- надає для кожної фронтенд-частини власний конфігураційний файл, який виконує ізоляцію коду та гарантує відсутність конфліктів між стилями або бібліотеками;
- виконую об'єднання модулів, кешування, а також техніку оптимізації lazy-loading. Такий підхід надзвичайно корисний у процесі проєктування та розгортання масштабований SPA;
- дозволяє завантажувати лише потрібні частини в потрібний

момент (динамічне завантаження компонентів зменшує час першого відкриття веб-сторінки застосунку).

Кожен мікрофронтенд відповідає за свою частину інтерфейсу (модуля), а активація цієї частини в браузері залежить від вибраного маршруту. У цьому процесі платформа Single-SPA є центральним контролером, яка вмикає чи вимикає певний мікрофронтенд у потрібний момент. Даний підхід до проєктування мікрофронтендів – на основі маршрутизації – один із найпростіших у розробці, оскільки не потребує глибокої інтеграції з іншими модулями системи.

Веб-застосунки, які побудовані за допомогою Single-SPA та Webpack, мають кожен свою внутрішню структуру, а також створюється головна конфігурація, яка відповідає за підключення частинок та маршрутизацію між ними. Такий підхід дозволяє створити багато модулів, а кожен із них може бути на своєму фреймворку (на React, Vue, Angular чи навіть на чистій мові програмування JavaScript). Підхід до інтеграції мікрофронтендів за допомогою Webpack та маршрутизації в Single-SPA має ряд переваг, до яких належать:

- відсутні обмеження на використання технологій (можна комбінувати будь-які фреймворки в межах одного застосунку);
- проста реалізація системи навіть при використанні малих команд для розробки завдяки відсутності складних механізмів взаємодії);
- незалежна розробка, оновлення та розгортання кожного модуля;
- логіка навігації за рахунок маршрутизації достатньо проста;
- легке масштабування без потреби змін в існуючих модулях.

Даний підхід незважаючи на великий ряд переваг, має деякі недоліки:

- можуть виникнути проблеми з глобальний стилями;
- важке керування спільними залежностями при різних версіях однакових бібліотек;
- якщо потрібна значна кількість маршрутів, то зростає складність

структури.

Отже, підхід маршрутизації за допомогою Single-SPA та Webpack в межах експериментального проєктування веб-застосунку є доцільним, оскільки саме він зможе забезпечити баланс під час розробки однією людиною. Даний підхід дозволить продемонструвати такі принципі незалежного проєктування як модульність, маршрутизація та збір окремих компонентів у єдиний веб-застосунок.

2.3 Інтеграція різних фреймворків: Vue + React

Сучасна тенденція, коли в межах одного веб-застосунку використовуються кілька фреймворків клієнтської частини, з'явилась не спроста. Розробники прагнуть використовувати найбільш корисні інструменти кожної технології в одному проєкті, щоб в подальшому дозволить створити гнучкий та незалежно масштабований застосунок.

Однією з найпоширеніших комбінацій застосування кількох веб-фреймворків для розробки мікрофронтедів становить поєднання Vue.js та React. Це сучасні та провідні бібліотеки, які використовують для побудови динамічний односторінкових веб-застосунків. Хоч вони частково схожі між собою, однак містять відмінності в архітектурі та наборі інструментів.

Також варто згадати, що основною ідеєю поєднання кількох фреймворків є реалізація окремих частин інтерфейсу у вигляді незалежний модулів. Компоненти Vue можна використати для реалізації тієї користувацької частини, в якій буде гнучке зв'язування даних та потреба в реактивності, а компоненти фреймворку React реалізовуватимуть складнішу логіку – логіку стану [19].

Успішна інтеграція цих двох веб-фреймворків можлива тільки тоді, коли буде узгоджений їх життєвий цикл. Такий цикл, з використанням платформи Single-SPA, виконується за допомогою спеціальних точок – функцій підключення (bootstrap mount, unmount та bootstrap). Ці функції

відповідають за підготовку, відображення та вилучення окремої частинки фронтенду з DOM. Для прикладу: користувач виконує певну дію на сторінці веб-застосунку, ця дія виконує активацію конкретного мікрофронтенду та запускає його у потрібні області. Після того як користувач покинув цю область – йде процес вилучення задля звільнення ресурсів. Даний підхід демонструє оптимальний вибір фреймворків, з яким буде складатись експериментальний застосунок в межах роботи над цією магістерською дисертацією.

2.4 Механізми комунікації між мікрофронтендами

Незважаючи на те, що кожен модуль клієнтської частини функціонуватиме незалежно та автономно, з часом, під час масштабування, виникає потреба в забезпеченні комунікації між ними, а також у передачі даних. Таку комунікацію між мікрофронтендами поділяють на два рівні: локальна (між компонентами, що знаходяться в межах конкретного фреймворку) та глобальну (між компонентами, що існують в різних фреймворках, чи навіть на різних серверах). До найпоширеніших підходів для реалізації комунікації належать: модель на основі подій, централізовані сховища стану, API, кастомні події (Custom Events), канал для повідомлень (BroadcastChannel API) та реактивні потоки для спільного стану (створені за допомогою RxJS).

Модель на основі подій є найпростішою та часто використовується для обміну даними між частинками фронтендів. Така комунікація здійснюється за допомогою відправлення та оброблення подій. Найбільшою перевагою такої моделі є простота в реалізація, оскільки події можна реалізувати не тільки за допомогою вбудований можливостей веб-браузера, а також через сторонні інструменти, які значною мірою спростять керування. У великих системах даний підхід може ускладнити контроль над подіями [20].

Підхід централізованого сховища заснований на взаємодії мікрофронтедів через спільну базу, яка належатиме тільки клієнтському рівню. Варто зазначити переваги такого методу – це зручність відлагодження, так як всі зміни будуть централізовано записані. Однак даний підхід містить значний недолік – може виникнути потреба в оновлення кількох мікрофронтендів. Для уникнення такого недоліку виконують розподіл стану на локальний та глобальний (використовується для обміну тільки критично важливих даних)

Також ще одним поширеним методом комунікації є комунікація через API. Під час використання такого підходу кожен мікрофронтенд виконує власну взаємодію з бекендом через REST. Перевагою такого роду комунікація є повна ізольованість модулів, однак при необхідності спільного стану на клієнтському рівні – потреба в додаткових мережевих запитах та механізмах оптимізації.

BroadcastChannel API – це браузерна технологія. Її можна використати під час розробки мікрофронтендів для комунікації між модулями, які у свою чергу завантажуватимуться як окремі застосунки. До переваг цього методу відносять простоту в реалізація, а також асинхронність. А недоліком технології є її несумісність із застарілими браузерами, а також обмежений обсяг для передачі даних.

До ще одного зручного та корисного способу комунікацій відносять кастомні події. Custom Events – це події, які створюють вручну та використовують в обміні даними між компонентами через DOM. Даний метод є досить корисним, особливо у випадках, коли веб-застосунок буде зібрано в Single-SPA сторінку. Custom Events хоч і потребує ретельної організації назв та подій, однак слугує простим рішенням. Недоліком цього методу є ускладнене відлагодження у масштабних застосунках.

Для забезпечення комунікації більш складних мікрофронтендів використовують реактивну бібліотеку RxJS. Дана бібліотека забезпечує створення глобального каналу для комунікації. Найбільшою перевагою

RxJS є чітке керування циклом підсписок мікрофронтендів та висока продуктивність, однак провадження цього методу комунікацій затребує в розробника більшого рівня технічної підготовки. Також даний метод може бути надто громізким для невеликих систем.

Висновки до розділу 2

У процесі написання другого розділу дипломної дисертації було проведено порівняльний аналіз платформ Single-SPA, Luigi, Frint.js та Piral, які надають інструменти для реалізації мікрофронтендів. Визначено, що Single-SPA є достатньо гнучким та має значну підтримку різноманітних фреймворків. Також дана платформа ідеально підходить до поступового масштабування застарілих системи завдяки інструментам для оптимізації продуктивності та незалежному підходу до розробки модулів. Однак досліджено недолік – існуючі проблеми при комунікаціях між модулями.

У той час існує фреймворк Luigi – потужний інструмент для корпоративних середовищ. Досліджено переваги: забезпечує єдиний дизайн через централізовану навігацію та теми, а також містить вбудовану авторизацію та ізоляцію, які обмежують виникнення конфліктів. Але слід зазначити, що такі недоліки як менша гнучкість у порівнянні з Single-SPA, складне навчання та погана підтримка змушують розробника розглянути інші платформи для проєктування мікрофронтендів.

Фреймворк Frint.js показав себе у якості потужного інструменту для масштабування застарілих застосунків. Однак недолік цього фреймворку полягає в тому, що його не можна використати в побудові мікрофронтендів з використання Vue.js.

Також проведений аналіз показав, що фреймворк Piral містить зручні та корисні інструменти для побудови сучасних мікрофронтендів. Однак такі недоліки як значна залежність від екосистеми React та складність конфігурації оболонки змушують здійснити вибір на користь іншої платформи.

У ході аналізу було обрано платформу Single-SPA. Бо вона найоптимальніша для проєктування веб-застосунку на мікрофронтендній архітектурі, оскільки забезпечить гнучку інтеграцію таких фреймворків як Vue + React, а також надасть можливість експериментувати з

різноманітними інструментами

Важливу увагу було приділено підходу інтеграції за допомогою Webpack та маршрутизації в Single-SPA. Використання Webpack забезпечить модульне завантаження елементів мікрофронтендів, а платформа Single-SPA дозволить динамічно підключати ту чи іншу частинку інтерфейсу веб-застосунку відповідно до використаного маршруту користувача.

Було розглянуто підхід до інтеграції двох фреймворків Vue та React, що у свою чергу дозволив поєднати переваги цих інструментів у розробці демонстраційного односторінкового веб-застосунка. Також під час виконання роботи було проаналізовано механізми комунікації між мікрофронтендами. Оскільки платформа буде масштабуватись та зростати, то існує потреба в узгодженій взаємодії між компонентами.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ НА ОСНОВІ МІКРОФРОНТЕНДІВ

3.1 Формування вимог та бізнес-процесів

Розглядаючи вимоги до додатку потрібно з’ясувати процеси, які будуть застосовані у кожній частині.

Загалом додаток повинен містити:

- 2 незалежно працюючі фреймворки;
- Правильно налаштований single-spa;
- Окремо створені та незалежно працюючі частини сайту (нотатник та форма);

Для частини, яка реалізовується завдяки React:

- Валідація полів (ім’я та місто містять мінімальну кількість символів, пошта перевіряється, вік від 18 до 100, місто мінімум 3 симв.);
- Пагінація елементів на кожні 3 внесені форми;
- Кнопка для очищення всіх внесених полів;
- Пошук по імені, яке вносилось в форму;
- Сортування (дата додавання, ім’я, місто, вік);
- Редагування створеної форми;
- Видалення внесеної форми.

Для частини, що створюється через Vue, потрібно реалізувати:

- Валідацію порожнього поля;
- Внесення дати є обов’язковим ;
- Включення таймера для помодоро (створюється додатково, щоб можна було виконувати завдання і виставляти таймер для відслідковування прогресу);

– Пагінація на кожні 3 нотатки;
– Визначення нотатки, як важливої;
– Перенесення завдання в прострочені;
– Редагування створеної нотатки;
– Видалення збереженого завдання;
– Можливість сортування за важливими або простроченими нотатками.

3.2 Моделювання процесів за допомогою BPMN, IDEF0

BPMN є системою умовних позначень, що використовується для моделювання бізнес-процесів. У аспекті цією роботи, необхідно показати, як користувач взаємодіятиме з реалізованою системою (рис. 3.1) [21].

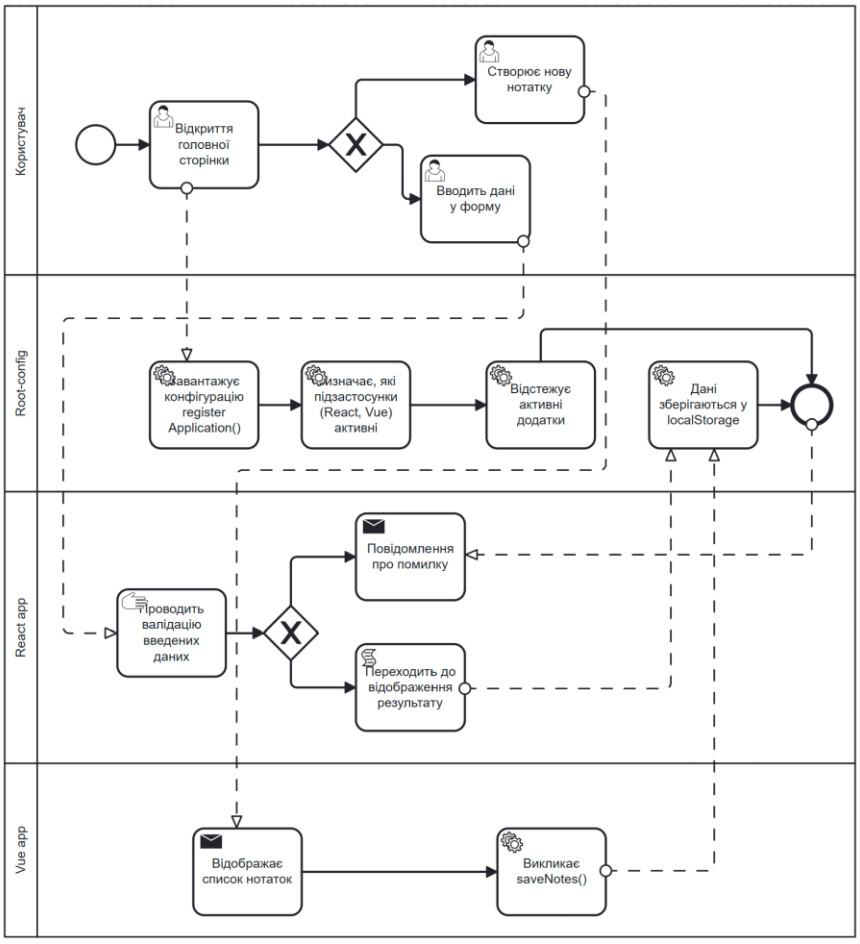


Рис. 3.1. BPMN діаграма роботи процесу

IDEF0 надає можливість детально оглянути функціональні блоки системи, що допоможе краще збагнути побудову мікрофронтендів.

Найперше будується контекстна діаграма IDEF0 (рис. 3.2), вона представляє, як функціонує система за допомогою масштабування, показуючи загальний шар роботи додатку [22].

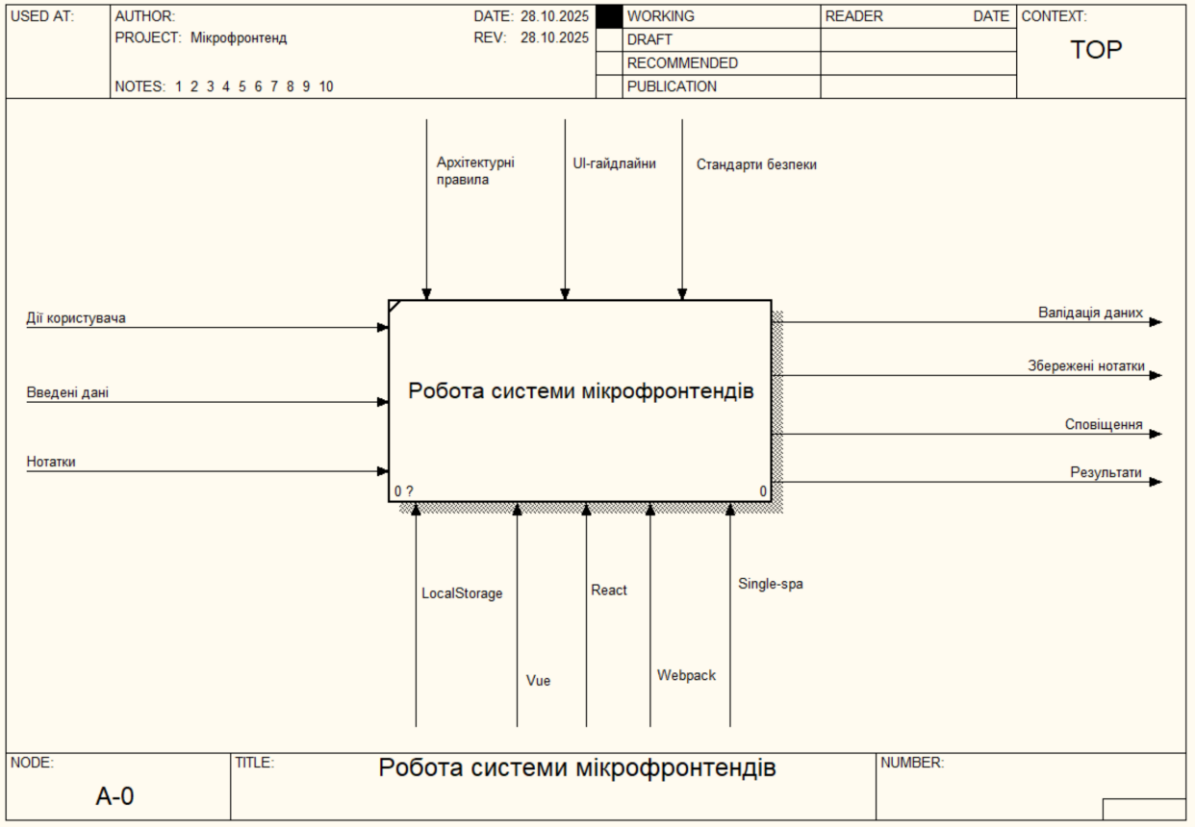


Рис. 3.2. Контекстна діаграма IDEF0

Реалізуючи загальний модуль діаграми, необхідно провести декомпозицію першого рівня, тут необхідно розкласти контекстну діаграму на 3 блоки (рис. 3.3).

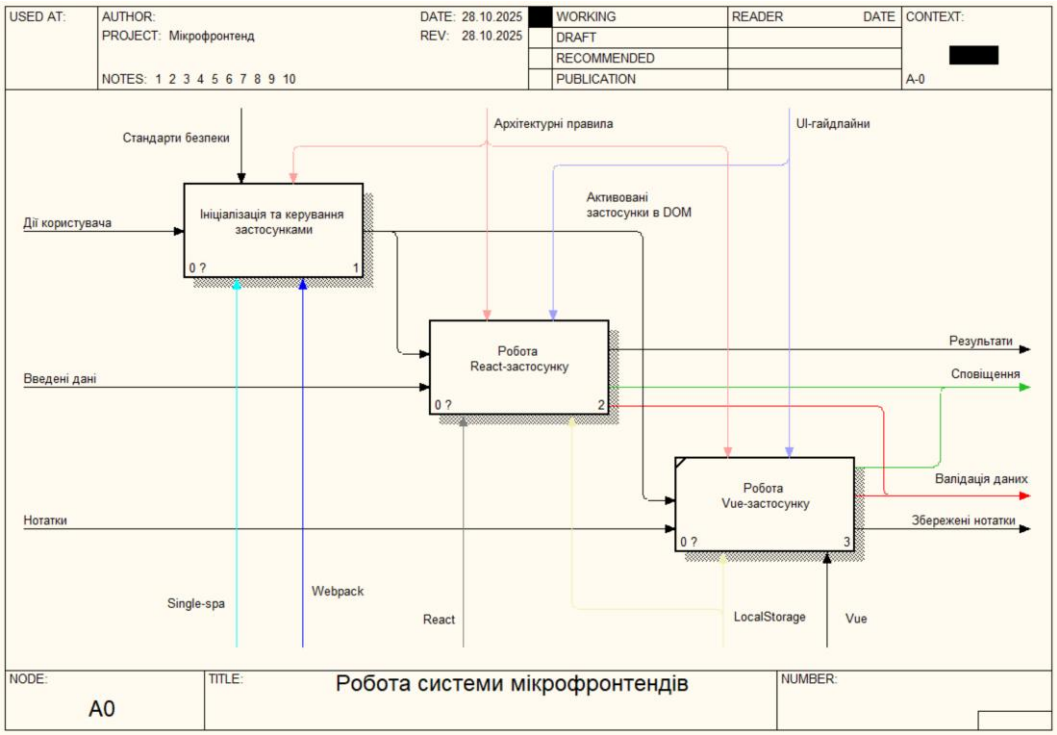


Рис. 3.3. Діаграма декомпозиції першого рівня нотації IDEF0

Останнім кроком є реалізація діаграм декомпозиції другого рівня, кожен з 3 блоків попередньої моделі розкладається на додаткові процеси, що детально показують роботу кожної з функцій (рис. 3.4-3.6).

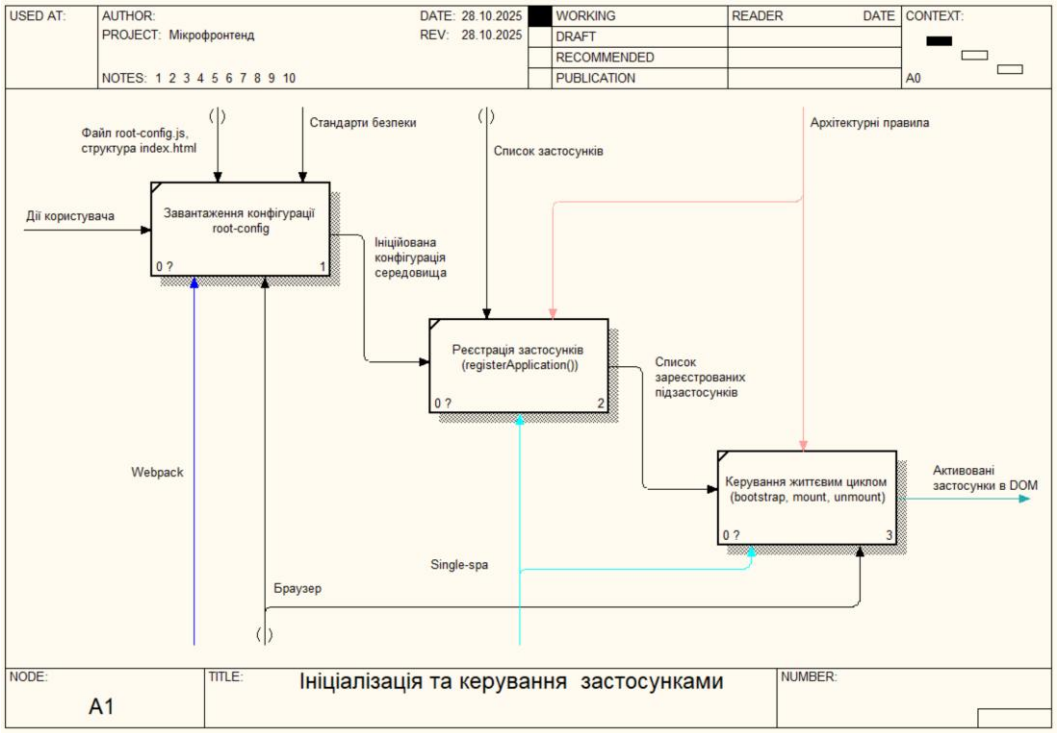


Рис. 3.4. Діаграми декомпозиції другого рівня нотації IDEF0 (Ч.1)

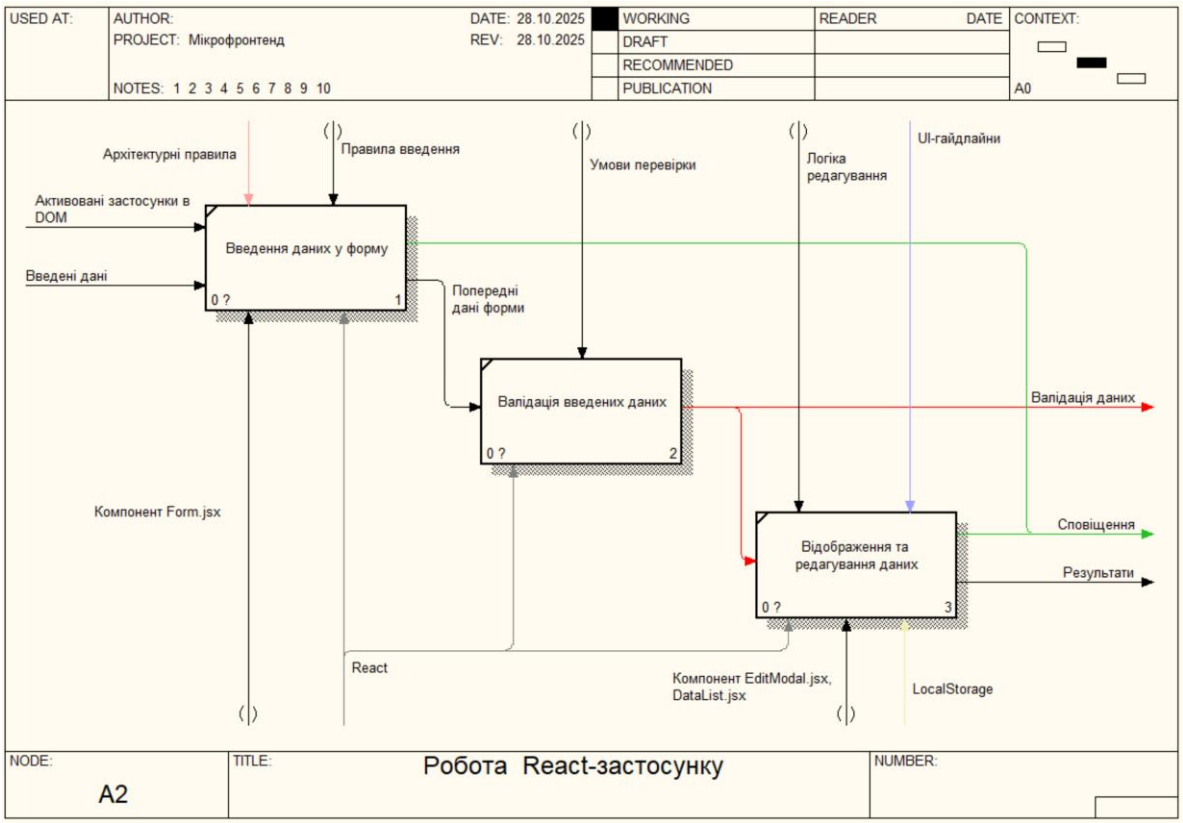


Рис. 3.5. Діаграми декомпозиції другого рівня нотації IDEF0 (Ч.2)

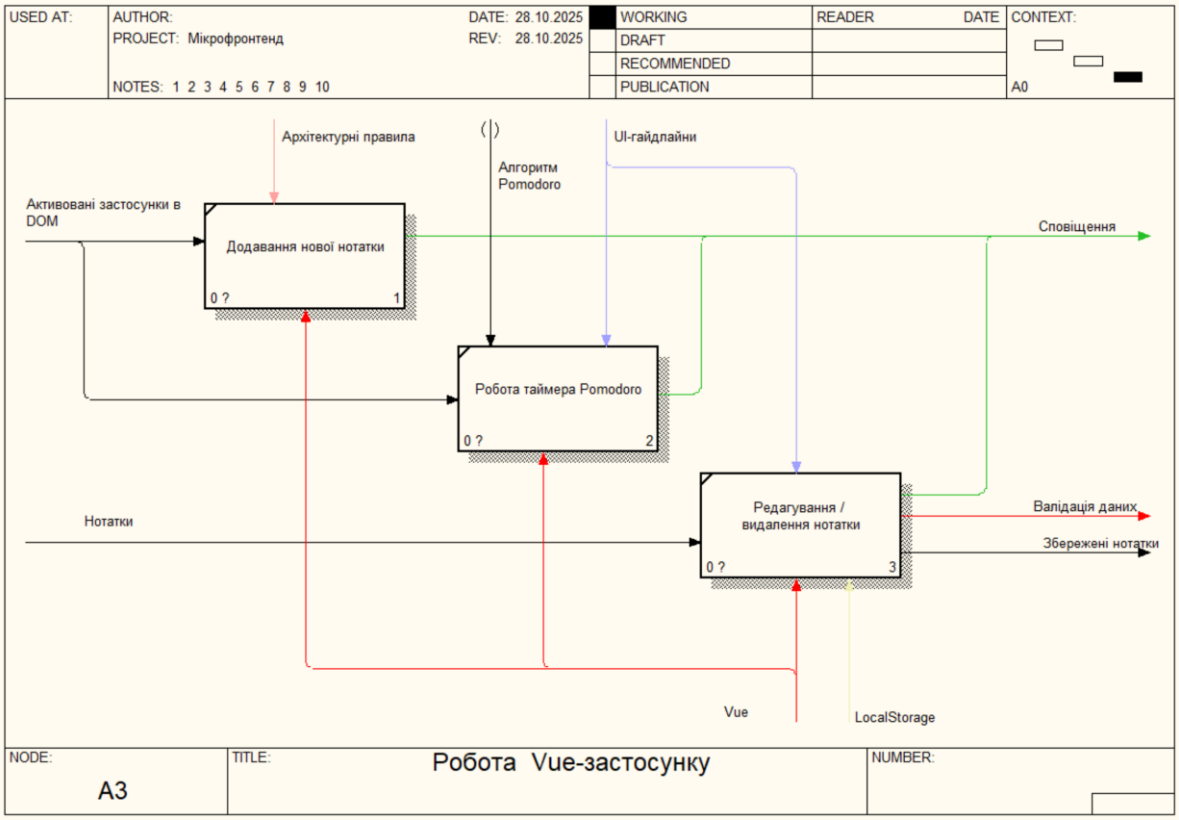


Рис. 3.6. Діаграми декомпозиції другого рівня нотації IDEF0 (Ч.3)

Дерево структури блоків допоможе підсумувати попередні діаграми у вигляді виведення тільки процесів, без зв’язків та стрілок (рис. 3.7).

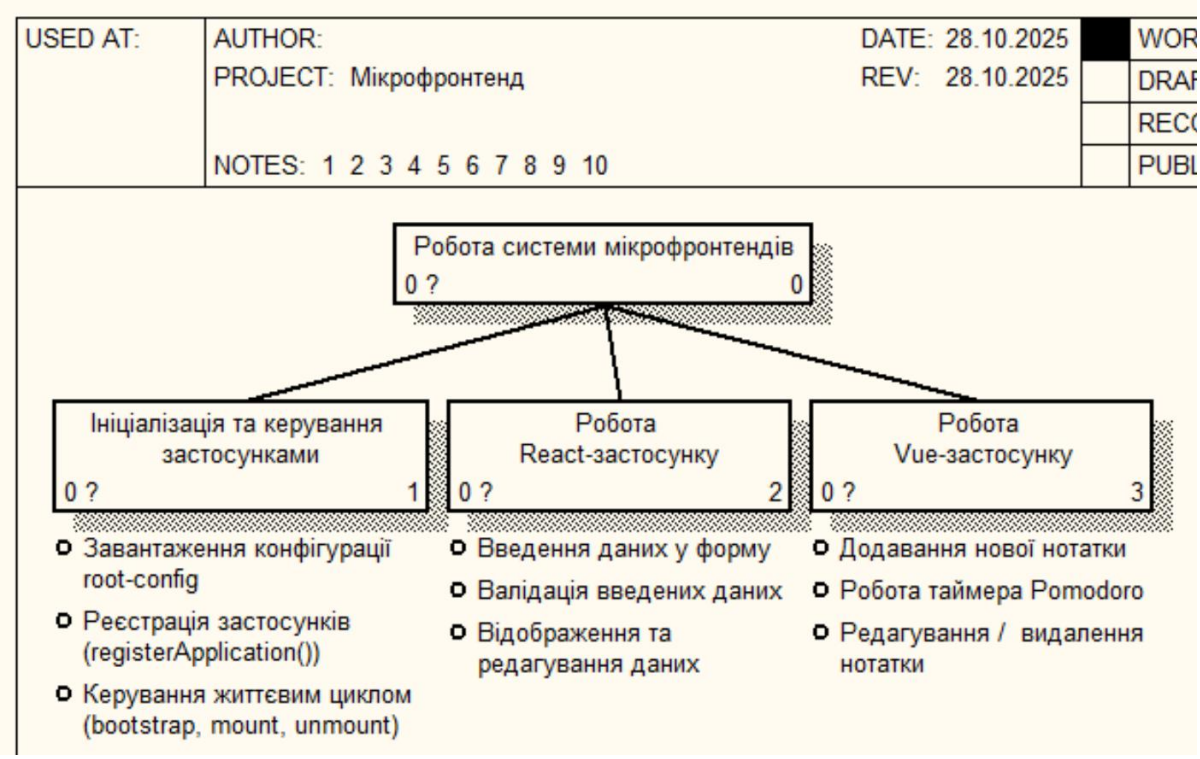


Рис. 3.7. Структура у вигляді дерева

IDEF3 описує послідовність процесів з сценаріями взаємодіями їх компонентів. Його декомпозиція пройшла через контекстну діаграму, щоб показати всю систему (рис. 3.8) [23].

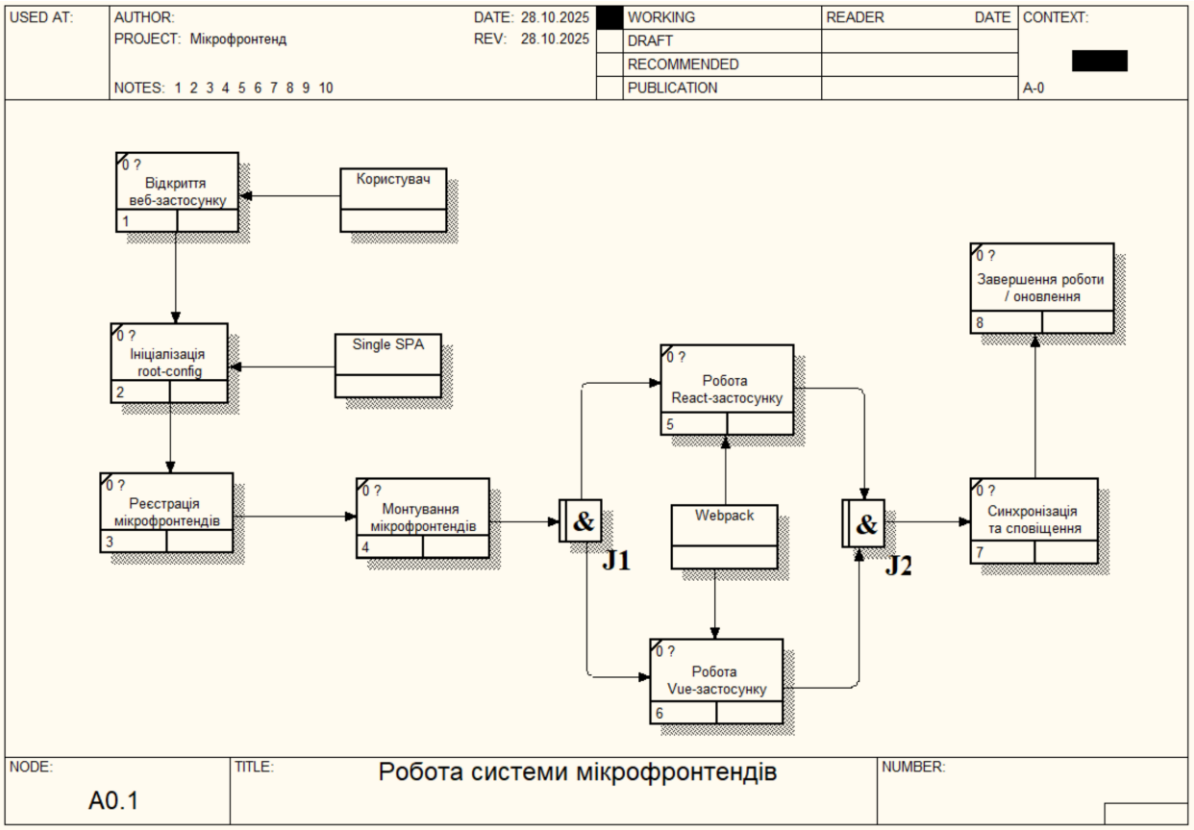


Рис. 3.8. IDEF3 системи

3.3 Проєктування архітектури додатка

Загалом архітектура проєкту ділиться на 3 логічні частини (рис. 3.9):

Root-config: центральна частина додатку, що з’єднує та керує готовими частинами сторінок. Відповідає за виведення застосунків на спільній HTML-сторінці.

React-app: частина додатку, яка повністю побудована завдяки фреймворку React, містить в собі форму для заповнення даних.

Vue-app: останній елемент, що зберігає в собі код написаний завдяки Vue.js, зберігає частину проєкту, що відповідає за створення та керування нотатками.

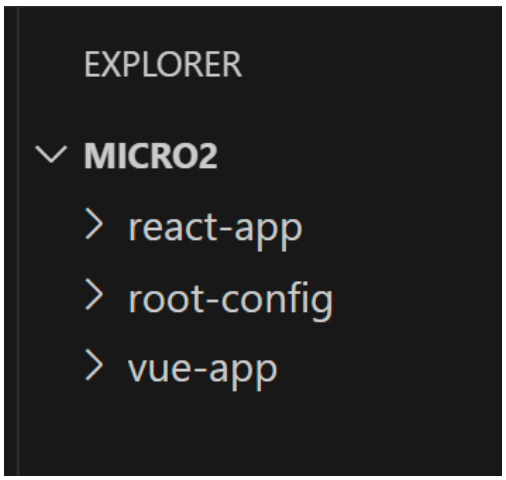


Рис. 3.9. Реалізовані каталоги проєкту

Наступним кроком реалізуються UML діаграми, які описують різноманітні елементи системи.

Діаграма deployment – допомагає з тим, щоб побачити компоненті, які використовуються при запуску хостингу застосунку (рис. 3.10).

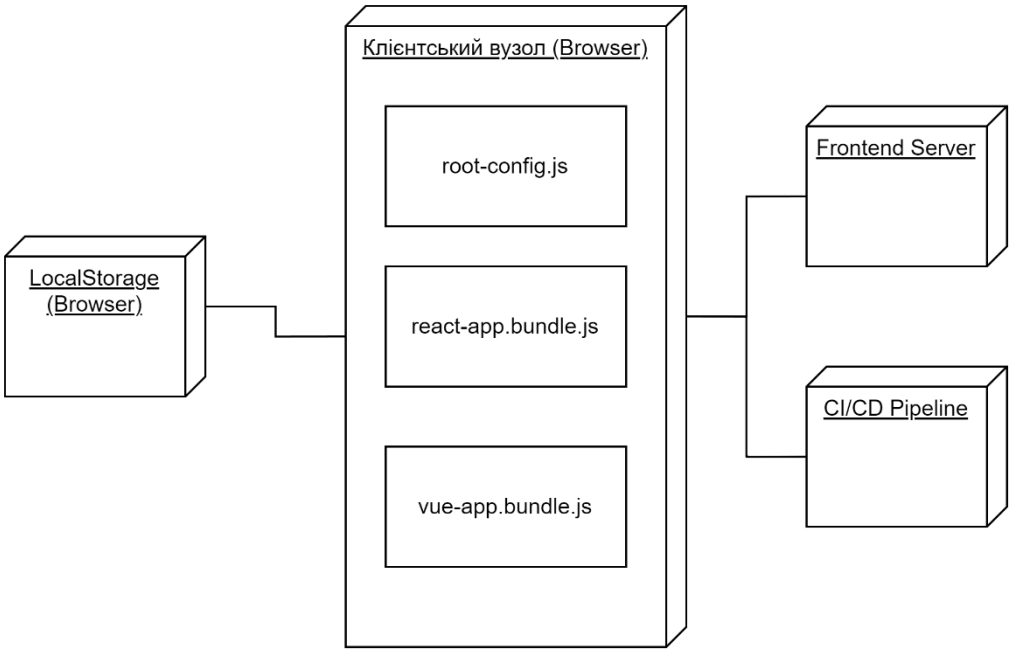


Рис. 3.10. Deployment diagram [24]

Діаграма компонентів – описує з яких компонентів складається система (рис. 3.11).

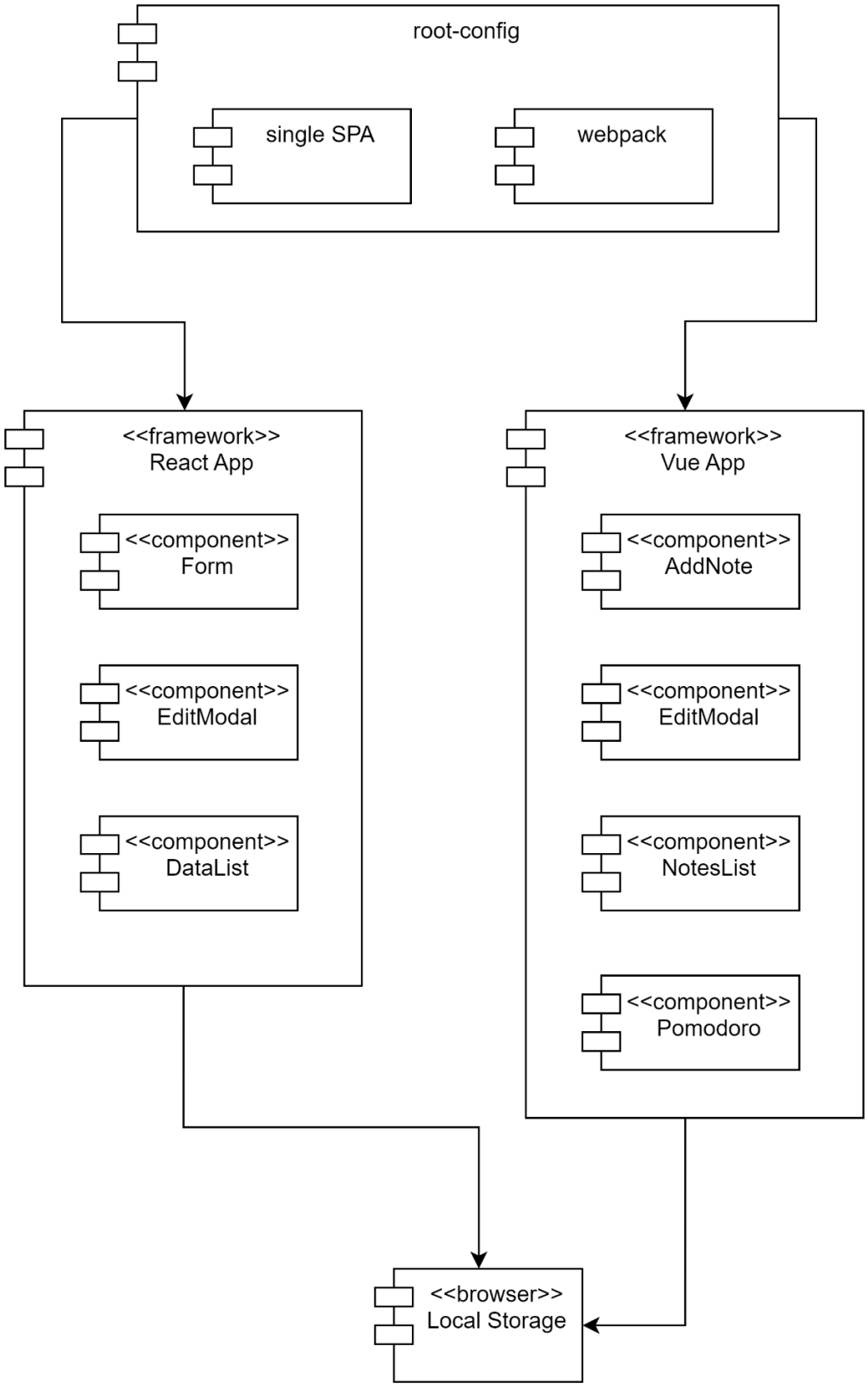


Рис. 3.11. Component diagram [25]

Діаграма послідовності – описує послідовність роботи між створеними програмними одиницями проєкту (рис. 3.12).

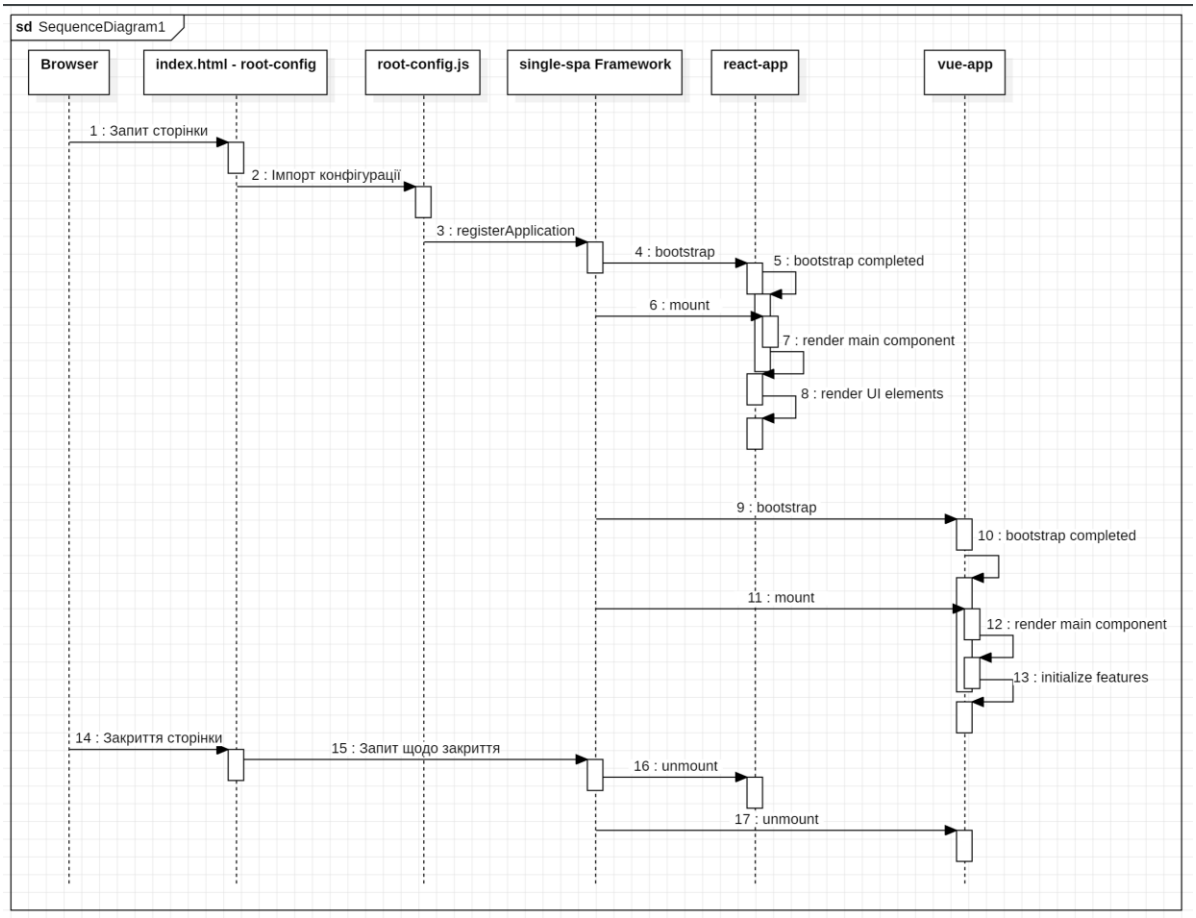


Рис. 3.12. Sequence diagram [26]

3.4 Реалізація з використанням Single-SPA

Для практичної реалізації кореневої частини проєкту, необхідно створити структуру, що допоможе керувати частинами, які будуть виводитись на кінцеву сторінку (рис. 3.13).

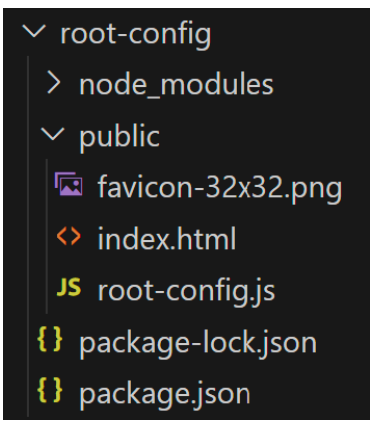


Рис. 3.13. Структура root-config [27]

Певні елементи, як node_modules, package-lock.jsoin, package.json є стандартними та застосовуються в усіх проєктах, що потребують просунутих налаштувань.

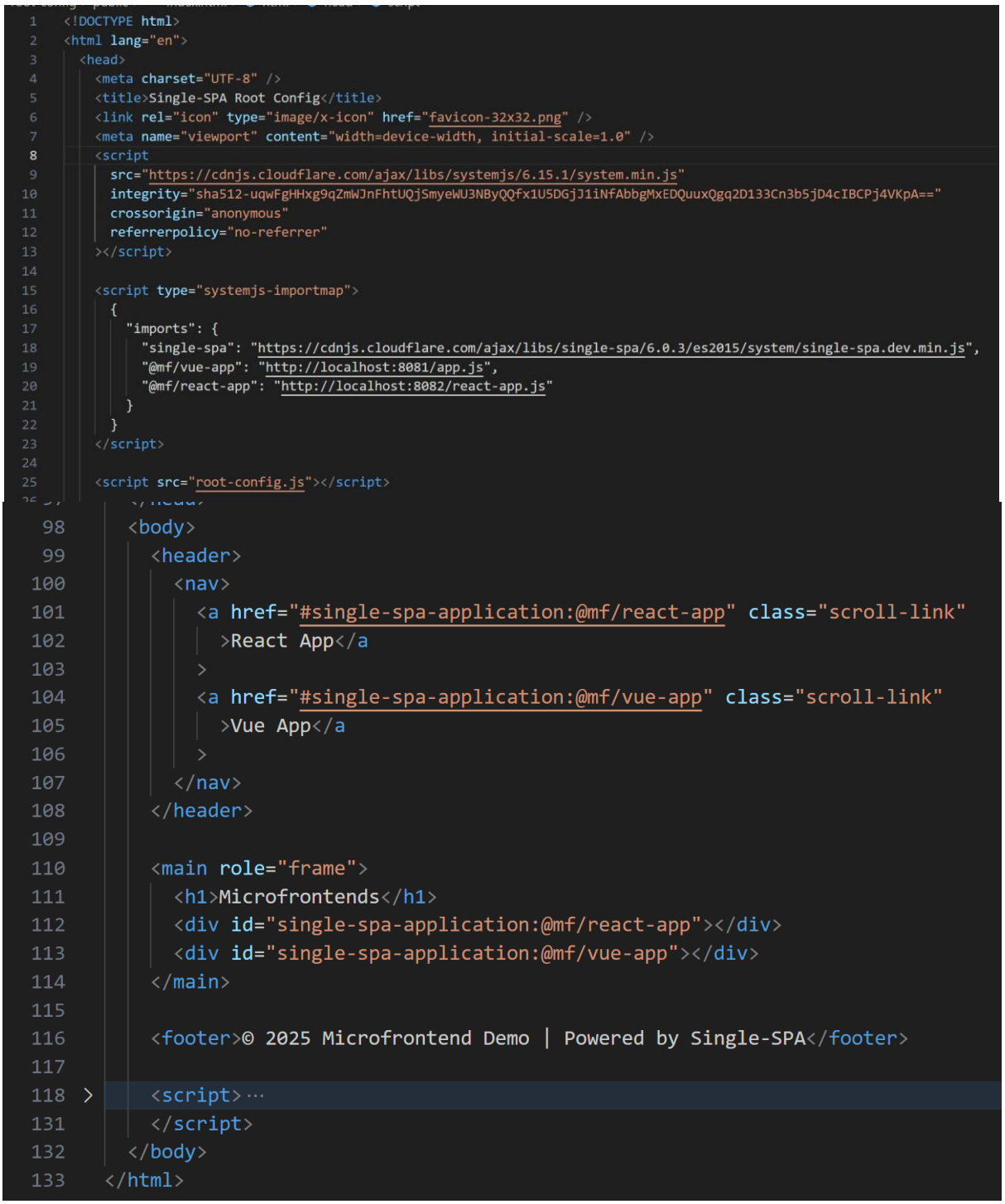
Root-config.js – реєструє логічні частини та обробляє посилання, які були сформовані після запуску фреймворків. Для цього він надає кожному з них ім’я, спеціальний індикатор та коли він є активним (див. Додаток А, рис. 3.14).

root-config > public > JS root-config.js > ...

```
System.import("single-spa").then((singleSpa) => {
  const { registerApplication, start } = singleSpa;

  registerApplication({
    name: "@mf/vue-app",
    app: () => System.import("@mf/vue-app"),
    activeWhen: ["/"],
  });

  registerApplication({
    name: "@mf/react-app",
    app: () => System.import("@mf/react-app"),
    activeWhen: ["/"],
  });

  start();
  console.log("✅ single-spa started");
});

```

Рис. 3.14. Код файлу Root-config.js [28]

Index.html – імпортує присвоєні імена та зв’язує їх з спеціальними посиланнями за якими працюють практичні частини сторінки. Додатково вибираючи місця розташування для елементів фронтенду (рис. 3.15).



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Single-SPA Root Config</title>
    <link rel="icon" type="image/x-icon" href="favicon-32x32.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script
      src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/6.15.1/system.min.js"
      integrity="sha512-uqwFgHHxg9qZmWJnFhtUQjSmyeWU3NByQQfx1U5DGjJ1iNfAbbgMxEDQuuxQgq2D133Cn3b5jD4cIBCPj4VKpA=="
      crossorigin="anonymous"
      referrerpolicy="no-referrer"
    ></script>

    <script type="systemjs-importmap">
      {
        "imports": {
          "single-spa": "https://cdnjs.cloudflare.com/ajax/libs/single-spa/6.0.3/es2015/system/single-spa.dev.min.js",
          "@mf/vue-app": "http://localhost:8081/app.js",
          "@mf/react-app": "http://localhost:8082/react-app.js"
        }
      }
    </script>

    <script src="root-config.js"></script>
  </head>
  <body>
    <header>
      <nav>
        <a href="#single-spa-application:@mf/react-app" class="scroll-link"
          >React App</a
        >
        <a href="#single-spa-application:@mf/vue-app" class="scroll-link"
          >Vue App</a
        >
      </nav>
    </header>

    <main role="frame">
      <h1>Microfrontends</h1>
      <div id="single-spa-application:@mf/react-app"></div>
      <div id="single-spa-application:@mf/vue-app"></div>
    </main>

    <footer>© 2025 Microfrontend Demo | Powered by Single-SPA</footer>

    <script>…
    </script>
  </body>
</html>
```

Рис. 3.15. Код файлу Index.html

3.5 Інтеграція модулів на Vue та React

Далі необхідно реалізувати два мікрофронтенди, що будуть працювати незалежно один від одного (рис. 3.16-3.17).

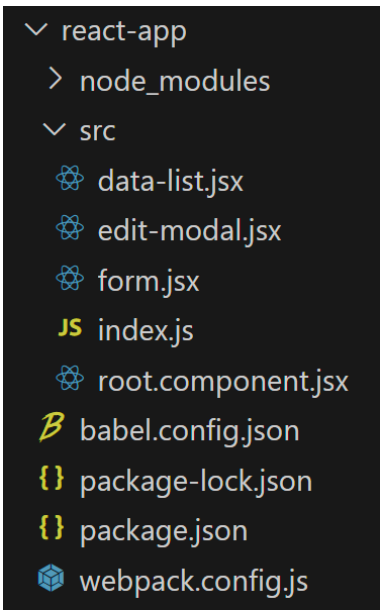


Рис. 3.16. Структура react-app [29]

Таблиця 3.1

Опис структури react-app

Елемент	Пояснення
Webpack.config.js	Конфігураційний файл Webpack, який визначає, як збирається застосунок. Тобто, які файли компілюються, які модулі підключаються, як обробляються стилі, зображення, JSX тощо.
Root.component.jsx (див. Додаток Б)	Головний компонент, який об'єднує всі частини застосунку (форму, список, модальне вікно).
Index.js	Точка входу застосунку. Експортує життєві цикли bootstrap, mount, unmount.
Data-list.jsx	Компонент, який відображає список даних, що були введені у формі.
Edit-modal.jsx	Модальне вікно для редагування вже введених даних.
Form.jsx	Компонент форми для введення нових даних. Містить поля введення, валідацію та логіку відправки або оновлення даних.

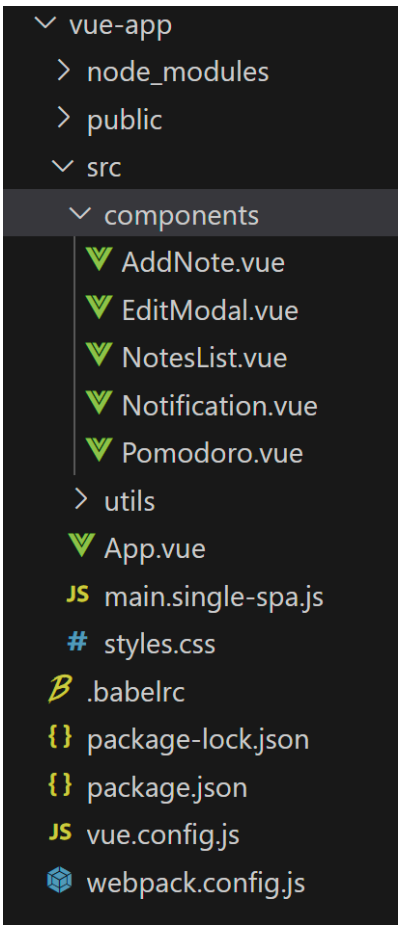


Рис. 3.17. Структура vue-app [30]

Таблиця 3.2

Опис структури vue-app

Елемент	Пояснення
Webpack.config.js	Конфігураційний файл Webpack. Визначає правила збірки Vue-компонентів, стилів, зображень.
Main.single-spa.js	Точка входу для інтеграції з мікрофронтендом. Експортує життєві цикли bootstrap, mount, unmount, які відповідають за ініціалізацію, монтування і демонтування застосунку.
App.vue (див. Додаток В)	Головний компонент застосунку, який збирає всі інші компоненти.
Addnote.vue	Компонент для додавання нової нотатки.
Editmodal.vue	Модальне вікно для редагування існуючих нотаток.

Таблиця 3.2 (продовження)

Опис структури vue-app

notesList.vue	Компонент, який виводить список усіх нотаток.
notification.vue	Компонент сповіщень. Показує повідомлення про успішне додавання, редагування або видалення нотаток.
pomodoro.vue	Компонент таймера «Помодоро» (Pomodoro timer). Реалізує функції запуску, паузи, скидання таймера і звукового сигналу після завершення циклу.

3.6 Тестування та результати впровадження

Для проведення тестування необхідно провести запуски кожного з серверів. Для цього вносяться спеціальні команди, що були присвоєні в package.json файлах.

Запуск root-config відбувається завдяки команді npm start (рис. 3.18).

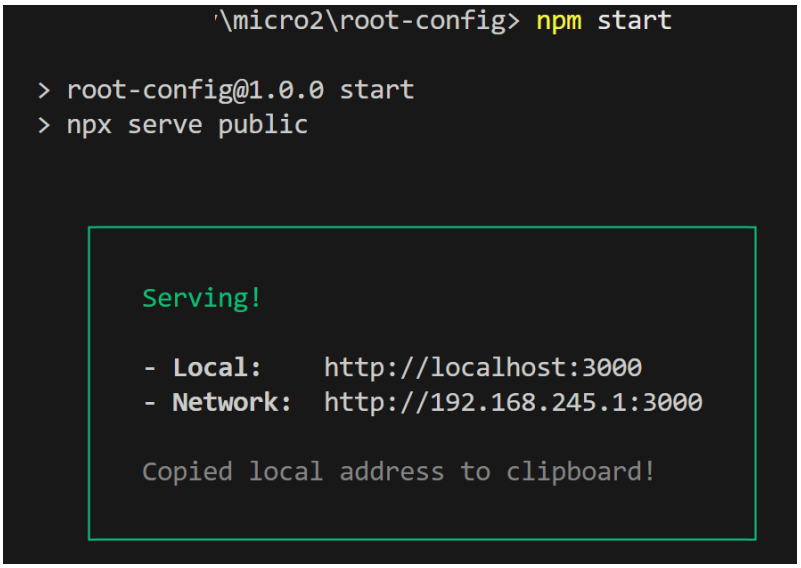


Рис. 3.18. Результат запуску загального застосунку

Запуск React-app відбувається за допомогою команди npm start (рис. 3.19) [31].

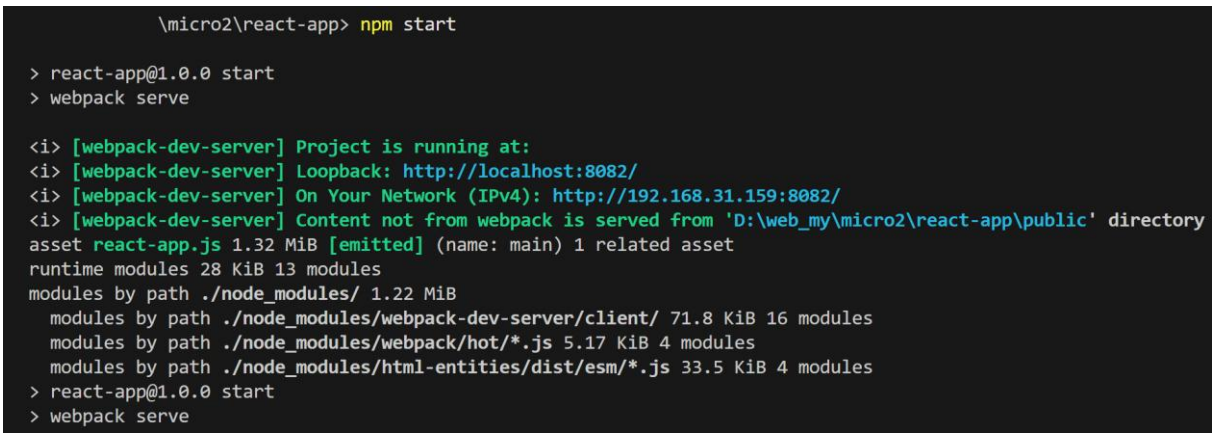


Рис. 3.19. Результат запуску React-застосунку

Запуск Vue-app відбувається завдяки команді npm run serve (рис. 3.20) [32]

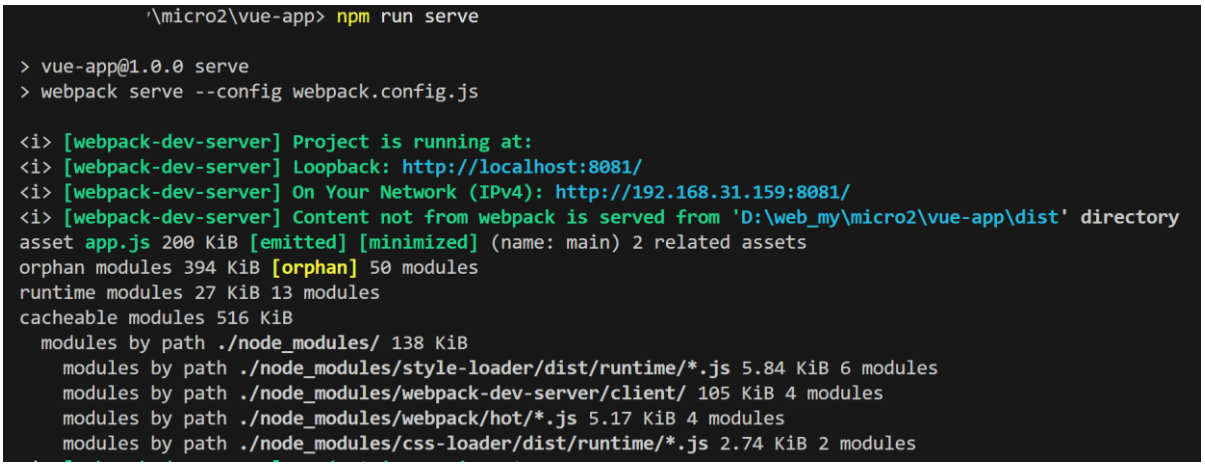


Рис. 3.20. Результат запуску Vue-застосунку

Далі для перегляду результатів необхідно перейти за посиланням, яке відобразилось в консолі root-config та можемо побачити результати створеного сайту.

Найперше доступний перегляд частини сайту, що створений завдяки React. Тут виводиться форма, у якій доступне заповнення особистих даних (рис. 3.21).

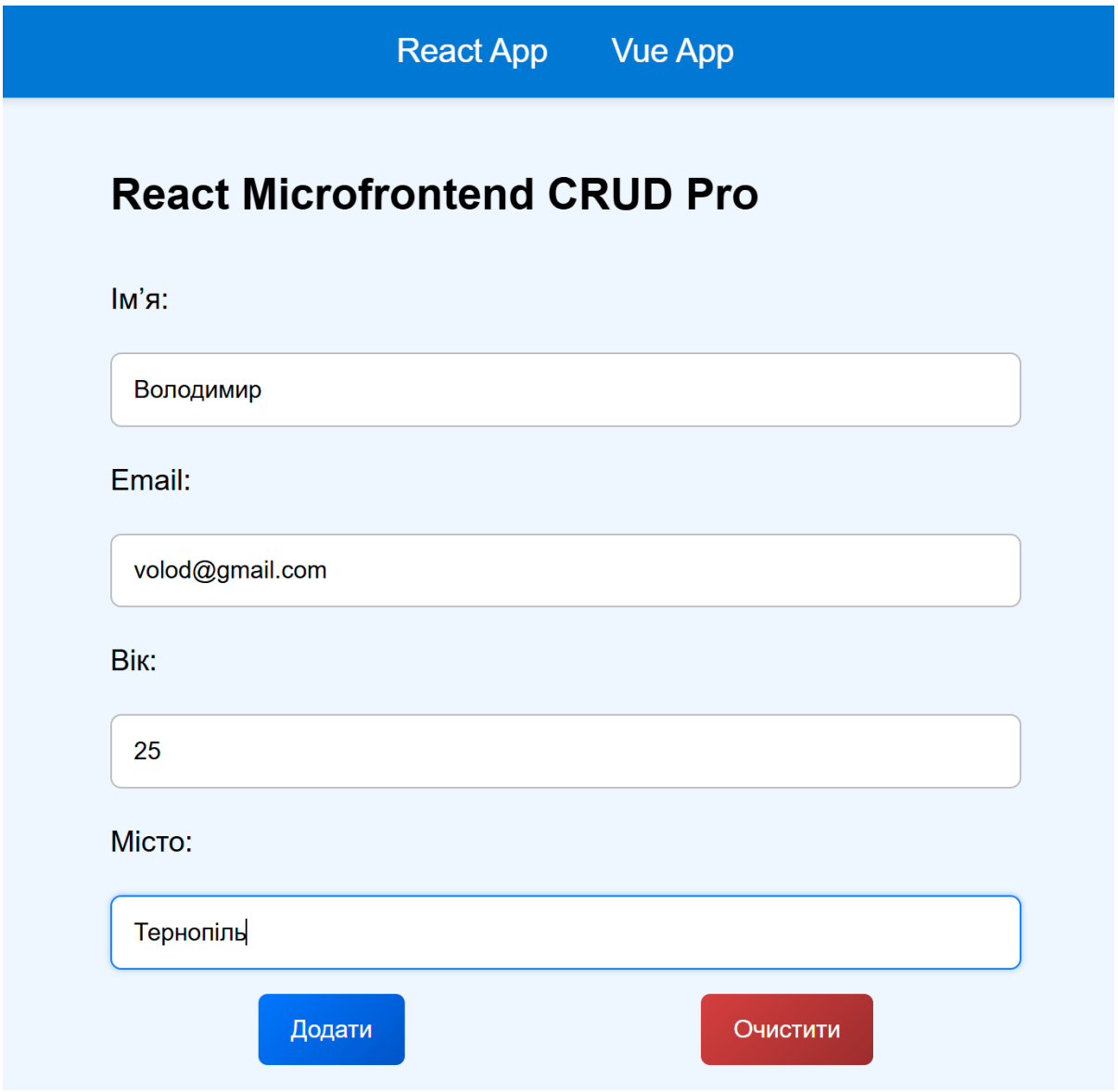


Рис. 3.21. Форма внесення даних

Після додавання можна переглянути результати внесення форм на панелі знизу. Список пагінується після кожних 3 елементів (рис. 3.22).

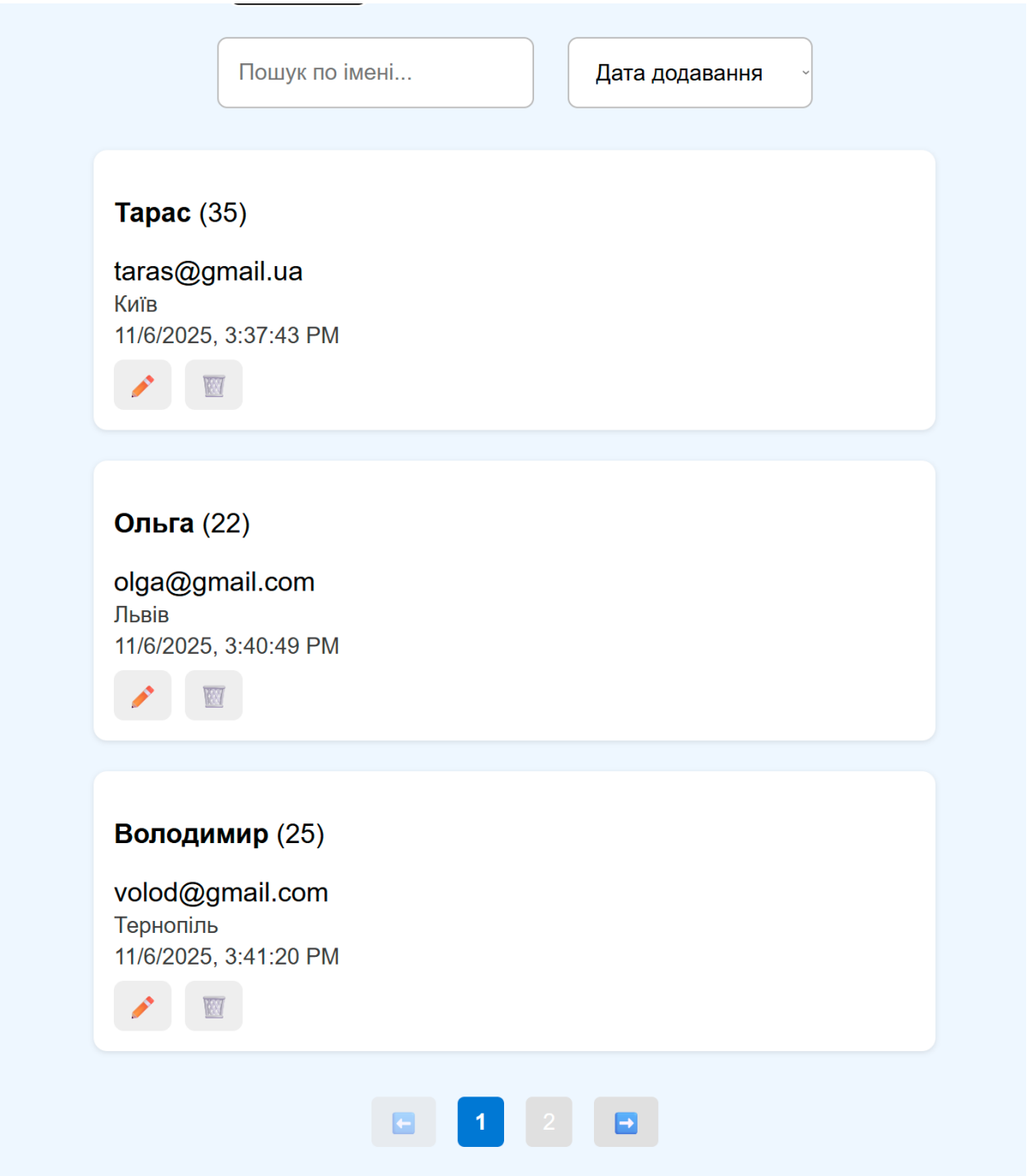


Рис. 3.22. Перегляд внесених даних

Натиснувши на іконку з олівцем, навпроти потрібного занесення даних, можливе їх редагування (рис. 3.23).

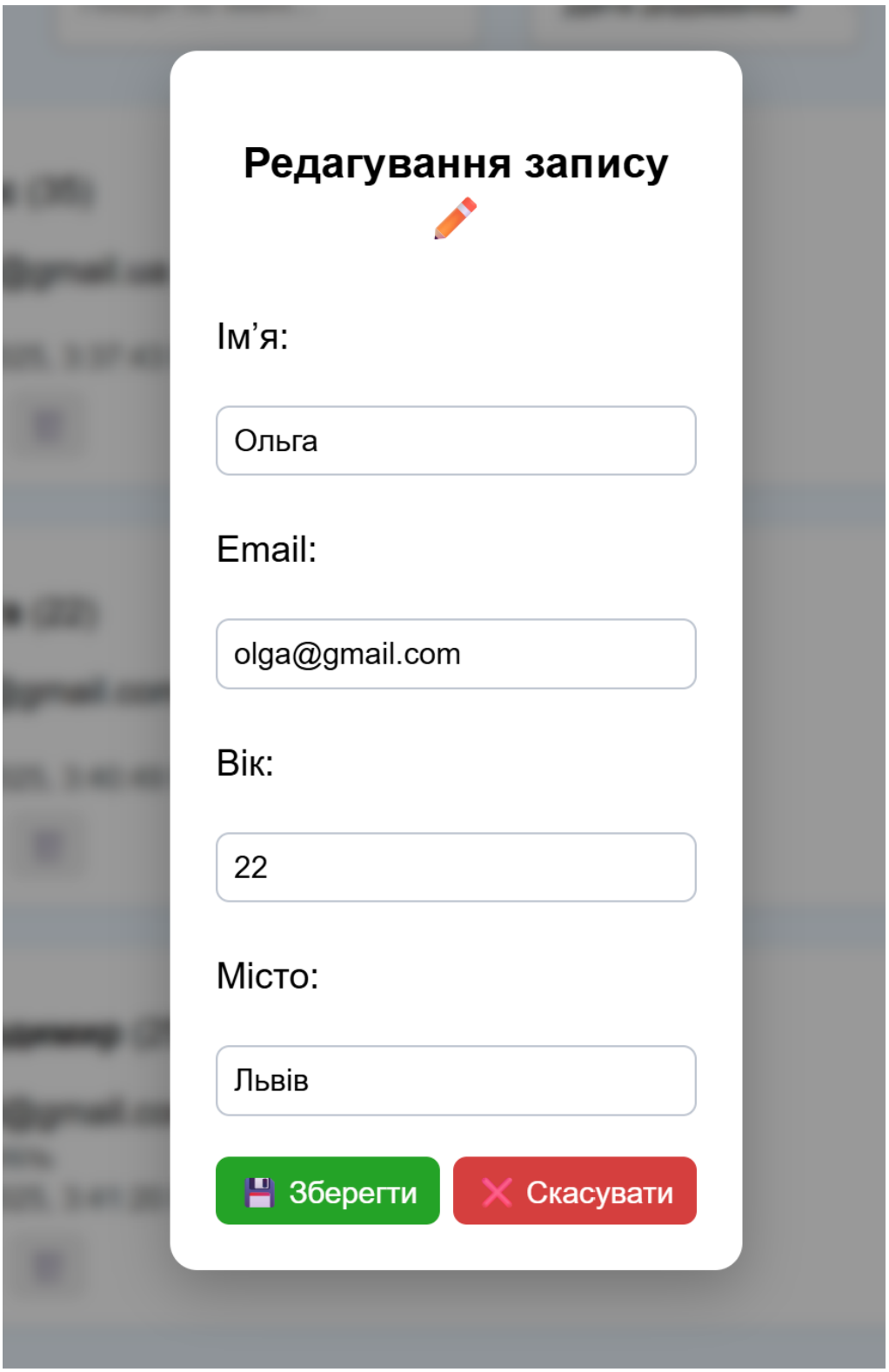


Рис. 3.23. Редагування записів

Додатково доступне сортування форм, що складається з: імені, віку, міста та дати внесення (рис. 3.24).

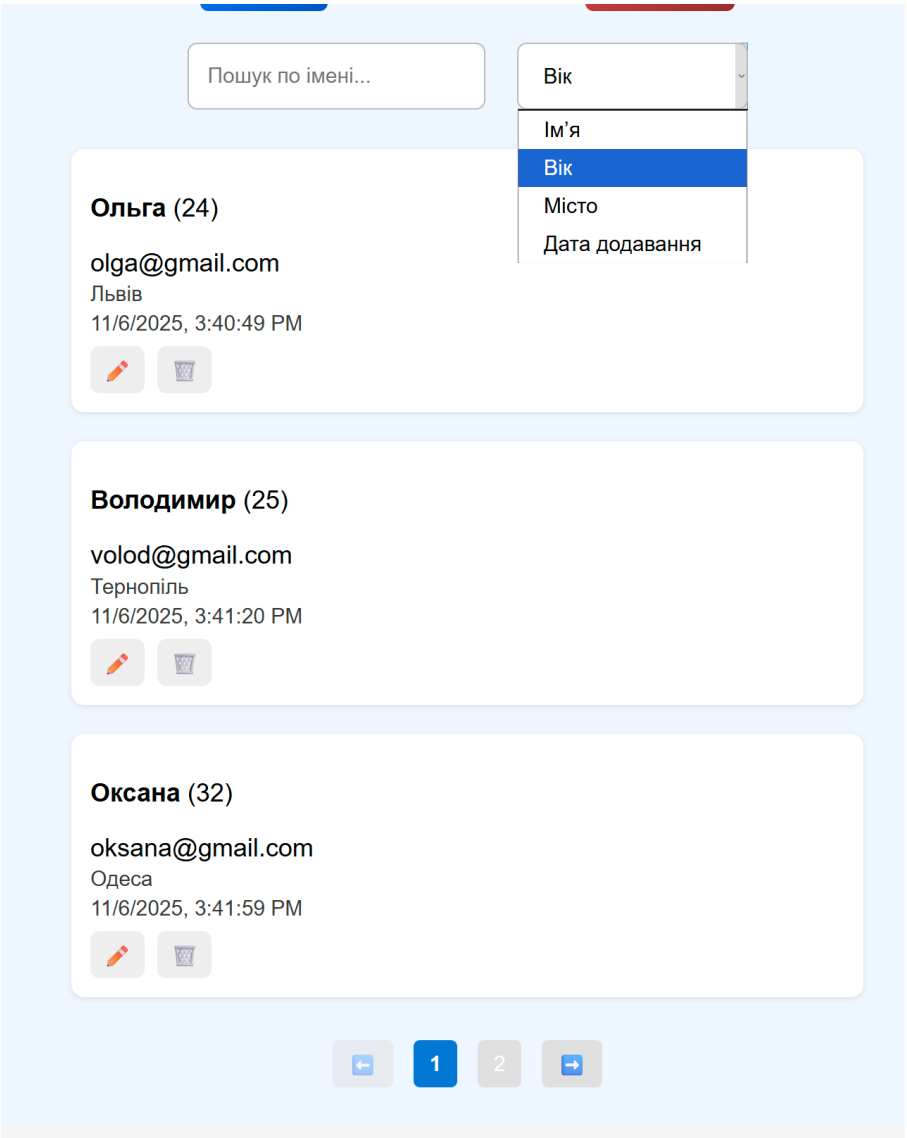


Рис. 3.24. Сортування записів

Після здійснення відповідних дій, про це буде сповіщено користувача, виведенням спеціальних повідомлень справа в кутку (рис. 3.25).

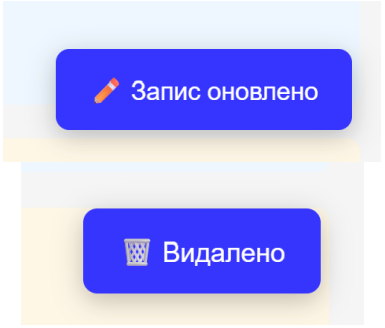


Рис. 3.25. Сповіщення щодо внесення змін

Далі доступний перегляд частини, що створена завдяки Vue.js. Найперше виводиться блок з таймером Pomodoro (рис. 3.26).

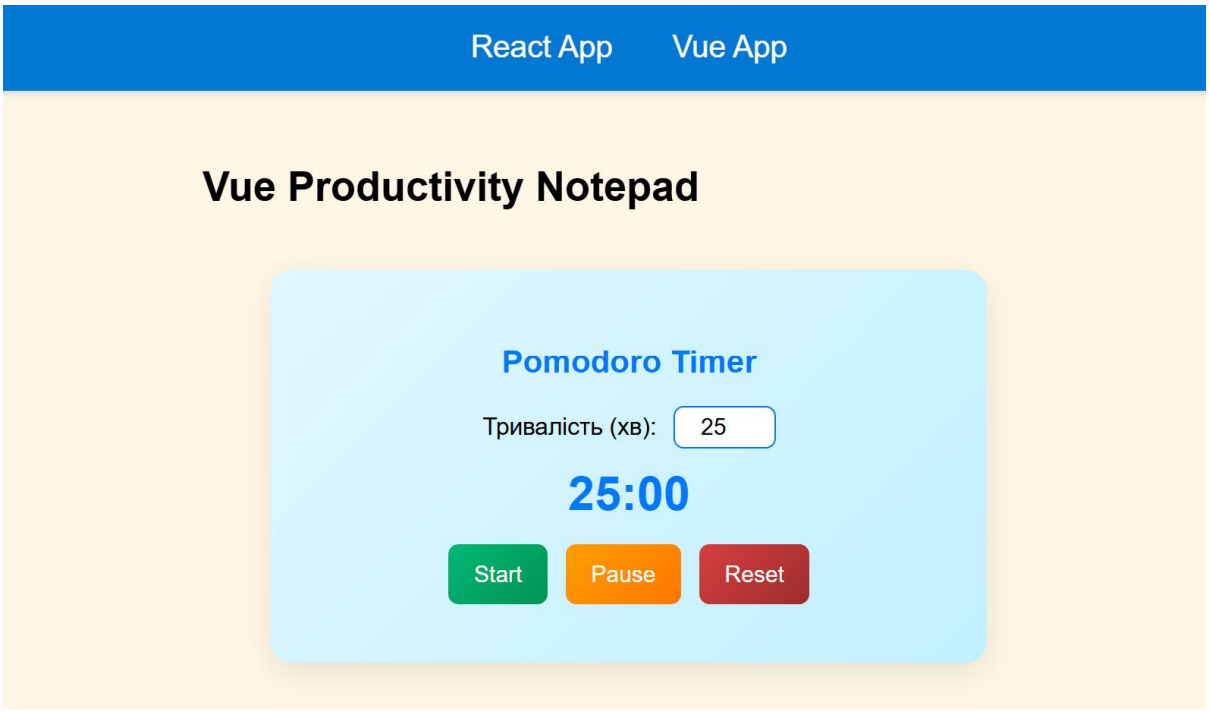


Рис. 3.26. Таймер помодоро

Далі йде внесення нових нотаток у додаток (рис. 3.27).

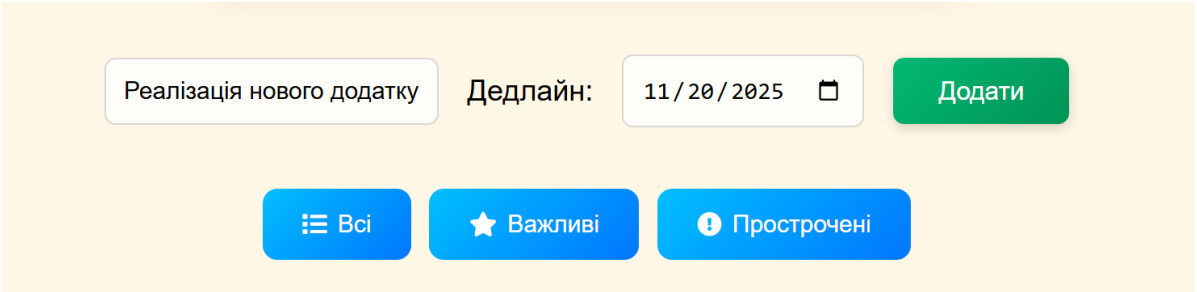


Рис. 3.27. Внесення нотатки

Переглянути всі нотатки, можна прогорнувши нижче форми внесення (рис. 3.28).

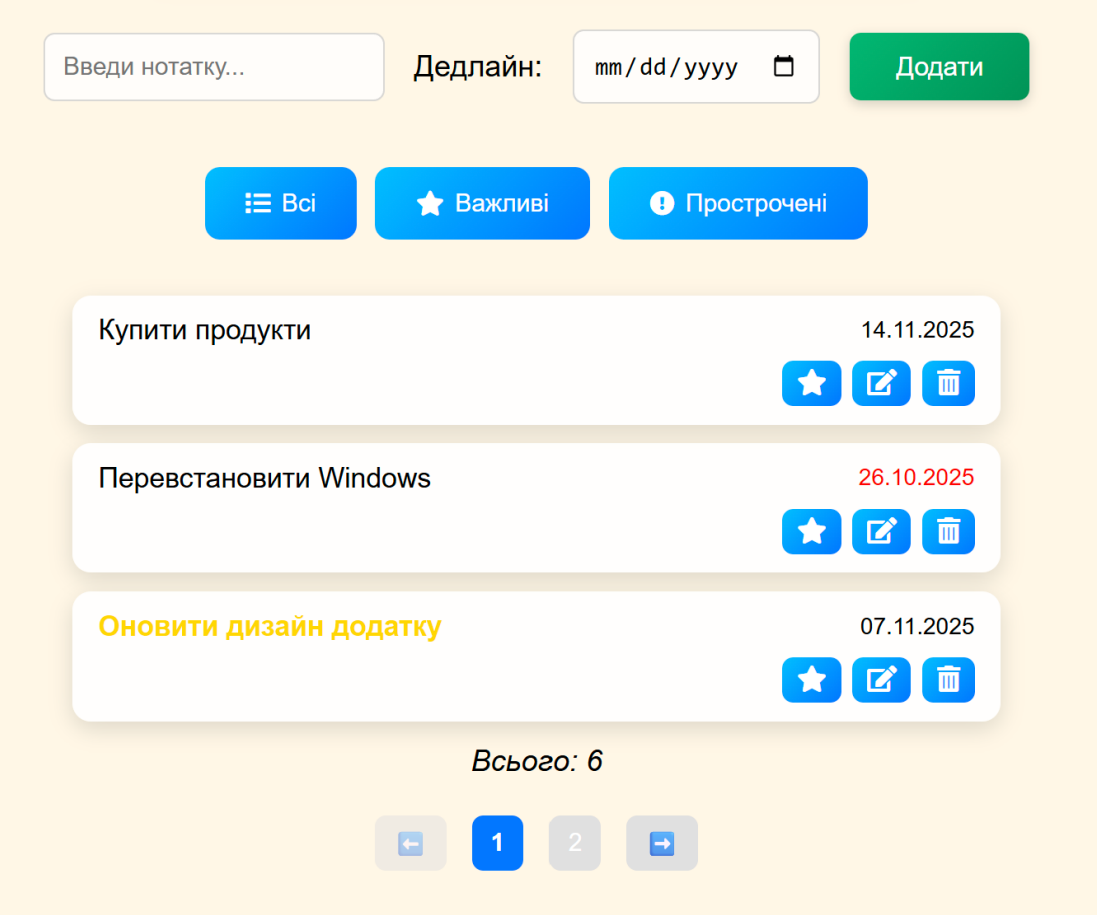


Рис. 3.28. Перегляд всіх нотаток

Натиснувши на олівець, можливо відредагувати внесені дані (рис. 3.29).

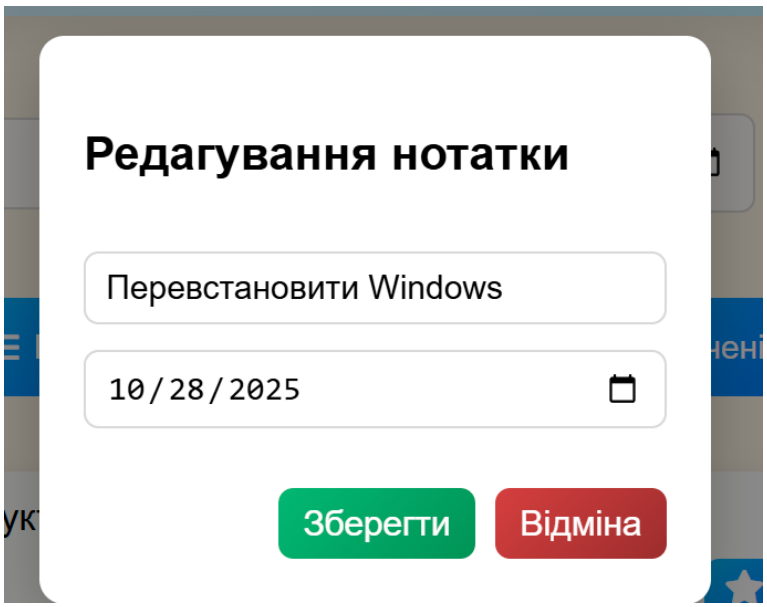


Рис. 3.29. Редагування нотатки

Для того, щоб відсортувати внесені нотатки, достатньо натиснути на відповідні кнопки. Тут доступні сортування за важливістю та чи були завдання прострочені (рис. 3.30).

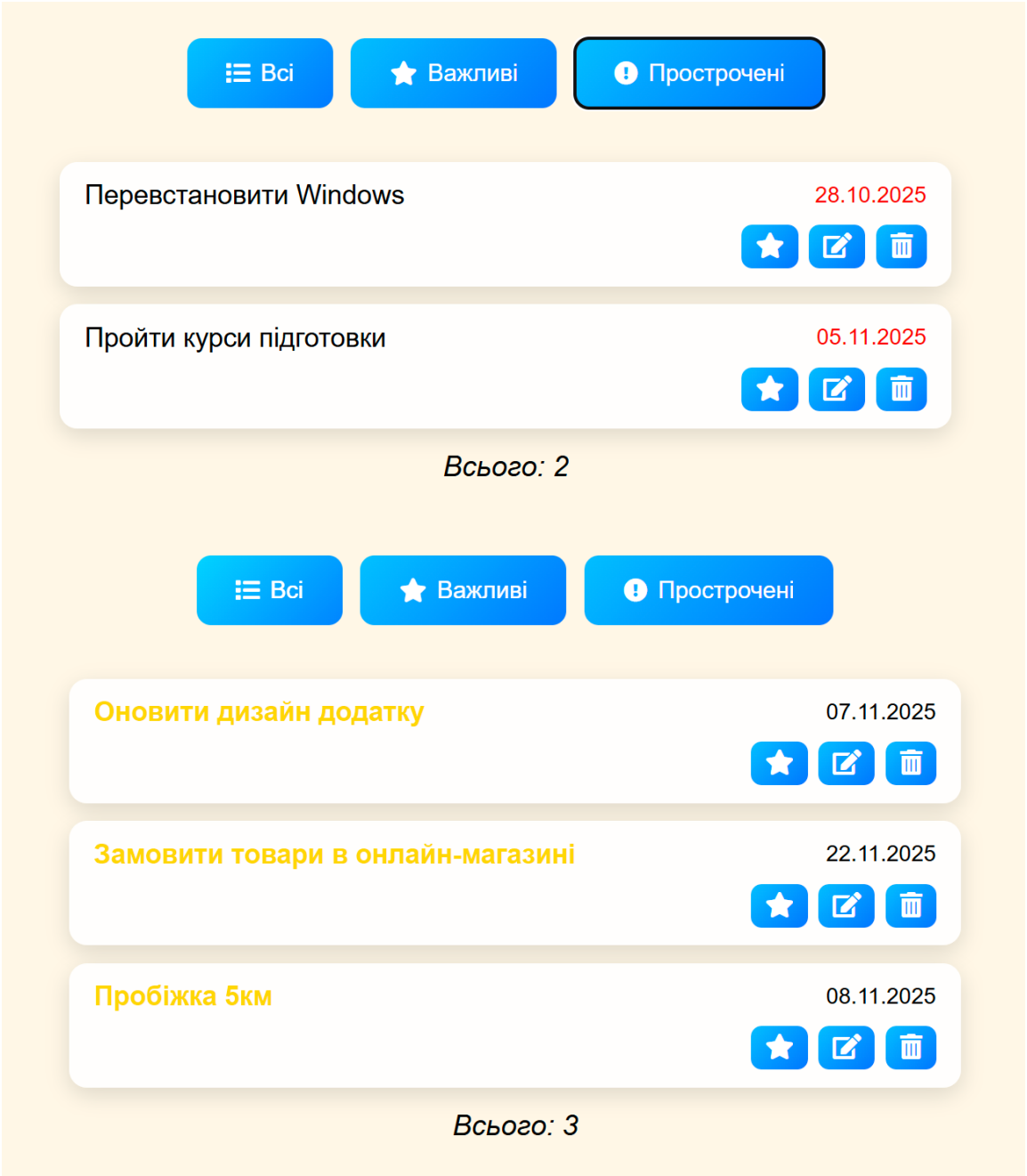


Рис. 3.30. Сортування нотаток

Після завершення кожної важливої дії, користувача буде сповіщено про неї, завдяки спеціальному сповіщенню (рис. 3.31).

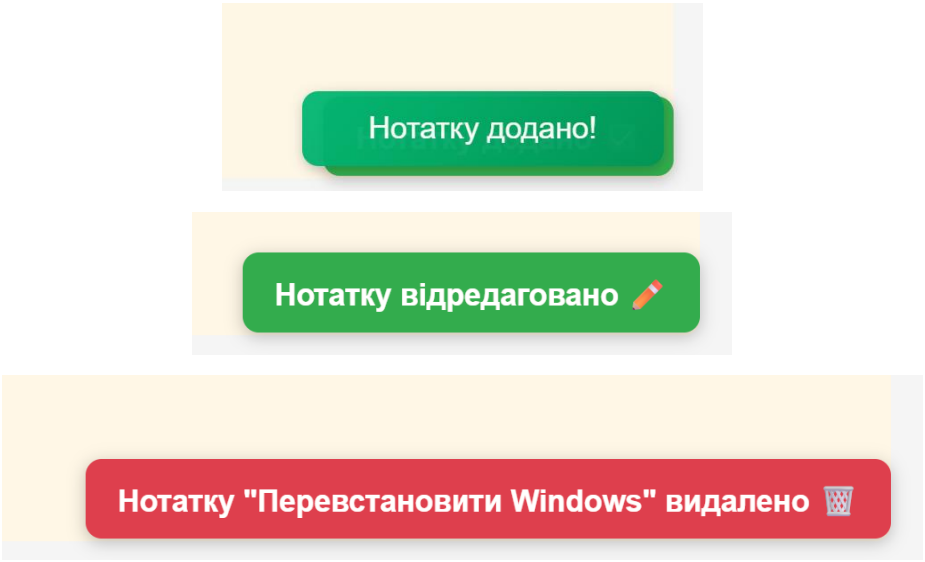


Рис. 3.31. Сповіщення щодо внесених змін

3.7 Оцінювання масштабованості та ефективності

3.7.1 Метрики продуктивності фронтенду

Для проведення аналізу потрібно перевірити впровадження проєкту. Для цього необхідно запустити їх в build. Щоб це виконати потрібно запустити в окремих мікрофронтендах.

Найперше в react потрібно додати спеціальні команди, що дозволять проводити запуски для білдів та аналізу створеного проєкту (рис. 3.32) [33].

```
"scripts": {
  "start": "webpack serve",
  "build": "webpack",
  "analyze": "source-map-explorer 'dist/react-app.js' 'dist/react-app.js.map' --no-border-checks",
  "build:analyze": "npm run build && npm run analyze"
},
```

Рис. 3.32. Команди для запуску та аналізу

Далі реалізований проєкт необхідно запустити на build, а не тестовий запуск (рис. 3.33) [34].

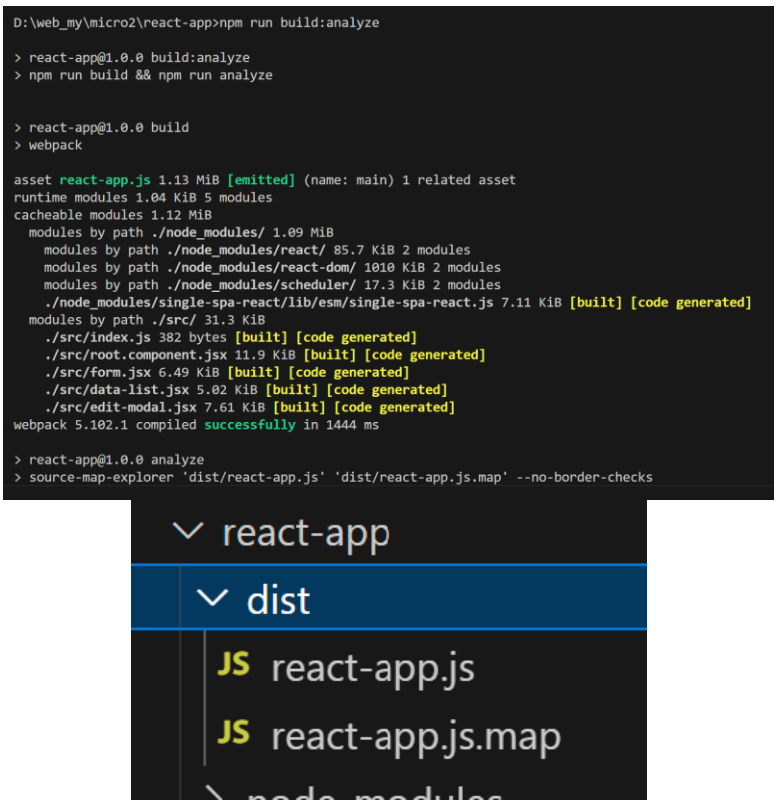


Рис. 3.33. Результат білду

Огляд реалізованих файлів, що надають опис – скільки розміру займають вони після білду (рис. 3.34).

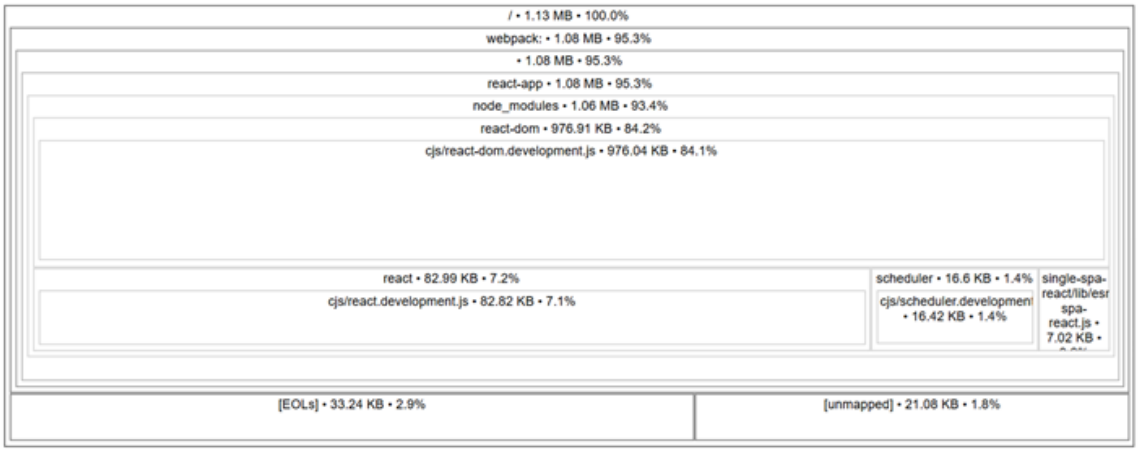
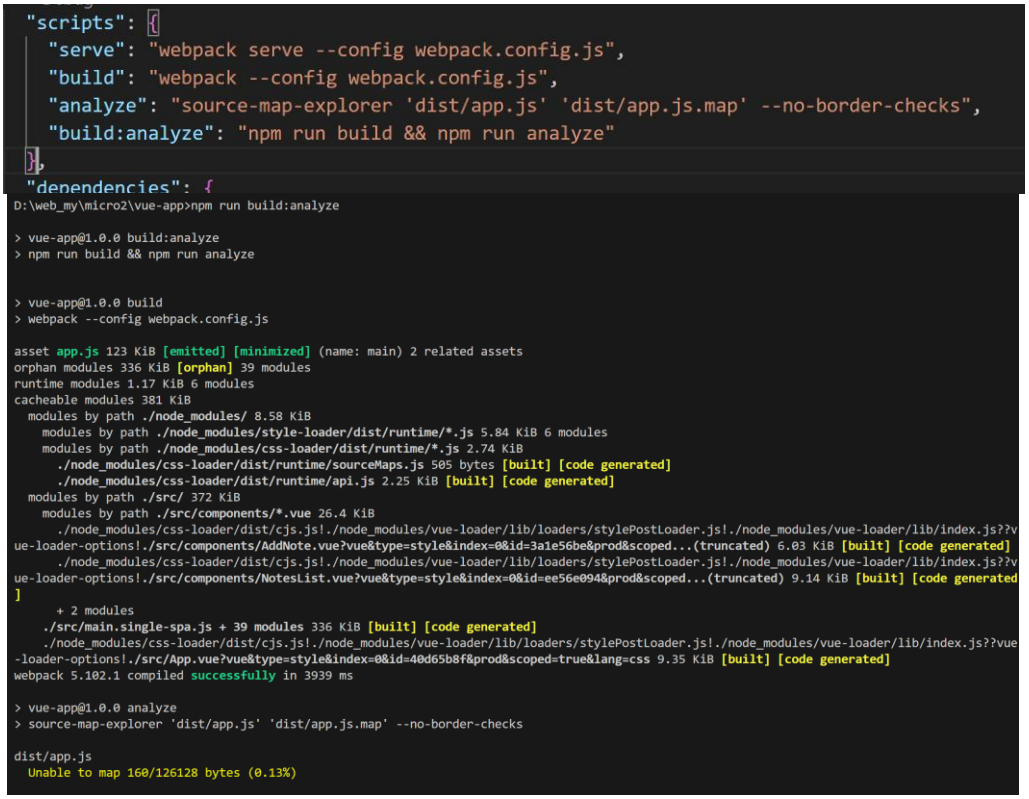


Рис. 3.34. Результат аналізу білду

Далі ці кроки необхідно провести для Vue (рис. 3.35) [35].



```
  "scripts": {
    "serve": "webpack serve --config webpack.config.js",
    "build": "webpack --config webpack.config.js",
    "analyze": "source-map-explorer 'dist/app.js' 'dist/app.js.map' --no-border-checks",
    "build:analyze": "npm run build && npm run analyze"
  },
  "dependencies": {
D:\web_my\micro2\vue-app>npm run build:analyze

> vue-app@1.0.0 build:analyze
> npm run build && npm run analyze


> vue-app@1.0.0 build
> webpack --config webpack.config.js

asset app.js 123 KiB [emitted] [minimized] (name: main) 2 related assets
orphan modules 336 KiB [orphan] 39 modules
runtime modules 1.17 KiB 6 modules
cacheable modules 381 KiB
  modules by path ./node_modules/ 8.58 KiB
    modules by path ./node_modules/style-loader/dist/runtime/*.js 5.84 KiB 6 modules
    modules by path ./node_modules/css-loader/dist/runtime/*.js 2.74 KiB
      ./node_modules/css-loader/dist/runtime/sourceMaps.js 505 bytes [built] [code generated]
      ./node_modules/css-loader/dist/runtime/api.js 2.25 KiB [built] [code generated]
  modules by path ./src/ 372 KiB
    modules by path ./src/components/*.vue 26.4 KiB
      ./node_modules/css-loader/dist/cjs.js!./node_modules/vue-loader/lib/loaders/stylePostLoader.js!./node_modules/vue-loader/lib/index.js??v
ue-loader-options!./src/components/AddNote.vue?vue&type=style&index=0&id=3a1e56be&prod&scoped...(truncated) 6.03 KiB [built] [code generated]
      ./node_modules/css-loader/dist/cjs.js!./node_modules/vue-loader/lib/loaders/stylePostLoader.js!./node_modules/vue-loader/lib/index.js??v
ue-loader-options!./src/components/NotesList.vue?vue&type=style&index=0&id=ee56e094&prod&scoped...(truncated) 9.14 KiB [built] [code generated
]
      + 2 modules
    ./src/main.single-spa.js + 39 modules 336 KiB [built] [code generated]
    ./node_modules/css-loader/dist/cjs.js!./node_modules/vue-loader/lib/loaders/stylePostLoader.js!./node_modules/vue-loader/lib/index.js??vue
-loader-options!./src/App.vue?vue&type=style&index=0&id=40d65b8f&prod&scoped=true&lang=css 9.35 KiB [built] [code generated]
webpack 5.102.1 compiled successfully in 3939 ms

> vue-app@1.0.0 analyze
> source-map-explorer 'dist/app.js' 'dist/app.js.map' --no-border-checks

dist/app.js
  Unable to map 160/126128 bytes (0.13%)
```

Рис. 3.35. Внесення та запуск команд для білду

Огляд реалізованих файлів, що надають опис – скільки розміру займають вони після білду для проєкту Vue (рис. 3.36).

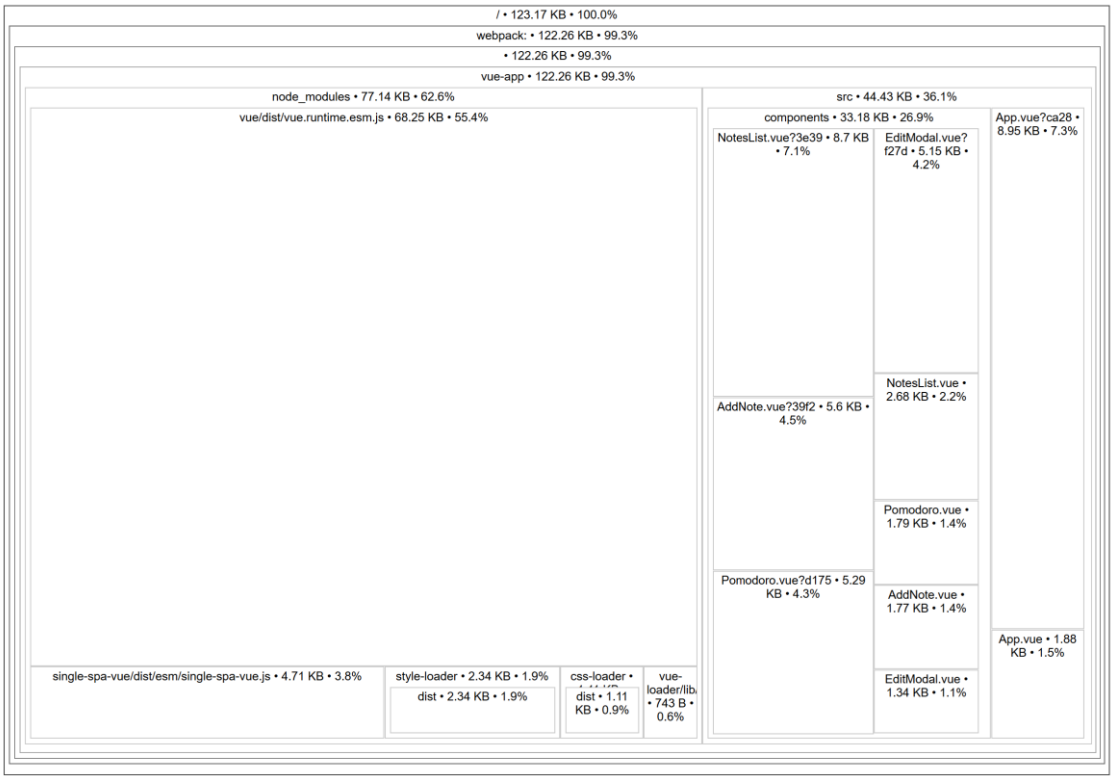


Рис. 3.36. Результат аналізу білду

Для перевірки ефективності необхідно звернутись до інструментів браузера (рис. 3.37) [36].

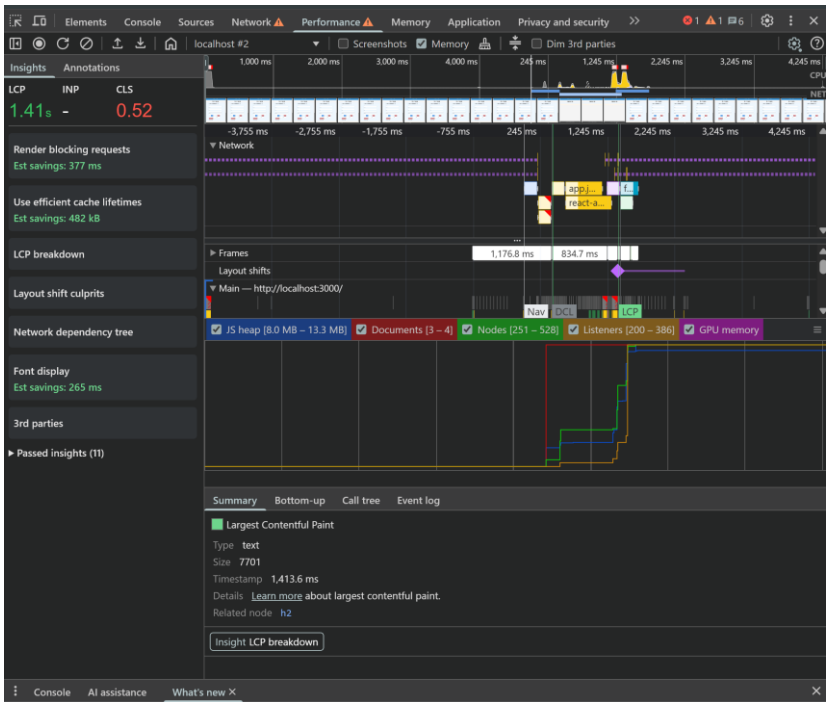


Рис. 3.37. Вкладка Performance

Наступним кроком буде перевірка на споживання пам’яті. Спочатку необхідно глянути на результати до використання сторінки (рис. 3.38) [37].

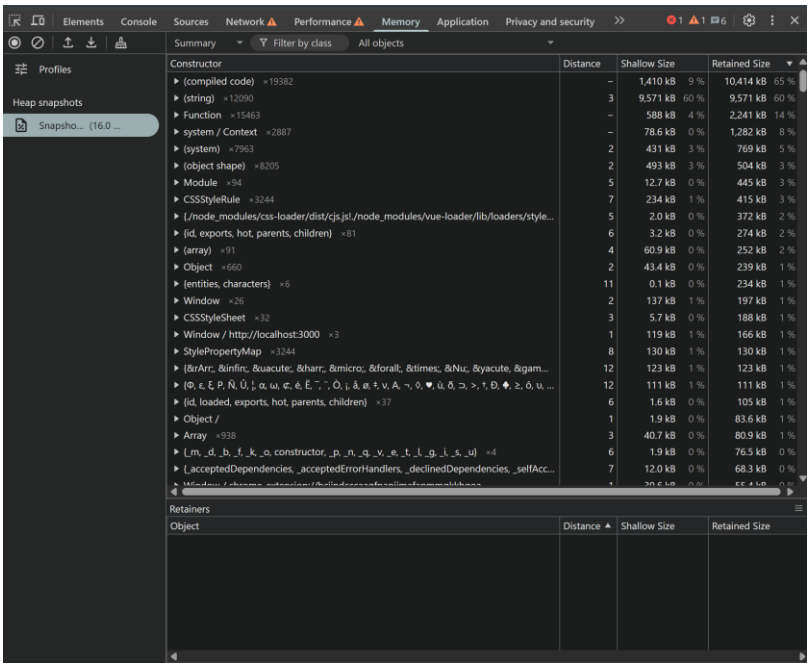


Рис. 3.38. Сторінка Memory до використання сторінки

Та далі споживання після її використання (рис. 3.39).

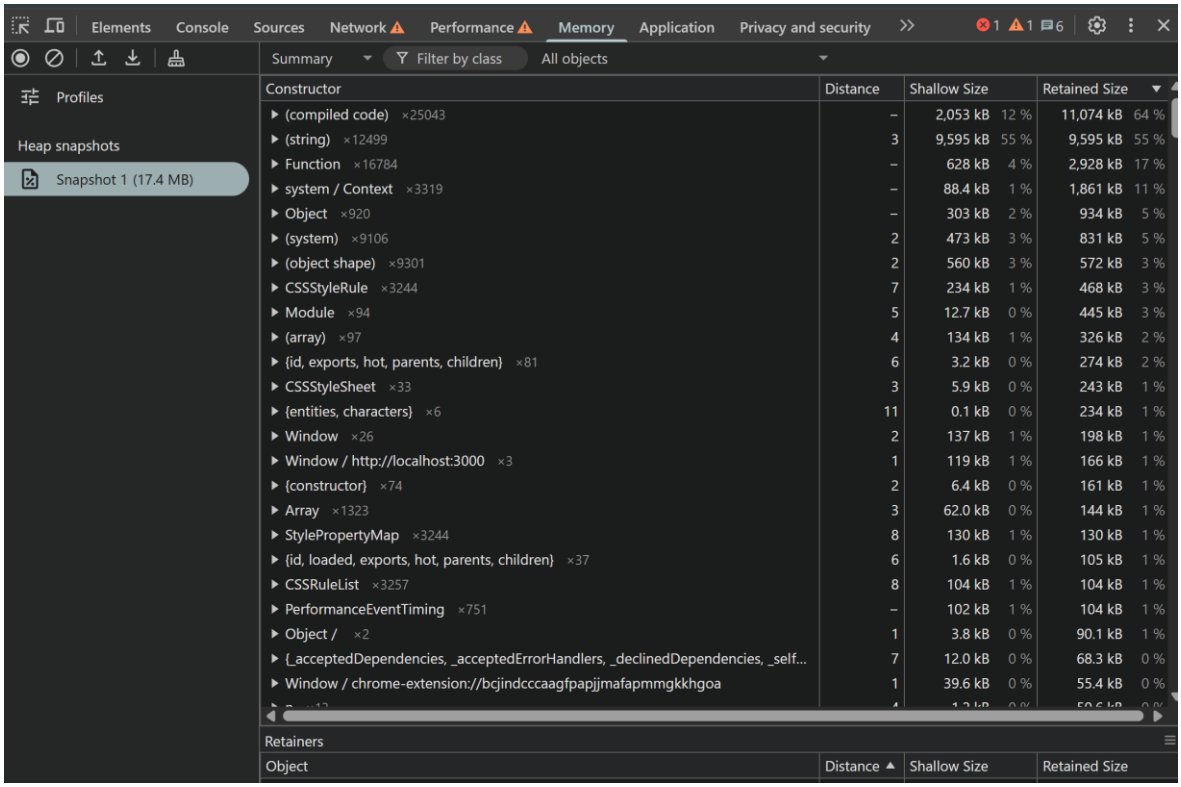


Рис. 3.39. Сторінка Memory до використання сторінки

У підсумку можна побачити, що memory leak не відбувається. Це означає, що все було виконано правильно та працює коректно. Більш детально підсумки підбиті на таблиці нижче (табл. 3.3).

Таблиця 3.3

Таблиця для опису отриманих результатів

Метрика	Значення	Коментар
First Contentful Paint	0.4 с	Відповідає стандарту
LSP	1.41 с	Відповідає стандарту
CLS	0.52	Варто мінімізувати
Render blocking requests	0.377 с	Відповідає стандарту
Use efficient cache lifetimes	482 Кб	Відповідає стандарту

Таблиця 3.3 (продовження)

Таблиця для опису отриманих результатів

Метрика	Значення	Коментар
Total Blocking Time	260 ms	Варто мінімізувати
Speed Index	0.6 с	Відповідає стандарту
Largest Contentful Paint	1.3 с	Нормальний рівень
Пам’ять (початок)	16 MB	-
Пам’ять (через 30 сек)	17,5 MB	memory leak немає

3.7.2 Аналіз продуктивності моноліту versus мікрофронтендів

Для проведення аналізу продуктивності мікрофронтендів та монолітного підходу, було необхідно реалізувати створений проєкт за допомогою одного фреймворку, що складається з двох частин (рис. 3.40).

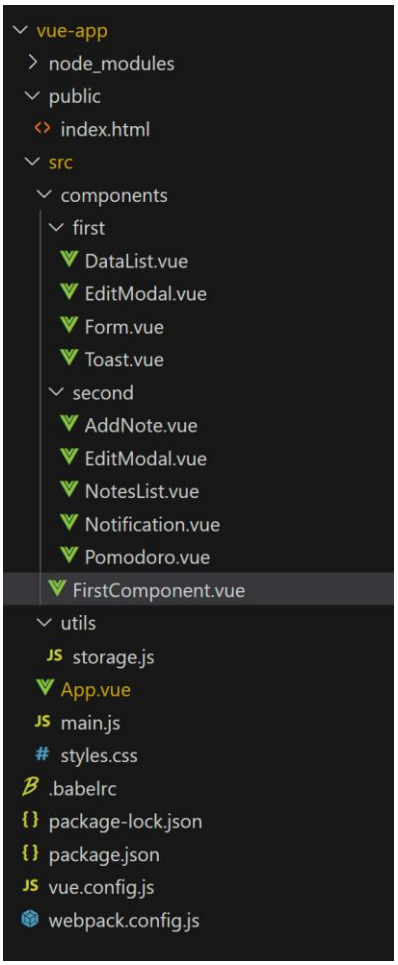


Рис. 3.40. Структура нового проєкту, що створений повність на Vue.js

Далі проєкти було запущено та проведено порівняння між ними. Спочатку наводяться результати перевірки для моноліту, для цього необхідно скористатись функцією Lighthouse (рис. 3.41-3.42) [38].

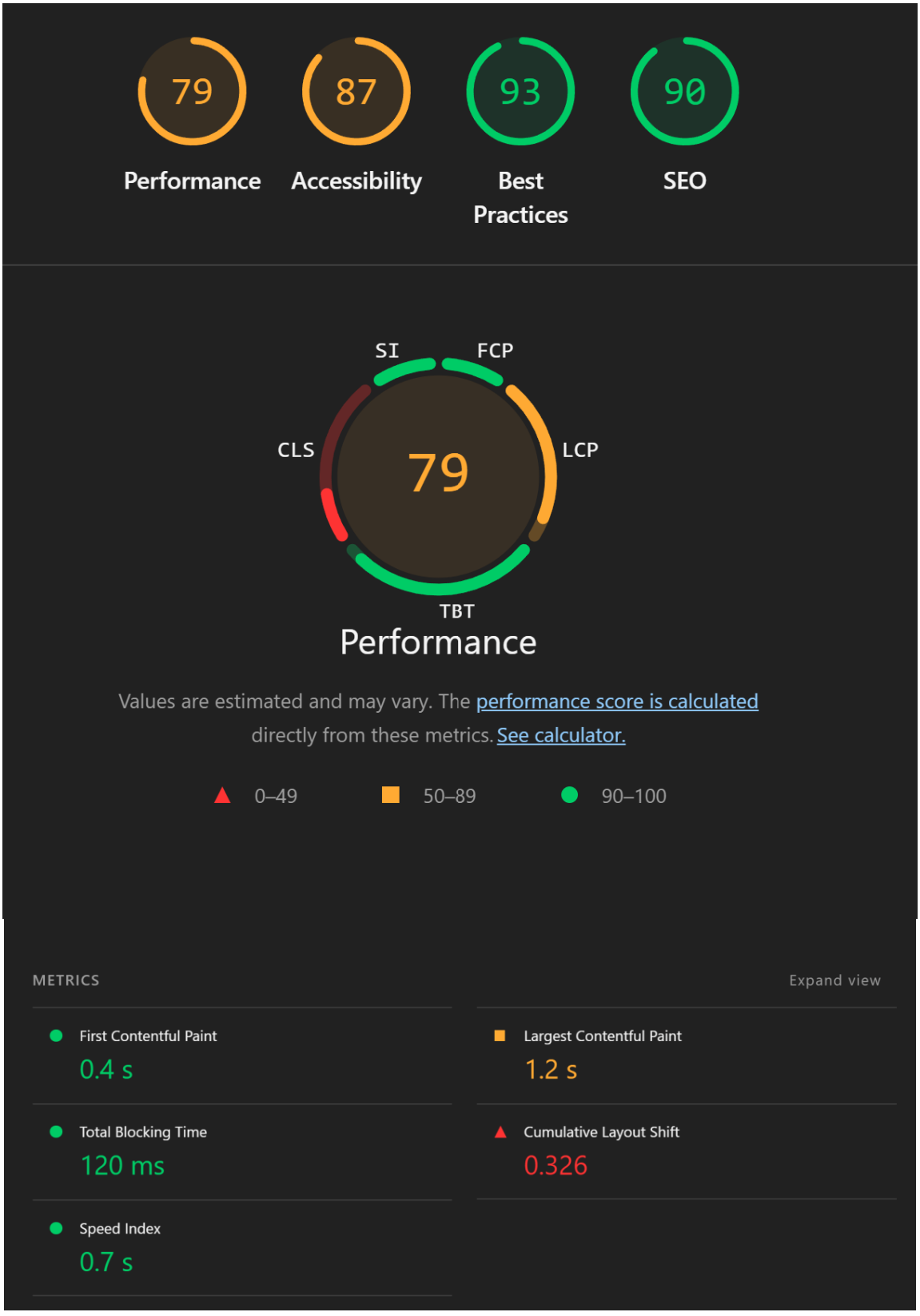


Рис. 3.41. Розділ Navigation для моноліту

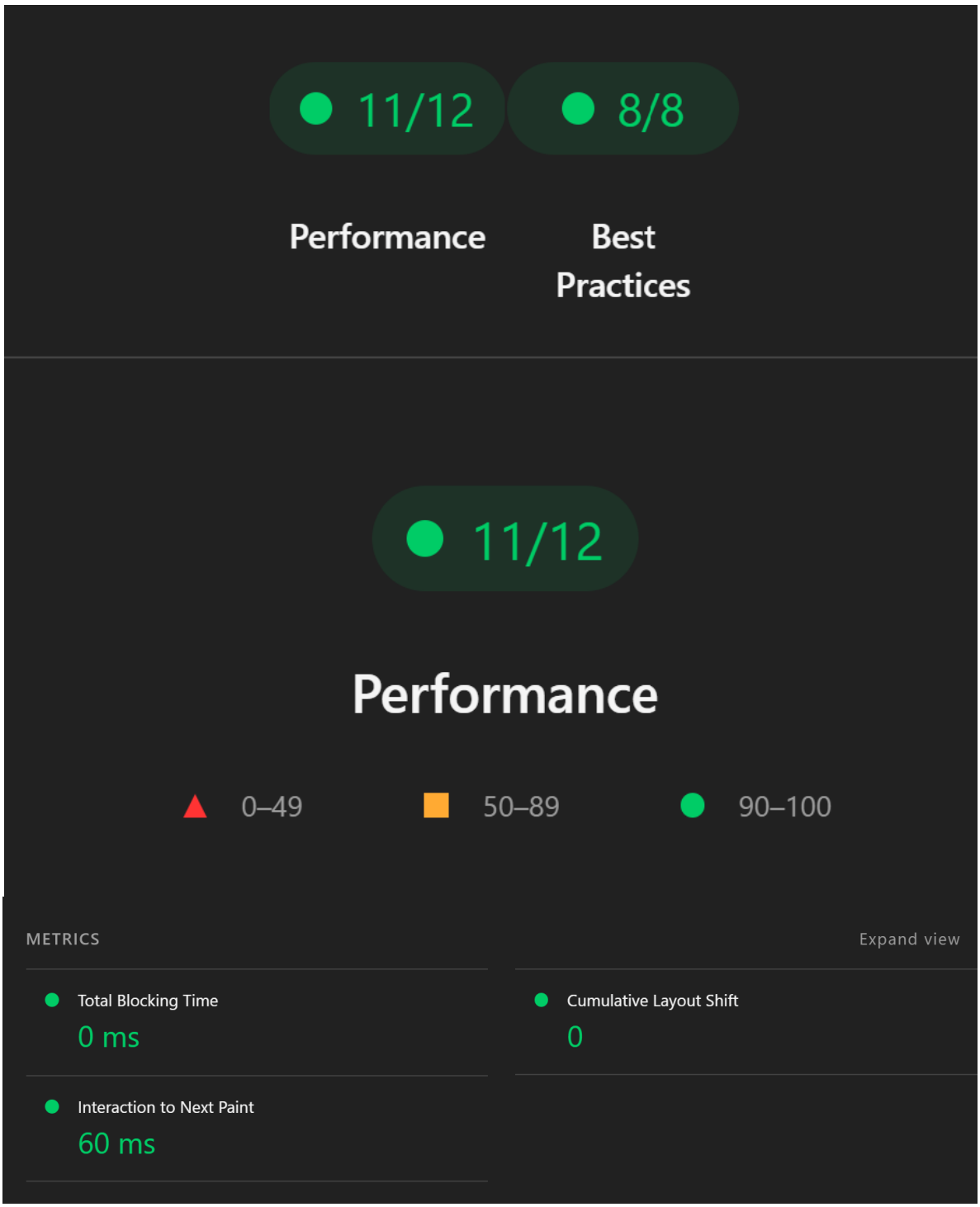


Рис. 3.42. Розділ Timespan для моноліту

Далі наводяться результати перевірки для мікрофронтендів, для цього необхідно далі скористатись функцією Lighthouse (рис. 3.43-3.44).

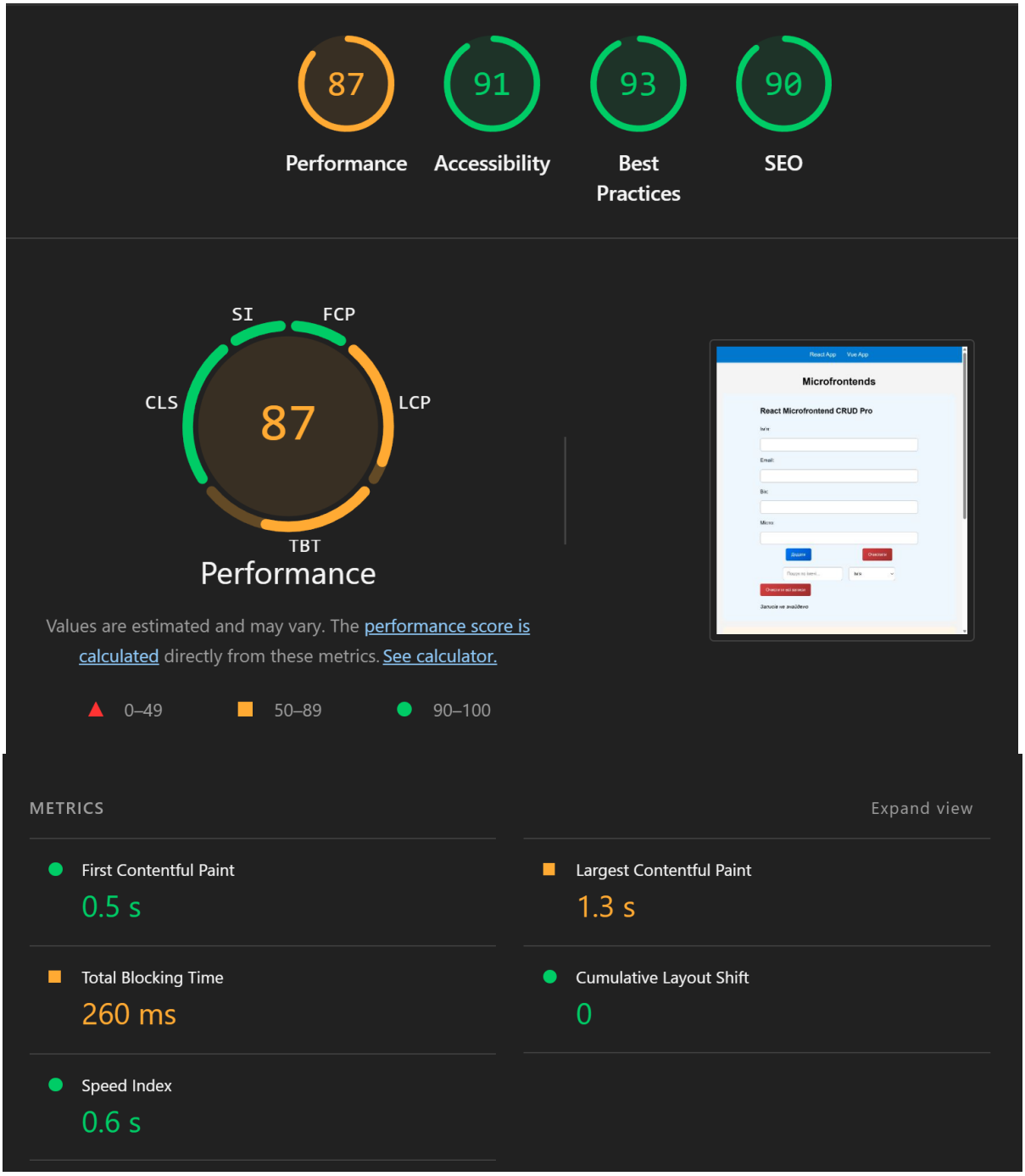


Рис. 3.43. Розділ Navigation для мікрофронтенду

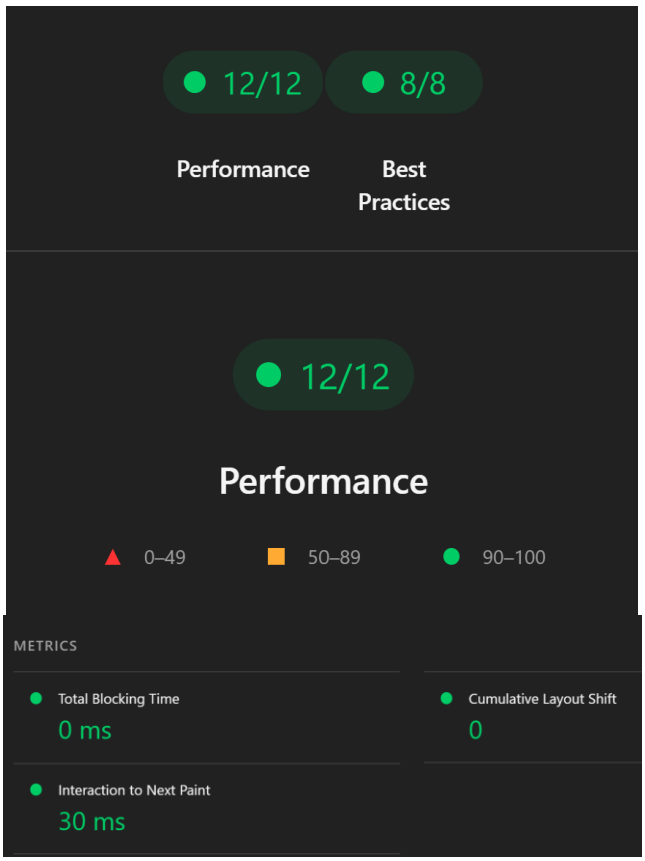


Рис. 3.44. Розділ Timespan для мікрофронтенду

Ці дані будуть подані для кращого порівняння у вигляді таблиці (табл. 3.4).

Таблиця 3.4

Порівняльна характеристика результатів перевірки монолітної та мікросервісної архітектур

Показник	Моноліт	Мікрофронтенд
Загальна оцінка (Lighthouse)		
Performance	79	87
Accessibility	87	91
Best Practices	93	93
SEO	90	90
Core Web Vitals		
First Contentful Paint (FCP)	0.4 с	0.5 с

Таблиця 3.4 (продовження)

Порівняльна характеристика результатів перевірки монолітної та мікросервісної архітектур

Largest Contentful Paint (LCP)	1.2 с	1.3 с
Total Blocking Time (TBT)	120 мс	260 мс
Cumulative Layout Shift (CLS)	0.326	0
Speed Index (SI)	0.7 с	0.6 с
Interaction to Next Paint (INP)	60 мс	30 мс
Best Practices (деталі)		
Performance (checks)	11	12
Best Practices (checks)	8	8

Провівши порівняльний аналіз між наведеними архітектурами, можна побачити чіткі відмінності, які є між ними. Це відображається у швидкодії, реактивності інтерфейсу та стабільності відображення застосунків.

Монолітний варіант відображає його, як єдиний додаток, тут всі модулі та реалізовані компоненти зв’язані між собою. Це загалом спрощує розробку проєкту на початковому етапі, але створює обмеження при масштабуванні та напряму впливає на завантаження сторінки. Розділивши ж її на різні елементи завдяки мікрофронтендам, дозволить покращити продуктивність застосунку.

Мікрофронтенд-оптимізація дозволяє уникнути блокування основного потоку при рендерингу сторінки. Ще одним з важливих показників є стабільність макету сторінки при завантаженні (CLS), монолітна версія показує значення 0.326, що вказує помітний зсув

елементів. Коливання такого типу залишають негативний досвід для користувачів, при використанні веб-сайту. Мікрофронтенд ж повністю усунув цю проблему.

Швидкість реакції інтерфейсу на дії з точки зору користувача (INP) значно покращився в мікрофонтенді, показавши 30мс, напроти 60мс в моноліті. Ці зміни свідчать про ефективне використання рендерингу елементів за допомогою використання оптимізованих React-хуків.

В загальному оцінка Performance зросла від 79 до 87. Це чітко підтверджує, що додаток готовий до запуску. Маючи швидке завантаження, стабільний макет та хорошу інтерактивність.

Тобто, мікрофронтендний підхід надає перевагу не тільки у масштабуванні, а й в зручності використання. Ізольовані модулі надають незалежні розгортання окремих компонентів, через що система стає надійнішою, продуктивнішою та кращою для розширення.

3.7.3 Метрики продуктивності фронтенду

При детальному розгляді можна точніше сказати, що мікрофронтендна архітектура показує себе краще при масштабуванні проєктів, є готовою для впровадженню завдяки наступним пунктам:

- повній стабільності макета (CLS = 0)
- покращеній інтерактивності (INP = 30 мс)
- відсутності помилок у Best Practices (усі 12/12 перевірок пройдено)

Водночас можна побачити незначне погіршення затримок при виконанні JavaScript (TBT = 260 мс), що оптимізується завдяки додатковим технологіям: розділення бандлів(вхідних скомпільований файлів JS), застосування розмежування окремих маршрутів та асинхронним імпортом модулів.

Надалі рекомендується внести кешування статичних ресурсів, lazy loading у великих компонентів та CDN для швидшого завантаження. Після

чого потрібно впровадити автоматизоване розгортання мікрофронтендів за допомогою CI/CD пайплайн, що надасть стабільність та своєчасне оновлення даних сторінки.

Для ізоляції кожного з частин мікрофронтенду рекомендується використати Docker, що в свою чергу спростить розгортання.

Загалом отримані результати засвідчують, що обраний підхід надає кращу стабільність, навіть через те, що необхідно затратити більше часу та ресурсів на реалізацію початкового застосунку.

Висновки до розділу 3

Під час написання третього розділу було проведено проєктування та реалізацію застосунку, що заснований на архітектурі з використанням мікрофронтенду. Першим етапом було виділення вимог до системи та здійснено опис бізнес процесів.

Після цього наведено моделювання процесів завдяки використанню нотацій IDEF0 та BPMN, які візуалізують роботу системи, структуруючи описані бізнес-операції. Це спрощує уявлення послідовності дій та покращує бачення взаємодії між системою та кінцевими користувачами.

Етап проєктування включав у себе визначення конкретних компонентів, модулів та їх зв’язки між ними. Центральним елементом системи є Single-SPA, який відповідає за динамічне завантаження, що необхідний для повноцінної роботи системи. Далі була описана реалізація окремих модулів додатку, які працюють на React та Vue.

Було протестовано застосунок через перевірки роботи модулів. Далі було запущено всі сервери та перевірено кожну функцію. Результати показали, що реалізована система відповідає всім вимогам та готова до ймовірного масштабування.

Заключний етап включав детальний аналіз продуктивності розробленого продукту. Була проведена перевірка запуску білду, який засновувався на архітектурах React та Vue, після чого були оцінені розміри скомпільованих файлів та використання пам’яті ними, що підтвердило відсутність витоків.

Після цього було наведено порівняння продуктивності монолітної та мікрофронтендної архітектур. Загалом результати порівняння показали, що мікрофронтендна архітектура забезпечила кращу швидкодію, інтерактивність, стабільність та хороші показники Core Web Vitals. Монолітна версія зможе спростити розробку тільки на перших етапах розробки, надалі при збільшенні погіршується рендеринг та нестабільність

макету.

Аналіз засвідчив, що обрана архітектура ізолюючи модулі, дозволяє самостійне розгортання кожного з елементів сайту. Водночас можна побачити збільшення Total Blocking Time, що можна уникнути через розділення бандлів та використання кешування.

Отримані результати підтверджують, що застосунок демонструє стабільність, відповідну швидкодію та відкритою до подальшого масштабування.

РОЗДІЛ 4
РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

4.1 Загальна характеристика стартап-проєкту

Під час написання цієї магістерської дисертації було розроблено демонстраційний веб-застосунок, який наочно показав практичне застосування мікрофронтендів у розробці сучасних, масштабованих та незалежних інтерфейсів. Розроблений застосунок демонструє перспективність та актуальність використання мікрофронтендів у напрямі веб-проєктування. Варто зазначити, що схожі реалізовані рішення майже не зустрічаються у відкритому доступі, тому продемонстроване рішення реалізації мікрофронтендів за допомогою Vue.js та React, платформи Single-SPA та збирача Webpack є актуальним. Також даний проєкт може слугувати дослідницькою та освітньою складовою. За допомогою табл. 4.1 виконано загальну характеристику стартап-проєкту.

Таблиця 4.1

Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вимоги до користувача
Зміст ідеї полягає в проєктуванні, розробці та розгортанні демонстраційного односторінкового веб-застосунку, який реалізовано з використання мікрофронтендної архітектури.	1.Використання у дослідницьких цілях. Розробники зможуть використати систему для експериментальних тестувань різних сценаріїв.	1. Користувач (розробник) має мати глибокі знання галузі веб-розробки.

Таблиця 4.1 (продовження)

Опис ідеї стартап-проєкту

Система створена за допомогою платформи Single-SPA та поєднує в собі використання Vue.js і React. Розгорнений застосунок демонструє переваги співіснування кількох незалежних частинок фронтенду на одній веб-сторінці.	2. Використання у процесі навчання. Веб-застосунок можна використовувати у навчальних цілях для демонстрації переваг та недоліків архітектури мікрофронтендів	2. Студент повинен мати базові знання у веб-розробці, а також розуміти принципи роботи використаних фреймворків.
	3. Використання на курсах підготовки веб-розробників.	3. Початківці у галузі веб-розробки мають мати базові знання технологій HTML, CSS та JavaScript, та розуміти принцип роботи з фреймворками Vue і React.

На цьому етапі буде проведений порівняльний аналіз показників існуючих архітектурних рішень для реалізації клієнтських частин, який має визначити гірші, аналогічні та кращі значення власної ідеї проєкту. Аналіз потенційних техніко-економічних переваг ідеї буде продемонстровано за допомогою табл. 4.2.

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

Техніко-економічні характеристики ідеї	Системи конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
	Мій проєкт	Проєкт на моноліті	Проєкт на SPA			
Простота налаштування та розгортання	+	±	±			+
Швидкість процесу розробки	±	+	+		+	
Незалежне оновлення веб-модулів	+	-	±			+
Хороша продуктивність навіть з великою кількість компонентів	±	+	±		+	
Висока стійкість до відмов	+	–	±			+
Можливість поділу розробки на команди	+	−	±			+
Легше масштабування	+	−	−			+

Таблиця 4.2 (продовження)

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

Зручніший процес тестування	+	−	±			+

4.2 Технологічний аудит ідеї проєкту

У цьому підрозділі буде проведений технологічний аудит, який забезпечить визначення середовищ, інструментів та технологій, потрібних для реалізації архітектури мікрофронтендів. Розглянуті технологічні стеки, які можна використати для реалізації архітектурних принципів мікрофронтендів, представлені за допомогою табл. 4.3.

Таблиця 4.3

Технологічна здійсненність ідеї проєкту

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Демонстраційний односторінковий веб-застосунок спроєктований з використанням архітектурного підходу мікрофронтендів	Vue, Angular, Web Components (Custom Elements)	Наявні	Доступна (потребує ручних налаштувань)
2		Svelte, React, Parcel, Webpack	Наявні	Доступна (гірша екосистема)
3		Vue, React, Single-SPA, Webpack	Наявні	Доступна (відкрите ПЗ та хороша документація)
Обрані технології реалізації ідеї проєкту: Vue, React, Single-SPA, Webpack				

Проведений технологічний аудит показав (видно з табл. 4.3), що реалізувати демонстраційний односторінковий веб-застосунок з використанням архітектурного підходу мікрофронтендів можна за допомогою значної кількості сучасних технологій, отже реалізація проєкту є можливою. Зважаючи на простоту реалізації, доступність технологій та реалізацію проєкту одноосібно – вибір було здійснено на користь Vue, React, Single-SPA та Webpack.

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Далі буде визначено ринкові можливості, які використовують під час впровадження, а також ряд загроз, які перешкоджають впровадженню проєкту. Всі нюанси продемонстровано за допомогою табл. 4.4-4.13.

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проєкту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	2-4 (якщо не брати до уваги кількість локальних платформ в межах навчальних закладів)
2	Загальний обсяг продаж, грн/ум.од	Загальний обсяг продаж може бути високим, зважаючи на популярність архітектури мікрофронтендів у навчальних кейсах
3	Динаміка ринку (якісна оцінка)	Зростає (оскільки інтерес до сучасного типу архітектури збільшується)

Таблиця 4.4 (продовження)

Попередня характеристика потенційного ринку стартап-проєкту

4	Наявність обмежень для входу (вказати характер обмежень)	Присутня конкуренція, потрібна висока технічна компетенція
5	Специфічні вимоги до стандартизації та сертифікації	Високі вимоги до дотримання стандартів безпеки
6	Середня норма рентабельності в галузі (або по ринку), %	10–15% (для комерційного використання)

Табл. 4.4 демонструє, що ринок такого роду веб-застосунків динамічно зростає та є перспективним. Оскільки основний попит на систему буде у викладачів, розробників та новачків у сфері розробки, то далі буде зазначено характеристику потенційних груп клієнтів за допомогою табл. 4.5.

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Використання в сучасних архітектурних рішень для швидкого	Освітні заклади. Студенти та викладачі IT-спеціальностей. Початківці у сфері розробки.	Кожен освітній заклад має мати демонстраційну систему, яку зможе показувати студентам і на фоні якої виконуватиме	- простота для розуміння (для адаптації під навчальних процес в освітніх

	проєктування та розгортання веб-застосунків	Розробники з досвідом	навчальний процес. Студенти зможуть швидко опанувати тему мікрофронтендів, а викладачі – спростити навчальний процес. Початків у сфері веб-розробки зможуть побачити приклади практичного вирішення своєї проблеми. Досвідчені розробники – оцінити якісь реалізації та масштабувати для своїх потреб.	закладах); - можливість швидко розгорнути таку систему; - стабільна робота на різних пристроях; - можливість масштабування існуючої системи під свої потреби.

За допомогою табл. 4.6-4.7 буде проведений аналіз ринкового середовища. Даний аналіз буде представлено в таблицях фактори загроз та фактори можливостей.

Таблиця 4.6

Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Незначна кількість користувачів.	Демонстраційна система може обмежуватись тільки для навчальних цілей. Тема мікрофронтендів може бути не включена в навчальний план	Розширення можливостей веб-застосунку.

Таблиця 4.6 (продовження)

Фактори загроз

2	Ризики застарілості системи.	Оскільки технології швидко змінюються, то використане дослідження архітектури може втратити актуальність з часом.	Слідкувати за сучасними технологіями та адаптувати демонстраційну систему до нових стандартів.
3	Складність впровадження мікрофронтендів	Проєктування та розгортання застосунків з архітектурним підходом мікрофронтендів вимагає певного рівня досвіду та навичок.	Використати спрощення системи, а також покроковий описів розгортання.

Таблиця 4.7

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростання інтересу до архітектури мікрофронтендів в освітніх програмах	Нові підходи до проєктування клієнтської частини набувають популярності через значну потребу в масштабуванні.	Співпраця з навчальними закладати для підтримки такого роду навчальних програм.

Таблиця 4.7 (продовження)

Фактори можливостей

2	Збільшений попит на універсальних розробників фронтенду.	Зростання функціональний можливостей в застосунках викликає збільшений попит на кваліфікованих та універсальний веб-розробників.	Співпраця з освітніми курсами, які виконують перепідготовку та підвищення кваліфікації веб-розробників.
3	Розширення фунціоналу	Закріплення затребуваності веб-системи за рахунок збільшення функціональних можливостей.	Розробка додаткових аналітичних панелей, які дозволять підвищити інтерактивність системи.

За допомогою табл. 4.8 буде представлено ступеневий аналіз конкуренції на ринку.

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Чиста конкуренція	Якщо подібні рішення існують, то вони на внутрішньому корпоративному рівні, а також в межах локальних навчальних закладів.	Компанія може забезпечити відкритий доступ до веб-системи та запровадити якісну документацію.

Таблиця 4.8 (продовження)

Ступеневий аналіз конкуренції на ринку

Міжнародний рівень конкуренції	Має потенціал для демонстрації сучасний архітектур в навчальних закладах країни, а також за кордоном.	Компанія може проводити вебінари
Міжгалузева конкуренція	Демонстраційна система складає конкуренцію не лише веб-застосунках на моноліті, а також подібним мікрофронтендним рішенням.	Компанія повинна демонструвати всі переваги цього архітектурного підходу назагал.
Товарно-видова конкуренція	Є веб-системи, які демонструють використання мікрофронтендів з іншим набором технологій.	Компанії потрібно показати переваги використання свого стеку технологій за допомогою демонстрації унікальних сценаріїв застосування.
Нецінова конкуренція	Зручність демонстраційної системи, а також продуктивність та простота використання.	Компанія мусить зробити веб-застосунок зрозумілим, простим та забезпечити оптимізацію продуктивності.
Немарочна конкуренція	Демонстраційна система поки що немає впізнаваності.	Компанія може просувати веб-застосунок в спільнотах розробників та конференціях для навчальних закладів.

Далі можна навести більше риси конкуренції на ринку за М. Портером (табл. 4.9).

Таблиця 4.9

Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари замінники
Складові аналізу	Аналогічні рішення, але виконані за допомогою іншої архітектури та технологій.	Інші розробники, які можуть створити подібні дослідницькі проєкти.	Розробники фреймворків та інструментів.	Студенти та викладачі IT-напрямів, початківці у сфері розробки, програмісти-ентузіасти.	Інші архітектурні підходи.
Висновки	Інтенсивність незначна, оскільки в більшості подібні рішення використовуються в навчальних цілях чи дослідженні.	Потенційний конкурент може створити систему з кращою інтеграцією. Тому слід працювати над новим функціоналом.	Прямий вплив від постачальників відсутній, однак потрібні оновлення.	Користувачі є найважливішою складовою, оскільки саме вони формують популярність та є рушієм розвитку.	Не є загрозою, однак можуть відволікати користувачів.

За допомогою вищезазначених таблиць – можна виконати обґрунтування факторів конкурентоспроможності (табл. 4.10)

Таблиця 4.10

Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проєктів значущим)
1	Унікальність пропозиції	Архітектурний підхід з використанням Vue, React та платформи Single-SPA рідко зустрічається серед іншим веб-застосунків
2	Якість технічної реалізації	Сучасна платформа Single-SPA і збирач Webpack дозволяє забезпечити високу сумісність та стабільність
3	Зручність у використанні	Мікрофронтендний підхід дозволяє отримати швидше завантаження сторінок для користувача завдяки модульності
4	Масштабування	Використана архітектура дозволяє швидко виконувати процес оновлення окремих модулів
5	Цінова складова	Для навчальний закладів безплатна базова версія, а для доступу до окремих частин коду – платна основа.

Аналіз сильних та слабких сторін стартап-проєкту відображено у табл. 4.11.

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін демонстраційного односторінкового веб-застосунку, який реалізовано з використання мікрофронтендної архітектури

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з демонстраційним односторінковим веб-застосунку, який реалізовано з використання мікрофронтендної архітектури						
			–3	–2	–1	0	+1	+2	+3
1	Унікальність пропозиції	19							+
2	Якість технічної реалізації	16					+		
3	Зручність у використанні	16				+			
4	Масштабування	20							+
5	Цінова складова	15					+		

Попередній аналіз дав змогу скласти SWOT-матрицю, яка наведена в табл. 4.12.

Таблиця 4.12

SWOT- аналіз стартап-проєкту

Сильні сторони: - можливість інтеграції різних фреймворків та модулів; - висока масштабованість; - використана сучасна архітектура.	Слабкі сторони: - складність конфігурування; - потреба в базових знаннях.

Таблиця 4.12 (продовження)

SWOT- аналіз стартап-проєкту

Можливості: - розвиток як навчальної чи демонстраційної платформи.	Загрози: - неактуальність обраних технологій в майбутньому; - проблеми з фінансування під час вдосконалення демонстраційного веб-застосунку; - поява великих конкурентів.

Розроблений SWOT-аналіз стартап-проєкту дозволяє визначити альтернативи ринкового впровадження (табл. 4.13).

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проєкту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Демонстраційні презентації для освітніх закладів та початківців на ІТ-курсах	Середня	2-3 місяці
2	Для залучення більшої кількості розробників – розміщення проєкту на GitHub	Середня	1-2 місяці
3	Розробка навчальних відеороликів, де пояснюється цінність реалізованого архітектурного рішення	Висока	4-6 місяців
4	Інтегрування демонстраційного веб-застосунку в навчальну програму закладів освіти	Висока	5-7 місяців

Табл. 4.13 демонструє, що строк реалізації перших двох варіантів найменший та ймовірність отримання ресурсів середня. Але для довготривалої перспективи доречніше буде використати третій та четвертий варіанти, оскільки саме вони дають найбільшу ймовірність залучити ресурси, хоча й з більшим строком реалізації.

4.4 Розроблення ринкової стратегії проєкту

Розробка ринкової стратегії проєкту буде описувати за допомогою табл. 4.14-4.17.

Таблиця 4.14

Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Студенти	Найбільш вмотивована цільова група, оскільки постійно шукає нові підходи у процесі навчання	Високий	Середня	Легка
2	Початківці у сфері веб-розробки	Висока, оскільки вони готові навчатись на проєктах будь-якої складності	Високий	Середня	Легка

Таблиця 4.14 (продовження)

Вибір цільових груп потенційних споживачів

3	Викладачі в освітніх закладах/освітніх платформах	Середня. Потрібно виконувати адаптацію навчальної програми	Середній	Низька	Середня
4	Сучасні компанії/розробники-ентузіасти	Низька. Цікавляться, але не завжди використовують	Низький	Висока	Складний вхід
Які цільові групи обрано: Студенти, початківці у веб-розробки, викладачі в освітніх закладах/освітніх платформах, ентузіасти.					

За допомогою табл. 4.15 буде сформовано базову стратегію розвитку.

Таблиця 4.15

Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Розробка навчальних відеороликів, де пояснюється цінність реалізованого архітектурного рішення	Концентрований маркетинг	Можливість продемонструвати технічні переваги мікрофронтендної архітектури через освітній контент.	Стратегія спеціалізації: орієнтація на створенні якісний навчальних матеріалів, де показана цінність використаних рішень

Далі буде визначено базову стратегію конкурентної поведінки (табл. 4.16).

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проєкт «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні, дане архітектурне рішення вже застосовується (найбільш поширено у великих корпораціях)	Компанії легше залучати нових споживачів.	Компанія може частково запозичувати підходи до проєктування від конкурентів, однак обов’язково має доповнювати власними рішення.	Стратегія наслідування лідеру

Після визначення базової стратегії конкурентної поведінки необхідно зазначити стратегію позиціонування (табл. 4.17).

Таблиця 4.17

Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту (три ключових)
1	Проста та ефективна демонстрація проєктування і розгортання веб-застосунку з використанням підходів мікрофронтендів	Стратегія диференціації	Швидше підключення модулів, гнучке масштабування, використання різних технологій для проєктування одного застосунку.	Гнучкість, інноваційний навчальний підхід, доступність

4.5 Розроблення маркетингової програми стартап-проєкту

Визначаю ключові переваги концепції потенційного товару. Результати наводжу за допомогою табл. 4.18.

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	У простому розумінні архітектури мікрофронтендів без складної теорії	Інтерактивна практична демонстрація дозволяє скоротити час на навчання	Легка структура проєкту, яка буде зрозуміла початківцях у галузі веб-розробки
2	У проведеному тестуванні кількох фреймворків в межах одного веб-застосунку	Дозволяє розглянути комбінацію використання Vue та React без виникнення конфліктів	Можлива подальше комбінування інших веб-фреймворків в межах одного застосунку
3	У навчальному інструменті для освітніх закладів	Надає можливість наочно побачити архітектурний підхід та повторити реалізацію на практиці	Документація

Також слід забезпечити розробку популярної трирівневої маркетингової моделі, вигляд якої представленого за допомогою табл. 4.19.

Таблиця 4.19

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Проєктування, розробка та розгортання демонстраційного односторінкового веб-застосунку, який реалізовано з використання мікрофронтендної архітектури. Система створена за допомогою платформи Single-SPA та поєднує в собі використання Vue.js і React. Розгорнений застосунок демонструє переваги співіснування кількох незалежних частинок фронтенду на одній веб-сторінці.		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Існуючий приклад інтеграції Vue та React компонентів.	М	Вр/Тх
	2. Демонстрація застосування платформи Single-SPA та збирача WebPack у якості динамічного підвантаження окремих модулів.	М	Вр/Тх
	3. Існуюча можливість інтегрування нових фреймворків для наявної системи.	М	Вр/Тх

Таблиця 4.19 (продовження)

Опис трьох рівнів моделі товару

	4. Швидке та зручне масштабування наявного веб-застосунку.	М	Вр/Тх
	5. Документація, яка включає опис структури веб-застосунка.	М	Вр/Тх
	Якість: створена система протестована на забезпечення швидкого завантаження компонентів та відсутності конфліктів між фреймворками/іншими інструментами проєктування.		
	Пакування: документація про проєктуванню та розгортанню яка міститься у пояснювальній записці; цифровий продукт – демонстраційний односторінковий веб-застосунок.		
	Марка: «MicroFronts»		
ІІІ. Товар із підкріпленням	До продажу: вимоги відсутні.		
	Після продажу: забезпечення підтримки шляхом усування багів та інтеграції нових фреймворків; розширення функціональної можливості демонстраційної копії.		
За рахунок чого потенційний товар буде захищено від копіювання: права на інтелектуальну власність			

Визначаю межі встановлення ціни. Результат наводжу в табл. 4.20.

Таблиця 4.20

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	Безкоштовні проєкти з вихідним програмним кодом	Навчальні чи комерційні шаблони (приклад проєкту, ціна від 1 тис. грн.)	Середній дохід, оскільки основну частку складають студенти, викладачі та початків у сфері розробки/ентузіасти	Для нижньої межі – безкоштовний базовий доступ без майбутньої підтримки; для верхньої – розширена платна (200 – 500 грн/в сезон оновлень) підтримка, майбутні додаткові модулі.

Визначення системи збут представлено у табл. 4.21.

Таблиця 4.21

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Користувачі мають можливість користуватись	Постачальник має забезпечити безперебійний доступ до	Канал нульового	За допомогою

	безкоштовною базовою версією, а додаткові можливості (доступ до розширеної документації, майбутні оновлення та підтримка) – платні.	демонстраційного веб-застосунку. Також він має слідкувати за оновлення, технічною підтримкою та масштабуванням (збільшенню кількості нових фреймворків для проєктування нових користувацьких модулів).	збуту.	сервісів збуту або офіційну платформу розробника.

За допомогою табл. 4.22. розроблено концепції маркетингових комунікацій.

Таблиця 4.22

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Студенти спеціальності веб-розробка	Соціальні мережі (Instagram, Facebook, а також платформи навчальних закладів)	Просте розуміння архітектури мікрофронтендів	Привернути увагу користувача до переваг та можливостей мікрофронтендів на базі демонстраційного веб-застосунку	Інтерфейс веб-застосунку – це завжди про швидкість

Таблиця 4.22 (продовження)

Концепція маркетингових комунікацій

2	Викладачі навчальних закладів	Професійні спільноти, технічні блоги	Зручне навчання студентів на демонстраційному веб-застосунку	Інформувати викладачів про сучасніші методи навчання	Швидко вчись та навчай
3	Розробники початківці, які бажають освоїти сучасні технології	Telegram-канали, форуми, GitHub	Розуміння архітектурних підходів завдяки можливості побачити на власні очі	Надати можливість власноруч спробувати систему	Вчись на реальному у досвіді

Висновки до розділу 4

Під час написання цього розділу магістерської дисертації було описано характеристику стартап-проєкту, проведено порівняльний аналіз показників існуючих архітектурних рішень для реалізації клієнтських частин, який показав гірші, аналогічні та кращі значення власної іде проєкту. Було проведено технологічний аудит, який забезпечив визначення середовищ, інструментів та технологій, потрібних для реалізації архітектури мікрофронтендів. Він показав, що реалізувати демонстраційний односторінковий веб-застосунок з використанням архітектурного підходу мікрофронтендів можна за допомогою значної кількості сучасних технологій, отже реалізація проєкту є можливою. Зважаючи на простоту реалізації, доступність технологій та реалізацію проєкту одноосібно – вибір було здійснено на користь Vue, React, Single-SPA та Webpack.

У ході роботи було визначено ринкові можливості, які використовують під час впровадження, а також ряд загроз, які перешкоджають впровадженню проєкту. Проведено аналіз ринкового середовища за допомогою таблиць фактори загроз та фактори можливостей. Було виконано обґрунтування факторів конкурентоспроможності та здійснено аналіз сильних і слабких сторін, визначено ринкову стратегію проєкту. При завершенні написання розділу – розроблено маркетингову програму стартап-проєкту, у процесі якої визначено ключові переваги концепції потенційного товару, встановлено цінові межі, сформовано систему збуту, а також розроблено концепції маркетингових комунікацій.

Проведений аналіз показав, що проєкт має перспективи успішного впровадження. Це пояснюється тим, що потенційно існують великі групи клієнтів, однак основним бар’єром може стати велика конкуренція, а також потреба в залученні висококваліфікованих веб-розробників. Оскільки

проєкт розроблено з використанням сучасних технологій, то рівень конкурентоспроможності такого продукту досить високий. Для мінімізації ризиків буде обрано стратегію поступового провадження.

ВИСНОВКИ

Розроблена магістерська дисертація присвячена аналізу підходів проєктування та реалізації застосунку на основі мікрофронтендів. Проєктування даної архітектури забезпечено за рахунок таких технологій, як Vue, React, а також платформи Single-SPA із збірником WebPack. Результатами дослідження є не тільки розробка демонстраційного веб-застосунку, а також оцінка його ефективності.

Під час роботи над дисертацією було досліджено монолітну архітектуру та розглянуто її обмеження, які включають складнощі при масштабуванні та розширенні функціональних потреб. Виявлено, що недоліки моноліту посприяли розвитку мікросервісної архітектури для розробки на серверному (не клієнтому) рівні, що у свою чергу лягло у принципи мікрофронтендів. Було розглянуто переваги мікрофронтендів, такі як: зручність при масштабуванні, можливість розробки веб-застосунку великою командою розробників та незалежному розгортанні. У той же час проаналізовано ряд недоліків (для яких уже існують напрацьовані шляхи вирішення): підвищені вимоги до архітектури, складність планування на початковому етапі та проблеми з єдиним стилем користувацького інтерфейсу.

Також проведено дослідження таких архітектурних підходів, як мікрофронтенди через маршрутизацію, мікрофронтенди через Web Components, патерн Module Federation та підхід компонування інтерфейсу на стороні сервера. У ході цього дослідження обрано реалізацію через маршрутизацію за допомогою інтеграції через Single-SPA та збіркою Webpack, оскільки це просто в інтеграції, гнучко та дозволяє ефективно керувати процесом розробки одноосібно.

Далі було проведено порівняльний аналіз платформ Single-SPA, Luigi, Frint.js та Piral, які надають інструменти для реалізації мікрофронтендів. Визначено, що Single-SPA є достатньо гнучким та має

значну підтримку різноманітних фреймворків. Розглянуто, що фреймворк Luigi потужний інструмент для проєктування та розробки, однак менш гнучкий за Single-SPA та погано підтримується розробниками. Фреймворк Frint.js добре зарекомендував себе у якості масштабування застарілих застосунків, однак у нього відсутня підтримка Vue.js. Також проведений аналіз показав, що фреймворк Piral містить зручні та корисні інструменти для побудови сучасних мікрофронтенді, але сильно залежить від екосистеми React та має складну конфігурацію оболонки. Тому було обрано платформу Single-SPA, бо вона найоптимальніша для проєктування веб-застосунку на мікрофронтендній архітектурі, оскільки забезпечила гнучку інтеграцію таких фреймворків як Vue + React, а також надала можливість експериментувати з різноманітними інструментами.

Під час проєктування було виділено вимоги до системи та здійснено опис бізнес процесів. Змодельовано процеси завдяки використанню нотацій IDEF0 та BPMN. Визначено конкретні компоненти та зв’язки між ними, описано реалізацію окремих модулів застосунку та здійснено тестовий запуск.

У процесі тестування було детально проаналізовано продуктивність веб-застосунку, яке складалось з оцінки розміру скомпільований файлів та використання пам’яті, що засвідчило відсутність витоків. Також було наведено порівняння продуктивності монолітної та мікрофронтендної архітектур, яке показало, що остання забезпечила кращу швидкодію, інтерактивність, стабільність та хороші показники Core Web Vitals. Зроблені висновки демонструють, що монолітна архітектура може спростити розробку тільки на перших етапах проєктування та розгортання, а надалі при масштабуванні – погіршується рендеринг та нестабільність макету. Аналіз засвідчив, що архітектура мікрофронтендів ізолюючи модулі, дозволяє самостійне розгортання кожного з елементів сайту, однак варто зазначити збільшення Total Blocking Time, цього можна уникнути через розділення бандлів та використання кешування. Отримані результати

підтверджують, що застосунок демонструє стабільність, відповідну швидкодію та є відкритим до подальшого масштабування.

Заключний етапом магістерської дисертації стала розробка стартап-проєкту. Тому під час роботи над ним надано характеристику стартап-проєкту, проведено порівняльний аналіз показників існуючих архітектурних рішень для реалізації клієнтських частин, який показав гірші, аналогічні та кращі значення власної іде проєкту. Здійснено технологічний аудит, який забезпечив визначення середовищ, інструментів та технологій, потрібних для реалізації архітектури мікрофронтендів. Визначено ринкові можливості, які використовують під час впровадження, а також ряд загроз, які перешкоджають впровадженню проєкту. Проведено аналіз ринкового середовища за допомогою таблиць фактори загроз та фактори можливостей. По завершенню написання стартап-проєкту – розроблено маркетингову програму, у процесі якої визначено ключові переваги концепції потенційного товару, встановлено цінові межі, сформовано систему збуту, а також розроблено концепції маркетингових комунікацій.

Виконаний аналіз показав, що проєкт має перспективи успішного впровадження, оскільки він реалізований за допомогою сучасних технологій, а також існують великі групи клієнтів. Досліджено, що основним бар’єром може стати велика конкуренція, а також потреба в залученні висококваліфікованих веб-розробників.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Evolution of Front-End Development: From HTML to Modern Frameworks [Електронний ресурс]. — Режим доступу: https://medium.com/@mattlcoleman88/the-evolution-of-front-end-development-from-html-to-modern-frameworks-110c995ec7fc — (11.10.2025).

2. How modern frontend development has evolved? Key trends from the State of Frontend 2020-2025 reports [Електронний ресурс]. — Режим доступу: https://tsh.io/blog/how-modern-frontend-development-has-evolved/ — (13.10.2025).

3. Монолітна архітектура ПЗ [Електронний ресурс]. — Режим доступу: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/ — (13.10.2025).

4. Архітектура модульного моноліту у Vue-застосунку [Електронний ресурс]. — Режим доступу: https://dou.ua/forums/topic/48721/ — (15.10.2025).

5. Мікросервісна архітектура [Електронний ресурс]. — Режим доступу: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d — (16.10.2025).

6. Порівняння мікросервісної та монолітної архітектур [Електронний ресурс]. — Режим доступу: https://foxminded.ua/porivnyannya-mikroservisnoyi-ta-monolitnoyi-arhitektur/ — (16.10.2025).

7. Застосування мікрофронтендної архітектури [Електронний ресурс]. — Режим доступу: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/afce736f-834c-4d8e-adba-8f8a6ff0d5f6/content — (16.10.2025).

8. Мікрофронтенди: переваги, підводні камені та практичні рішення [Електронний ресурс]. — Режим доступу: https://dou.ua/forums/topic/54035/ — (18.10.2025).

9. Microfrontends [Електронний ресурс]. — Режим доступу: https://webbylab.com/uk/news/microfrontends/ — (20.10.2025).

10. Handling Routing in a Microfrontend Architecture [Електронний ресурс]. — Режим доступу: https://webbylab.com/uk/news/microfrontends/ — (20.10.2025).

11. Designing Micro Frontend with Web Components [Електронний ресурс]. — Режим доступу: https://medium.com/@sathyaleo08/designing-micro-frontend-with-web-components-830a50f6c7fa — (20.10.2025).

12. Webpack 5 Module Federation: A game-changer in JavaScript architecture [Електронний ресурс]. — Режим доступу: https://angular.love/webpack-5-module-federation-a-game-changer-in-javascript-architecture — (22.10.2025).

13. Understanding Server-Side and Client-Side Composition in Microfrontends [Електронний ресурс]. — Режим доступу: https://article.arunangshudas.com/understanding-server-side-and-client-side-composition-in-microfrontends-55814b5efe20 — (23.10.2025).

14. Micro Frontends: After one year with Single-SPA [Електронний ресурс]. — Режим доступу: https://dev.to/psamim/micro-frontends-after-one-year-with-single-spa-1eoo — (23.10.2025).

15. Micro frontends with Luigi [Електронний ресурс]. — Режим доступу: https://www.linkedin.com/pulse/micro-frontends-luigi-muni-kumar-gundu — (25.10.2025).

16. FrintJS Documentation [Електронний ресурс]. — Режим доступу: https://frint.js.org/ — (25.10.2025).

17. Piral - The Ultimate Framework for Micro Frontends [Електронний ресурс]. — Режим доступу: https://piral.io/ — (25.10.2025).

18. React Micro-Frontend Architecture using single-spa — EP03 [Електронний ресурс]. — Режим доступу: https://medium.com/@MastersTechie/react-micro-frontend-architecture-using-single-spa-ep03-eb707c6c1d53 — (25.10.2025).

19. How to add Vue (or another framework) to your React Micro... [Електронний ресурс]. — Режим доступу: https://rtbell.dev/blog/frontend/add-other-framework-to-micro-frontend — (25.10.2025).

20. How to render React components in Vue [Електронний ресурс]. — Режим доступу: https://dev.to/amirkian007/how-to-render-react-components-in-vue-1e0f — (25.10.2025).

21. What is BPMN [Електронний ресурс]. — Режим доступу: https://www.visual-paradigm.com/guide/bpmn/what-is-bpmn/ — (26.10.2025).

22. Creating Model Using IDEF0 [Електронний ресурс]. — Режим доступу: https://umalogic.com/help/docs/current/doku.php/en/manual/creating_model/notation_idef0 — (26.10.2025).

23. IDEF3 Process Description Capture Method [Електронний ресурс]. — Режим доступу: https://www.conceptdraw.com/How-To-Guide/idef3 — (26.10.2025).

24. What is Deployment Diagram [Електронний ресурс]. — Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/ — (26.10.2025).

25. Component-Based Diagram in Software Engineering [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/software-engineering/component-based-diagram/ — (26.10.2025).

26. Sequence Diagram Tutorial [Електронний ресурс]. — Режим доступу: https://creately.com/guides/sequence-diagram-tutorial/ — (27.10.2025).

27. Single-SPA: Getting Started Overview [Електронний ресурс]. — Режим доступу: https://single-spa.js.org/docs/getting-started-overview/ — (27.10.2025).

28. Single-SPA Configuration Guide [Електронний ресурс]. — Режим доступу: https://single-spa.js.org/docs/configuration/ — (27.10.2025).

29. Single-SPA React Ecosystem [Електронний ресурс]. — Режим доступу: https://single-spa.js.org/docs/ecosystem-react — (27.10.2025).

30. Single-SPA Vue Ecosystem [Електронний ресурс]. — Режим доступу: https://single-spa.js.org/docs/ecosystem-vue/ — (27.10.2025).

31. Create React App Documentation [Електронний ресурс]. — Режим доступу: https://create-react-app.dev/docs/getting-started/ — (28.10.2025).

32. Vue CLI Service Guide [Електронний ресурс]. — Режим доступу: https://cli.vuejs.org/guide/cli-service.html — (28.10.2025).

33. Build a React App from Scratch [Електронний ресурс]. — Режим доступу: https://react.dev/learn/build-a-react-app-from-scratch — (28.10.2025).

34. Analyzing the Bundle Size in Create React App [Електронний ресурс]. — Режим доступу: https://create-react-app.dev/docs/analyzing-the-bundle-size/ — (28.10.2025).

35. Vue CLI Deployment Guide [Електронний ресурс]. — Режим доступу: https://cli.vuejs.org/guide/deployment.html — (28.10.2025).

36. Web Performance Fundamentals [Електронний ресурс]. — Режим доступу: https://web.dev/learn/performance — (29.10.2025).

37. Chrome DevTools: Memory Analysis [Електронний ресурс]. — Режим доступу: https://developer.chrome.com/docs/devtools/memory — (29.10.2025).

38. Chrome Lighthouse Documentation [Електронний ресурс]. — Режим доступу: https://developer.chrome.com/docs/lighthouse — (29.10.2025).

ДОДАТОК А. ROOT-CONFIG.JS

```
System.import("single-spa").then((singleSpa) => {
  const { registerApplication, start } = singleSpa;

  registerApplication({
    name: "@mf/vue-app",
    app: () => System.import("@mf/vue-app"),
    activeWhen: ["/"],
  });

  registerApplication({
    name: "@mf/react-app",
    app: () => System.import("@mf/react-app"),
    activeWhen: ["/"],
  });

  start();
  console.log("single-spa started");
});
```

ДОДАТОК Б. ROOT.COMPONENT.JSX

```
import React, { useState, useEffect } from "react";
import Form from "./form";
import DataList from "./data-list";
import EditModal from "./edit-modal";

export default function Root() {
  const [list, setList] = useState([]);
  const [editItem, setEditItem] = useState(null);
  const [search, setSearch] = useState("");
  const [sort, setSort] = useState("name");
  const [toast, setToast] = useState(null);

  const showToast = (text, type = "success") => {
    setToast({ text, type });
    setTimeout(() => setToast(null), 2600);
  };

  useEffect(() => {
    const saved = localStorage.getItem("mf-list");
    const savedSort = localStorage.getItem("mf-sort");

    if (saved) setList(JSON.parse(saved));
    if (savedSort) setSort(savedSort);
  }, []);
  const save = (data) => localStorage.setItem("mf-list",
JSON.stringify(data));
  const saveSort = (v) => localStorage.setItem("mf-sort", v);

  const addRecord = (item) => {
    const updated = [...list, { id: Date.now(), ...item }];
    setList(updated);
    save(updated);
    showToast("☑ Запис додано");
  };

  const deleteRecord = (id) => {
    const updated = list.filter((x) => x.id !== id);
    setList(updated);
    save(updated);
    showToast("🗑 Видалено");
  };

  const updateRecord = (changed) => {
    const updated = list.map((x) => (x.id === changed.id ? changed : x));
    setList(updated);
    save(updated);
    setEditItem(null);
```

```
    showToast("✏️ Запис оновлено");
  };

  return (
    <div className="wrapper">
      <h2>React Microfrontend CRUD Pro</h2>

      <Form onSubmit={addRecord} />

      <div className="filters">
        <input
          placeholder="Пошук по імені..."
          value={search}
          onChange={(e) => setSearch(e.target.value)}
        />

        <label htmlFor="country"></label>
        <select
          id="country"
          value={sort}
          onChange={(e) => {
            setSort(e.target.value);
            saveSort(e.target.value);
          }}
        >
          <option value="name">Ім’я</option>
          <option value="age">Вік</option>
          <option value="city">Місто</option>
          <option value="createdAt">Дата додавання</option>
        </select>
      </div>

      <DataList
        data={list}
        search={search}
        sort={sort}
        onDelete={deleteRecord}
        onEdit={(item) => setEditItem(item)}
      />

      {editItem && (
        <EditModal
          data={editItem}
          onClose={() => setEditItem(null)}
          onSave={updateRecord}
        />
      )}
      {toast && <div className={`toast ${toast.type}`}>{toast.text}</div>}
    </div>
  );
}
```

ДОДАТОК В. ROOT.COMPONENT.JSX

```
<template>
  <div class="wrapper">
    <h2>Vue Productivity Notepad</h2>

    <Pomodoro />

    <AddNote @add-note="addItem" />

    <NotesList
      :notes="notes"
      @remove="removeItem"
      @toggle="toggleFav"
      @edit="openEditor"
    />

    <EditModal
      :visible="editVisible"
      :initialText="editText"
      :initialDeadline="editDeadline"
      @save="saveEdit"
      @cancel="editVisible = false"
    />

    <div v-if="toast.visible" :class="['toast', toast.type]">
      {{ toast.message }}
    </div>
  </div>
</template>

<script>
import { loadNotes, saveNotes } from "./utils/storage";
import AddNote from "./components/AddNote.vue";
import NotesList from "./components/NotesList.vue";
import EditModal from "./components/EditModal.vue";
import Pomodoro from "./components/Pomodoro.vue";

export default {
  components: { AddNote, NotesList, EditModal, Pomodoro },
  data() {
    return {
      notes: [],
      editIndex: null,
      editText: "",
      editDeadline: "",
      editVisible: false,
      toast: { visible: false, message: "", type: "" },
    };
```

```
  },
  mounted() {
    this.notes = loadNotes();
  },
  methods: {
    showToast(msg, type = "info") {
      this.toast = { visible: true, message: msg, type };
      setTimeout(() => (this.toast.visible = false), 2000);
    },
    addItem(text, deadline) {
      this.notes.push({
        id: Date.now(),
        text,
        deadline,
        fav: false,
      });
      saveNotes(this.notes);
      this.showToast("Нотатку додано ☑", "success");
    },
    removeItem(i) {
      const removed = this.notes[i].text;
      this.notes.splice(i, 1);
      saveNotes(this.notes);
      this.showToast(`Нотатку "${removed}" видалено 🗑`, "danger");
    },
    toggleFav(i) {
      this.notes[i].fav = !this.notes[i].fav;
      saveNotes(this.notes);
      const status = this.notes[i].fav
        ? "★ Додано у важливі"
        : "Видалено з важливих";
      this.showToast(status, "info");
    },
    openEditor(i) {
      this.editIndex = i;
      this.editText = this.notes[i].text;
      this.editDeadline = this.notes[i].deadline;
      this.editVisible = true;
    },
    saveEdit(text, deadline) {
      this.notes[this.editIndex].text = text;
      this.notes[this.editIndex].deadline = deadline;
      saveNotes(this.notes);
      this.editVisible = false;
      this.showToast("Нотатку відредаговано 🖉", "success");
    },
  },
};
</script>

<style scoped>
```

```
.toast {
  position: fixed;
  bottom: 20px;
  right: 20px;
  padding: 10px 16px;
  border-radius: 8px;
  color: white;
  font-weight: bold;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.25);
  opacity: 0.95;
  transition: 0.3s;
  z-index: 999;
}

.toast.success {
  background: #28a745;
}
.toast.danger {
  background: #dc3545;
}
.toast.info {
  background: #0077ff;
}
.vue-app {
  max-width: 440px;
  margin: 0 auto;
  padding: 25px;
  font-family: "Segoe UI", sans-serif;
  background: rgba(255, 255, 255, 0.55);
  border-radius: 14px;
  backdrop-filter: blur(10px);
  box-shadow: 0 0 25px rgba(0, 0, 0, 0.18);
  animation: fadeIn 0.6s ease;
}

.vue-app h2 {
  text-align: center;
  font-weight: 700;
  letter-spacing: 0.3px;
  background: linear-gradient(135deg, #0077ff, #00c2ff);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
}

button {
  cursor: pointer;
  border-radius: 6px;
  border: none;
  padding: 6px 12px;
  transition: 0.25s;
  font-size: 14px;
```

```
  background: linear-gradient(135deg, #00b873, #009457);
  color: white;
}

button:hover {
  filter: brightness(1.12);
}

.note-card {
  background: rgba(255, 255, 255, 0.65);
  border-radius: 8px;
  padding: 12px;
  margin-bottom: 10px;
  box-shadow: 0 0 6px rgba(0, 0, 0, 0.1);
  transition: 0.25s;
}

.note-card:hover {
  transform: scale(1.01);
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.15);
}

.note-card.fav {
  border-left: 6px solid gold;
  background: rgba(255, 244, 189, 0.85);
}

.deadline {
  font-size: 12px;
  color: #555;
  font-weight: 600;
}

.modal-bg {
  position: fixed;
  inset: 0;
  background: rgba(0, 0, 0, 0.45);
  display: flex;
  justify-content: center;
  align-items: center;
  animation: fadeIn 0.3s ease;
}

.modal {
  background: #fff;
  padding: 18px;
  border-radius: 10px;
  width: 280px;
  display: flex;
  flex-direction: column;
  gap: 8px;
```

```
  animation: pop 0.35s ease;
  box-shadow: 0 8px 30px rgba(0, 0, 0, 0.25);
}

@keyframes fadeIn {
  from {
    opacity: 0;
    transform: translateY(8px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

@keyframes pop {
  0% {
    transform: scale(0.75);
    opacity: 0;
  }
  100% {
    transform: scale(1);
    opacity: 1;
  }
}
</style>
```