ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Сучасні каркаси web-додатків
Лісняк Андрій Олександрович

Лабораторна робота 6. Робота з Vue Router

Мета

Навчитися пов’язувати URL із вкладеними представленнями Vue, завантажувати товар за реактивним параметром маршруту та будувати хлібні крихти з метаданих маршрутизатора.

Теоретичні відомості

Маршрутизація пов’язує адресу браузера з представленням застосунку. Від адреси залежать вибір компонентів, дані для відображення та стан, який можна відновити після переходу або оновлення сторінки. Тема розвиває поняття компонентів і композиційних функцій, розглянуті раніше, та узгоджена з лекцією 6 «Маршрутизація у Vue Router». Механізми викладено для Vue Router 4 і Composition API Vue 3.

1. Адреса та представлення в SPA

Односторінковий застосунок (SPA) — вебзастосунок, у якому після початкового завантаження документа JavaScript оновлює інтерфейс під час внутрішніх переходів.

Маршрутизація — зіставлення адреси браузера з представленням застосунку.

Запис маршруту — правило конфігурації, яке пов’язує шлях із компонентом або групою дочірніх маршрутів і може містити додаткові властивості навігації.

Vue Router узгоджує поточну адресу, показані компоненти та історію браузера. Його конфігурація визначає, яке представлення відповідає кожному шляху. Власна адреса потрібна станам інтерфейсу, до яких має бути можливий окремий перехід, зокрема через збережене посилання.

`createRouter()` створює маршрутизатор за конфігурацією `routes` і вибраною реалізацією `history`. Маршрутизатор підключається через `app.use(router)` до монтування застосунку.

Компонент RouterLink створює посилання для внутрішньої навігації без повного завантаження нового документа. Компонент `RouterView` визначає місце відображення компонента поточного маршруту. Звичайні компоненти всередині представлення зберігають свої пропси, події та слоти.

Іменований маршрут — запис маршруту з унікальним значенням `name`, яке дозволяє сформувати ціль переходу без записування повного шляху.

Ціль переходу може містити `name`, динамічні параметри `params` та параметри запиту `query`. Маршрутизатор будує адресу за конфігурацією й кодує

значення параметрів. Імена маршрутів унікальні в межах маршрутизатора. Для цілі з `path` параметри `params` ігноруються, тому динамічну адресу будують через `name` і `params` або передають уже сформований шлях.

Перехід починається з адреси або опису цілі. Маршрутизатор зіставляє шлях із конфігурацією, визначає параметри та перевіряє можливість переходу. Після його підтвердження поточний маршрут оновлюється, а `RouterView` відображає відповідні компоненти (рис. 1).

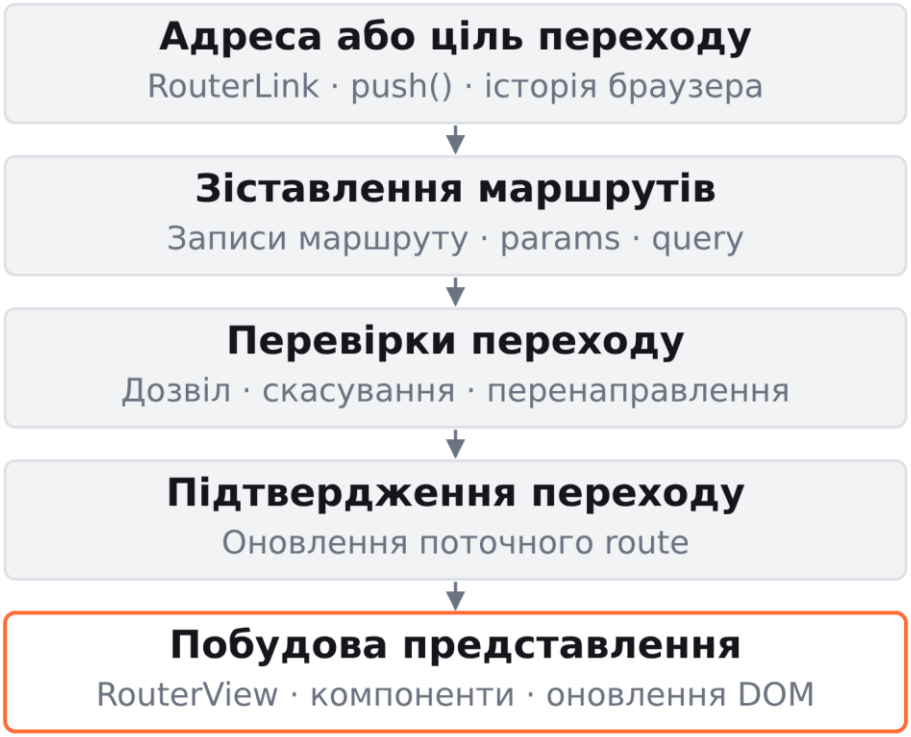

Рисунок 1 — Основні етапи побудови представлення під час успішного переходу.

Схема показує клієнтську навігацію після завантаження застосунку. Завантаження коду компонента та отримання даних API є окремими операціями. Дані можна отримати до підтвердження переходу або вже в показаному представленні.

2. Параметри та цілі переходу

Динамічний параметр — змінна частина шляху, позначена двокрапкою в конфігурації маршруту та доступна після зіставлення через `route.params`.

Параметри запиту URL (query) — значення після знака питання в адресі, які уточнюють стан представлення та доступні через `route.query`.

Фрагмент URL (hash) — частина адреси, що починається знаком `#` і доступна в поточному маршруті через `route.hash`.

Динамічний параметр може визначати ресурс, який відображається, а query може зберігати умови його представлення. Параметри звичайного динамічного сегмента надходять рядками; повторювані сегменти можуть давати масиви. Значення query також можуть бути рядками, масивами або `null`, а

відсутнє поле дає `undefined`. Перед використанням значення перевіряються й перетворюються до типу, якого потребує логіка застосунку.

Композиційна функція `useRoute()` надає реактивний об'єкт поточного маршруту. Функція `useRouter()` повертає маршрутизатор із методами навігації. Поточний маршрут описує стан представлення, а маршрутизатор виконує переходи та узгоджує їх з історією.

Перенаправлення (redirect) — правило, яке переводить навігацію з одного маршруту на іншу ціль.

Перенаправлення застосовується, коли адреса має вести до іншого маршруту, зокрема для вибору початкового представлення застосунку. Перенаправлення визначає нову ціль; гарди виконуються для цільового маршруту.

Активність посилання визначається зіставленим записом маршруту й відповідними `params`, без урахування `query` та `hash`. Клас `router-link-active` може позначати і посилання на батьківські рівні. Клас `router-link-exact-active` позначає точну активність без включення батьківських рівнів.

3. Модель історії та режими Vue Router

Історія сеансу браузера — послідовність записів навігації окремої вкладки або вікна з позицією поточного запису.

Історія має записи перед поточною позицією та після неї. Вона може містити переходи між різними документами й переходи всередині SPA. Тому кнопка «Назад» може повернути до іншого сайту, а не до батьківського рівня застосунку. Перехід до батьківського рівня задається явною ціллю.

Після початкового відкриття застосунку звичайний успішний перехід через `router.push()` додає запис після поточного. Якщо перед цим виконано повернення назад, колишні наступні записи вилучаються. Метод `router.replace()` замінює лише поточний запис і не збільшує довжину історії (рис. 2). `push()` підходить для окремого навігаційного кроку, `replace()` — для уточнення поточного стану, яке не має створювати додатковий крок повернення.



Рисунок 2 — Дія методів навігації на записи історії браузера.

Методи `router.back()` і `router.forward()` вибирають попередній та наступний записи. `router.go(n)` переміщує позицію на задану кількість кроків; від'ємне значення спрямовує назад, додатне — вперед. Ці операції не додають записів. Якщо потрібної позиції немає, перехід не відбувається.

У браузері History API надає `pushState()`, `replaceState()` і методи переміщення історією. `pushState()` та `replaceState()` можуть змінювати URL без завантаження нового документа. Переміщення між записами того самого документа породжує `popstate`, за яким застосунок відновлює представлення. Самі виклики `pushState()` і `replaceState()` цю подію не породжують. Vue Router організовує таку синхронізацію та перевірки переходу, тому навігація в ньому виконується через методи маршрутизатора.

Режим історії Vue Router — спосіб, у який маршрутизатор зберігає поточну адресу й записи переходів та пов'язує їх із середовищем виконання.

Режим задається властивістю `history` під час виклику `createRouter()`. Функції створення історії надають три способи її організації.

Функція	**Розміщення адреси маршруту**	**Зв'язок з історією браузера**
`createWebHistory`	Звичайний шлях URL	History API та кнопки браузера
`createWebHashHistory`	Фрагмент URL після #	Записи й кнопки браузера
`createMemoryHistory`	Внутрішній стан маршрутизатора	Адресний рядок і кнопки браузера не керують цією історією

Режим HTML5 history — режим маршрутизатора, у якому шлях представлення записується звичайною частиною URL та узгоджується з історією браузера.

`createWebHistory()` використовує цей режим. Пряме відкриття вкладеної адреси потребує правила повернення HTML-документа SPA на сервері. У режимі `createWebHashHistory()` шлях маршруту міститься після `#`. Цей фрагмент не надсилається серверу, тому окреме правило повернення документа для вкладених маршрутів не потрібне.

`createMemoryHistory()` зберігає власну послідовність записів у пам'яті. Автоматичний початковий перехід у ньому не запускається. Після підключення маршрутизатора початкову адресу задають явно через `router.push()` або `router.replace()`. Режим придатний для тестів і побудови HTML на сервері. Методи маршрутизатора переміщують його внутрішню позицію, але не змінюють історію вкладки браузера. Документація режимів історії розрізняє цю внутрішню історію та історію браузера.

Запис історії не є автоматично збереженим знімком стану компонентів. Повернення до URL відновлює навігаційний стан; локальні чернетки та завантажені дані потребують власного механізму збереження або повторного отримання.

4. Навігаційні гарди

Навігаційний гард (navigation guard) — функція, яка перевіряє перехід і може дозволити його продовження, скасувати або перенаправити.

Аргумент `to` описує ціль, а `from` — поточний маршрут. Повернення `false` скасовує перехід, повернення цілі починає інший перехід. `true`, `undefined` або відсутність поверненого значення дозволяють продовжити перевірки (рис. 3). Викинутий `Error` скасовує перехід і передається обробникам `router.onError()`. Асинхронний гард може очікувати результат операції; до її завершення перехід залишається непідтвердженим.

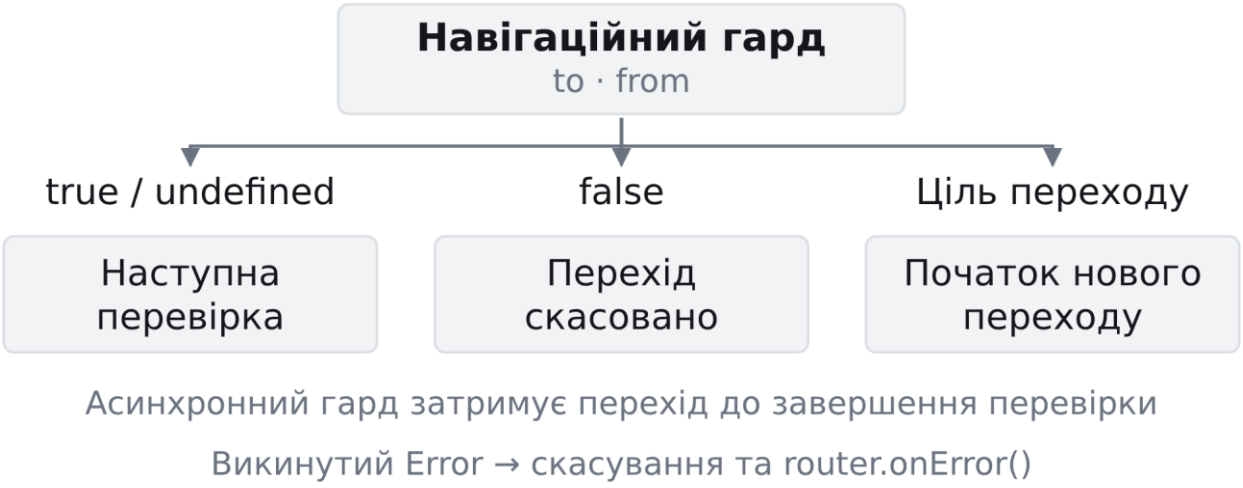

Рисунок 3 — Рішення гарда та продовження навігації.

Глобальний `router.beforeEach()` застосовується до переходів у всьому застосунку. `beforeEnter` у записі маршруту перевіряє вхід у цей маршрут. Зміна лише його `params`, `query` або `hash` не запускає `beforeEnter`. Перехід між дочірніми маршрутами зі спільним батьком також не повторює батьківський `beforeEnter`.

У Composition API `onBeforeRouteUpdate()` реагує перед оновленням маршруту, коли екземпляр компонента використовується повторно. `onBeforeRouteLeave()` перевіряє вихід із представлення і може скасувати його за наявності незбережених змін. Обидві функції реєструються в `setup` або `script setup` і мають доступ до локального стану компонента. В `onBeforeRouteUpdate()` нові параметри читаються з `to`, оскільки поточний `route` ще не оновлено.

`beforeRouteEnter` належить Options API й виконується до створення екземпляра компонента, тому не має доступу до нього через `this`. Окремого `onBeforeRouteEnter()` у Composition API немає. Підтримується також форма гарда з аргументом `next`, який викликається один раз на кожному шляху виконання. Для `beforeRouteEnter` виклик `next(callback)` дозволяє отримати створений екземпляр пізніше.

Послідовність успішної навігації охоплює такі етапи.

1. Перехід ініціюється; у компонентах, що залишають маршрут, виконуються гарди виходу.
2. Виконуються глобальні `beforeEach`, гарди оновлення повторно використаних компонентів, потім `beforeEnter` у нових записах маршруту.
3. Завантажуються асинхронні компоненти маршрутів; виконуються їхні `beforeRouteEnter`.
4. Виконуються глобальні `beforeResolve`, після чого перехід підтверджується.
5. Викликаються глобальні `afterEach` і запускається оновлення DOM.
6. Виконуються зворотні виклики, передані через `next` у `beforeRouteEnter`, зі створеними екземплярами компонентів.

`afterEach` є обробником після навігації й не може її скасувати. Третій аргумент `failure` містить збій навігації, якщо перехід було перервано. Методи `push()` і `replace()` повертають Promise, завершення якого дозволяє дочекатися результату.

Збої навігації перевіряються через `isNavigationFailure()` і `NavigationFailureType`. `aborted` означає відмову гарда через `false`, `cancelled` — початок іншого переходу до завершення поточного, `duplicated` — перехід до вже поточного маршруту. За успіху, зокрема після успішного перенаправлення, результат Promise не містить такого збою. Непередбачений виняток відхиляє Promise; це інший випадок, ніж звичайний збій навігації. Клієнтська перевірка доступу не замінює перевірку прав на сервері.

Гард керує можливістю переходу, а спостерігач `watch()` за властивістю `route` реагує на вже підтверджену зміну маршруту. Завантаження в гарді може

затримувати відкриття представлення до отримання даних. Завантаження після підтвердження дозволяє відразу показати оболонку та індикатор очікування. Обидва підходи потребують обробки помилок і визначеного власника даних.

5. Вкладені маршрути та спільна оболонка

Вкладений маршрут — дочірній запис у масиві `children`, який описує навігаційний рівень усередині батьківського маршруту.

Оболонка представлень (layout) — компонент зі спільною розміткою та місцем для змінного дочірнього вмісту.

Якщо батьківський маршрут має компонент, його вкладений `RouterView` показує компонент дочірнього маршруту. Зміна дочірнього представлення може зберігати оболонку, її розмітку та дані. Дочірній шлях без початкової косої риски доповнює шлях предка; порожній дочірній шлях визначає вміст за адресою самого батьківського рівня. Ім'я цього дочірнього маршруту дозволяє явно обрати його під час іменованого переходу.

Шлях дочірнього запису з початковою `/` є абсолютним, хоча компонент може лишатися вкладеним. Батьківський запис без `component` групує маршрути; `RouterView` пропускає цей рівень. Схема на рис. 4 показує випадок, коли батьківський компонент задано, як у розділі Nested Routes.

Властивість props у записі маршруту дозволяє відокремити компонент від `useRoute()`. За `props: true` параметри `route.params` передаються як пропси; функція в `props` може обчислити потрібні значення з маршруту.

Спільні дані доцільно зберігати в компоненті, який потрібен усім представленням, що їх використовують. Слот RouterView надає значення `Component`, яке відповідає компоненту обраного маршруту. Його можна відобразити через `<component :is="…">` і передати спільні дані пропсами. Такий підхід потребує узгодженого контракту пропсів у відповідних дочірніх компонентах.

Маршрутизатор визначає, що відображати, але не створює спільне сховище даних автоматично. Завантаження та зміну даних організовує їхній власник, а дочірні представлення використовують отримані значення.

Ієрархія записів маршруту перетворюється на дерево представлень (рис. 4). Батьківський компонент містить вкладений `RouterView`, а дочірній компонент займає його місце. За переходу між дочірніми представленнями спільна оболонка може залишатися тим самим екземпляром.

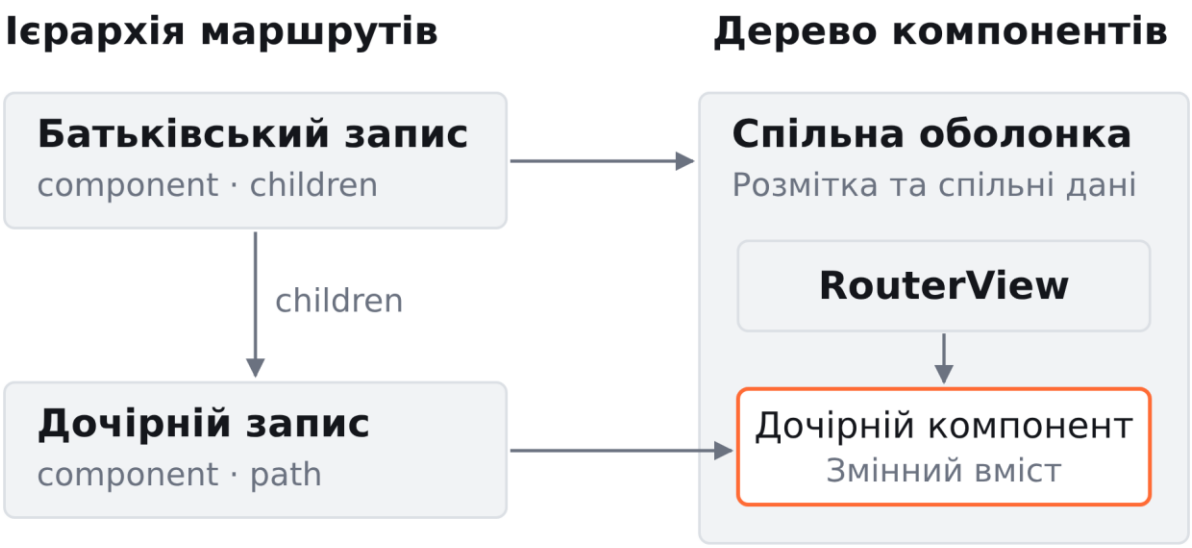

Рисунок 4 — Зв’язок ієрархії маршрутів із вкладенням компонентів.

6. Реактивне завантаження за маршрутом

За зміни параметра того самого запису маршруту Vue Router повторно використовує екземпляр компонента, якщо його відтворення не примусово змінено, наприклад через `key`. Звичайна зміна параметра не спричиняє нового монтування. Тому завантаження лише в `onMounted()` не забезпечує оновлення даних після зміни параметра.

Джерелом завантаження може бути гетер потрібного параметра або обчислюване значення `computed()`. Спостерігач `watch()` реагує на зміну цього джерела; параметр `immediate: true` запускає початковий виклик. Реактивне джерело передається гетером або ref, а не вже прочитаним рядком із `route.params`. Такий підхід наведено в Data Fetching та Composition API. Спостереження за конкретною залежністю запобігає повторним запитам через зміни маршруту, які не впливають на дані. Логіку завантаження можна винести в композиційну функцію, яка приймає реактивне джерело та повертає дані, стан завантаження й помилку.

Асинхронні відповіді можуть надходити в іншому порядку, ніж запускалися запити. `AbortController` дозволяє скасувати попередній `fetch()`. Скасування реєструється як очищення побічного ефекту через аргумент `onCleanup` callback-функції `watch()`. Очищення виконується перед повторним запуском спостерігача та під час його зупинки. Перевірка актуальності операції перед записуванням результату додатково захищає стан від застарілої відповіді. Застаріла операція також не повинна змінювати помилку або завершувати індикатор нового завантаження.

Після зміни ресурсу його попередні дані очищуються або явно позначаються як застарілі. Стан завантаження, мережева помилка, відповідь про відсутній ресурс і успішна відповідь без елементів мають різне значення для інтерфейсу. fetch() не відхиляє Promise лише через HTTP-статус помилки, тому перевіряються `response.ok` і `response.status`.

7. Метадані та хлібні крихти

Метадані маршруту — додаткова інформація в полі `meta` запису маршруту, яку використовує логіка застосунку.

Хлібні крихти — упорядкований навігаційний ланцюжок батьківських рівнів і поточної сторінки, який показує її місце в ієрархії застосунку.

Масив `route.matched` містить записи, зіставлені з адресою, від батьківського до дочірнього. Їхні поля `meta` дозволяють описати підписи та цілі окремих рівнів. Об'єкт route.meta є поверхневим, нерекурсивним об'єднанням метаданих цих записів. Значення однакових ключів дочірнього запису перекривають батьківські, тому об'єднаний об'єкт не замінює перелік рівнів.

Склад метаданих для крихт визначається застосунком. Vue Router не задає вбудованого формату підписів або правил побудови ланцюжка. Записи, які не представляють окремого навігаційного рівня, можна пропускати, щоб спільний підпис не дублювався.

Якщо підпис залежить від завантажених даних, він обчислюється реактивно. До отримання відповіді використовується тимчасовий підпис, пов'язаний із поточним ресурсом. Після завантаження він оновлюється. Передавання даних або підпису від їхнього власника дозволяє побудувати навігацію без окремого HTTP-запиту. Поділ тексту URL за косою рискою не відтворює метадані та назви, отримані із зовнішнього джерела.

Шаблон WAI-ARIA для крихт передбачає `nav` із доступною назвою. Крихти можна подати впорядкованим списком `ol`. Попередні рівні є посиланнями, а поточний рівень може бути текстом. Для посилання поточної сторінки задається `aria-current="page"`; для текстового елемента цей атрибут необов'язковий. Декоративні роздільники приховуються від читача екрана. Tab переміщує фокус між посиланнями, Enter виконує перехід. Довгі підписи мають переноситися без обрізання.

8. Стан інтерфейсу в URL

URL може зберігати стан, який має відновлюватися за посиланням, зокрема умови пошуку, сортування або вибраний розділ. Чернетка введення та тимчасовий стан взаємодії можуть залишатися локальними. Це дозволяє відокремити поточне редагування від підтвердженого стану представлення.

Відкладений виклик (debounce) — виконання дії після заданої паузи без нових викликів, коли кожен новий виклик перезапускає очікування.

Вибір полів URL, тривалості debounce й правил скасування належить логіці застосунку. Vue Router надає реактивний маршрут і переходи, але не реалізує цю синхронізацію автоматично.

Під час введення debounce дозволяє відкласти оновлення URL і запуск запиту до завершення паузи. Зміна чернетки може негайно скасовувати запит, результат якого вже не відповідає введенню. Якщо після скасування підтверджене значення лишилося тим самим, для повторного завантаження потрібен окремий запуск, бо значення джерела спостерігача не змінилося. За

звичайного переходу до вже поточного маршруту Vue Router повертає збій `duplicated`. Сама така навігація не є способом повторити скасований запит.

Синхронізація має два напрямки. Дії в інтерфейсі оновлюють маршрут, а зміна маршруту через посилання або історію браузера відновлює інтерфейс. Відновлення не повинно запускати відкладений запис попередньої чернетки назад у URL. Під час переходу потрібні query-параметри передаються явно, оскільки довільні поля query не успадковуються автоматично.

Відновлення стану з URL не означає збереження завантажених даних або локального стану компонентів. Після знищення екземпляра компонента його локальні значення втрачаються, якщо для них не передбачено окремого збереження.

9. Пряме відкриття та невідомі адреси

Внутрішній перехід обробляє Vue Router. Після оновлення сторінки або прямого відкриття адреси браузер спершу запитує документ у сервера. У режимі HTML5 history сервер має повернути початковий HTML-документ застосунку. Після його завантаження Vue Router вибирає представлення. Правило такого повернення налаштовується на вебсервері; воно має зберігати окрему обробку статичних файлів та API.

Універсальний маршрут (catch-all) — запис із шаблоном, який охоплює довільну послідовність сегментів шляху.

Шаблон `/:pathMatch(.*)*` дозволяє показати представлення для адрес, які не відповідають конкретним маршрутам. Невідома адреса застосунку та відповідь API про відсутній ресурс є різними ситуаціями й обробляються окремо. Представлення «Сторінку не знайдено» саме по собі не задає HTTP-статус документа. Статична SPA може показувати його після отримання HTML-документа зі статусом 200.

Функціональні вимоги

- Каталог із роботи 5 зберігає пошук, негайне скасування запиту та дозавантаження; картка відкриває окрему сторінку свого товару без перезавантаження документа
- Адреса /products/:id показує зображення, назву, ціну, рейтинг, наявність та огляд товару; адреса /products/:id/reviews показує відгуки в тій самій оболонці
- Перехід між оглядом і відгуками одного товару не виконує додаткового запиту товару; зміна id оновлює дані та скасовує попереднє завантаження
- Завантаження, мережева помилка, відсутній товар і порожній список відгуків мають окремі стани; помилковий запит можна повторити
- Крихти огляду мають рівні Каталог / Назва товару, крихти відгуків — Каталог / Назва товару / Відгуки, сторінки /about і 404 — власний

рівень Про застосунок або Сторінку не знайдено; попередні рівні є посиланнями, останній рівень є текстом поточної сторінки
- Під час завантаження крихта товару містить поточний id; після відповіді з'являється назва; дані й назва попереднього товару не відображаються для нового id
- Пошук після 400 мс без нового введення оновлює query-параметр q; готовий URL, reload і переходи назад та вперед відновлюють поле й умову пошуку
- Посилання картки, розділів і крихт зберігають q, тому повернення до каталогу відновлює пошук; накопичені сторінки списку після повернення завантажуються заново
- Кореневий шлях перенаправляє на каталог, /about показує інформацію про застосунок, а невідомий шлях показує окрему сторінку 404
- Пряме відкриття й оновлення адрес огляду та відгуків працюють без попереднього відвідування каталогу
- За розмірів 360×800, 768×1024, 1440×900 і 844×390 немає горизонтальної прокрутки та обрізаних крихт; довгі назви переносяться, посилання доступні з клавіатури
- Сторінка містить видимі ПІБ, групу й відомості про середовище виконавця

Технічні вимоги

- Проєкт lab6 створено на основі власного lab5 без копіювання node_modules, dist і вкладеного .git; використано Vue 3 SFC, JavaScript, Composition API та script setup
- Vue Router 4.6.4 встановлено як dependency, використано createRouter і createWebHistory, маршрутизатор підключено до монтування застосунку; package-lock.json збережено
- Конфігурація містить іменовані маршрути products, product-overview, product-reviews та about, кореневий redirect і catch-all маршрут /:pathMatch(.*)*
- Ієрархію задано через children з ProductsLayout, ProductLayout і порожніми дочірніми шляхами; дочірні шляхи :id та reviews не починаються з косої риски
- App містить загальну оболонку та зовнішній RouterView; ProductLayout володіє даними товару й передає product дочірнім представленням через слот RouterView
- Навігацію виконано через RouterLink та іменовані цілі з params і query; useRoute надає поточний маршрут, а useRouter — команди навігації
- URL API товару обчислюється за поточним id; useFetch очищує старі дані, перевіряє response.ok, зберігає HTTP-статус помилки й не застосовує застарілу відповідь
- Некоректний id не створює запиту до API; useFetch підтримує відсутній URL і очищення запиту під час розмонтування

- Крихти задаються власним полем meta.breadcrumb; useBreadcrumbs обчислює їх за route.matched та реактивною назвою товару, порожні дочірні маршрути не додають дублікатів
- Breadcrumbs отримує масив елементів пропсом items і не виконує HTTP-запитів; використано nav із доступною назвою, ol, aria-current=page та приховані від читача екрана роздільники
- Поле пошуку є чернеткою вводу, q є підтвердженою умовою запиту; replace оновлює пошук без нового запису історії, зовнішня навігація не запускає зворотний запис старого вводу
- Стилі каталогу збережено, нові елементи оформлено BEM-класами; готові компоненти breadcrumbs, Pinia, localStorage і ручний поділ URL для крихт не використовуються

Необхідне обладнання та матеріали

- Node.js 24 LTS, npm, Git і редактор коду
- Власний Vue-проєкт лабораторної роботи 5 із каталогом і composables
- Vue Router 4.6.4 із локальним підключенням через npm
- Браузер із DevTools, панеллю Network та емуляцією розмірів екрана
- Доступ до публічного API DummyJSON

Хід роботи

1. Підготовка проєкту

Скопіюйте власний проєкт із лабораторної роботи 5 з каталогу `lab5` до нового каталогу `lab6` у корені репозиторію `mjsf-mag`. Не копіюйте `node_modules`, `dist` і вкладений каталог `.git`.

Після копіювання виконайте окремими командами:

```
cd lab6
npm ci
npm install --save-exact vue-router@4.6.4
npm run dev -- --port 5176 --strictPort
```

Відкрийте http://localhost:5176. Перевірте початковий каталог, пошук, скасування запиту й дозавантаження з роботи 5. Збережіть ПІБ, групу та відомості про середовище у видимій частині сторінки. Початковий результат має відповідати власному lab5. Dev-сервер залиште запущеним для наступних етапів; команди в іншому терміналі виконуйте з каталогу lab6.

2. Розподіл на оболонки та представлення

У README зафіксуйте відповідальність модулів і дерево маршрутів. Збережіть компоненти ProductSearchForm, ProductCatalog, ProductCard і CatalogLoadMore з роботи 5. Перенесіть координацію пошуку, запитів та

дозавантаження з App до ProductCatalogView. App має містити загальну шапку, основну навігацію, підвал і зовнішній RouterView.

Додайте таку структуру до наявного src:

```
src/
├── router/
│   └── index.js
├── layouts/
│   ├── ProductsLayout.vue
│   └── ProductLayout.vue
├── views/
│   ├── ProductCatalogView.vue
│   ├── ProductOverviewView.vue
│   ├── ProductReviewsView.vue
│   ├── AboutView.vue
│   └── NotFoundView.vue
├── components/
│   └── Breadcrumbs.vue
└── composables/
    └── useBreadcrumbs.js
```

ProductsLayout містить RouterView для каталогу або оболонки товару. ProductLayout містить спільну інформацію товару та власний RouterView для огляду й відгуків. Спочатку створіть мінімальні представлення з заголовками, щоб перевірити маршрути до появи HTTP-логіки.

3. Конфігурація та основна навігація

У router/index.js створіть маршрутизатор через createRouter і createWebHistory(import.meta.env.BASE_URL). Підключіть його через app.use(router) до app.mount(). Визначте маршрути:

Шлях	**Ім’я кінцевого маршруту**	**Результат**
/	redirect на products	Відкриття каталогу
/products	products	Каталог
/products/:id	product-overview	Оболонка й огляд товару
/products/:id/reviews	product-reviews	Оболонка й відгуки
/about	about	Інформація про застосунок
/:pathMatch(.*)*	not-found	Невідома сторінка

Запис /products пов’язаний із ProductsLayout. Його children містить порожній шлях каталогу та :id для ProductLayout. У children товару порожній шлях відкриває огляд, reviews — відгуки. Дочірні шляхи задайте без початкової косої риски. Імена надайте кінцевим представленням, а не лише оболонкам.

Основну навігацію реалізуйте RouterLink. Назву товару в ProductCard перетворіть на посилання з name: 'product-overview', params.id і поточним query. На цьому етапі перехід має змінити URL і показати заголовок нового

представлення без завантаження нового HTML-документа. Перевірте вкладені адреси без даних, /about, redirect і невідомий шлях.

4. Завантаження товару за id

У ProductLayout отримайте маршрут через useRoute(). Побудуйте computed-джерело URL API за route.params.id. Воно має змінюватися при переході до іншого товару, але лишатися тим самим при перемиканні розділу:

```
https://dummyjson.com/products/12?delay=800
```

Підключіть useFetch із роботи 5. Уточніть його контракт для сторінки товару. На початку нового запуску очищуйте старі data та error. Перевіряйте response.ok, зберігайте status помилки й застосовуйте відповідь лише для поточного, нескасованого запуску. Наявність abort не замінює перевірку актуальності перед записом результату.

Дозволений id є рядком додатного цілого числа. Якщо він некоректний, показуйте «Товар не знайдено» без запиту; для цього useFetch має приймати null як відсутність URL. HTTP 404 також показує відсутній товар. Інші помилки показують повідомлення й кнопку повтору через refetch.

Для успіху відтворіть у спільній оболонці зображення, назву, ціну, рейтинг і наявність товару. Форматування ціни виконуйте через Intl.NumberFormat. Значення 12 використовується як адреса для перевірки, а не як жорстко задане джерело даних у компоненті.

У панелі Network переконайтеся, що пряме відкриття /products/12 створює один запит товару. Під час очікування має бути видно стан завантаження, після відповіді — дані, а за /products/999999 — повідомлення про відсутній товар.

5. Огляд і відгуки у спільній оболонці

ProductOverviewView отримує product як обов’язковий пропс і показує опис та характеристики. ProductReviewsView отримує той самий пропс і показує автора, оцінку, текст і дату кожного відгуку. Якщо reviews порожній, відтворіть окреме повідомлення.

Передайте product через слот RouterView у ProductLayout. Обидва представлення мають однаковий контракт і не виконують запитів самостійно. Створіть RouterLink для product-overview і product-reviews з поточними params.id та query. Активне посилання виділіть за router-link-exact-active. Ці переходи є навігацією між сторінками, тому використовуйте nav із посиланнями, а не ролі tablist і tab з лабораторної роботи 4.

Очистіть Network, перемкніть огляд і відгуки кілька разів та перевірте відсутність нових запитів товару. Спільні зображення, заголовок і ціна мають залишатися на місці; змінюється лише дочірній вміст.

6. Метадані та хлібні крихти

Створіть навігаційний ланцюжок, який показує місце поточної сторінки. Наприклад, для `/products/12/reviews` він має вигляд «Каталог / Назва товару

/ Відгуки». «Каталог» і назва товару є посиланнями, «Відгуки» позначає поточну сторінку.

6.1. Опишіть рівні в конфігурації маршрутів

У кожному з наведених записів додайте об'єкт `meta.breadcrumb`. Це власне поле застосунку, а не готовий механізм крихт Vue Router.

Запис маршруту	**Значення `meta.breadcrumb`**
`/products` із ProductsLayout	`{ label: 'Каталог', name: 'products' }`
Дочірній `:id` із ProductLayout	`{ dynamic: 'product', name: 'product-overview' }`
Дочірній `reviews`	`{ label: 'Відгуки', name: 'product-reviews' }`
`/about`	`{ label: 'Про застосунок', name: 'about' }`
`/:pathMatch(.*)*`	`{ label: 'Сторінку не знайдено', name: 'not-found' }`

`label` задає готовий підпис, `name` — ім'я маршруту для посилання. Позначка `dynamic: 'product'` означає, що підпис треба взяти з даних товару. У порожніх дочірніх маршрутах каталогу та огляду це поле не додавайте, інакше відповідний рівень повториться двічі.

6.2. Сформуйте масив крихт

Створіть композиційну функцію `useBreadcrumbs(productTitle)`. Аргумент із назвою товару зробіть необов'язковим, щоб викликати функцію також на сторінках без товару. Через `useRoute()` отримайте поточний маршрут і всередині `computed()` переберіть `route.matched` від предка до нащадка. Для кожного запису з `record.meta.breadcrumb` створіть елемент `{ label, to }`; записи без цього поля пропустіть.

У `to` вкажіть `name` із метаданих. Для огляду та відгуків додайте `params: { id: route.params.id }`. Якщо `route.query.q` є рядком, збережіть його в `query`. Наприклад, посилання рівня товару має вести до огляду того самого товару зі збереженим пошуком. Використовуйте метадані кожного запису, бо об'єднане `route.meta` не зберігає окремий опис усіх рівнів.

ProductLayout передає назву як `ref`, `computed` або гетер. Зчитуйте аргумент через `toValue()` всередині `computed()`. До завантаження показуйте «Товар №12» із поточним id, після відповіді — `product.title`, для відсутнього товару — «Товар не знайдено». Додатковий запит заради назви не потрібний; конфігурацію маршрутів під час завантаження не змінюйте.

6.3. Виведіть і перевірте крихти

Створіть Breadcrumbs із пропсом `items`. Він лише відображає отриманий масив. Використайте `nav` із доступною назвою, `ol` і `li`. Усі елементи, крім останнього, виведіть як RouterLink; останній — як текст із `aria-current="page"`. Роздільники приховайте через `aria-hidden="true"`.

Додайте компонент до ProductCatalogView, ProductLayout, AboutView і NotFoundView. Перевірте результат на кожній сторінці:

- каталог — «Каталог»;
- огляд — «Каталог / Назва товару»;
- відгуки — «Каталог / Назва товару / Відгуки»;
- `/about` — «Про застосунок»;
- невідома адреса — «Сторінку не знайдено».

На сторінці відгуків натискання назви відкриває огляд, а натискання «Каталог» — список зі збереженим `q`. Це має працювати і після прямого відкриття адреси відгуків у новій вкладці.

7. Узгодження пошуку та адреси

Зробіть так, щоб посилання `/products?q=phone` відновлювало пошук phone. Текст, який користувач зараз вводить, зберігайте в локальній чернетці. Значення `q` в URL є підтвердженою умовою пошуку, за якою формується запит до API. Під час введення ці значення можуть тимчасово відрізнятися.

7.1. Відновіть пошук із URL

Прочитайте `route.query.q`. Якщо це рядок, використайте його; інакше вважайте пошук порожнім. Початкове значення поля має збігатися з цим рядком. URL запиту до API обчислюйте за підтвердженим `q` і поточним `skip`, а не безпосередньо за кожним натисканням клавіші.

Відкрийте `/products?q=phone` у новій вкладці. Поле має містити phone, а перший запит каталогу — шукати phone. Повторіть перевірку після reload.

7.2. Запишіть введений пошук у URL

Під час кожної зміни поля негайно скасуйте поточний запит, як у роботі 5. Після паузи 400 мс приберіть крайові пробіли й порівняйте введений рядок із підтвердженим `q`.

Якщо рядок змінився, оновіть адресу через `router.replace({ name: 'products', query: ... })`. Для порожнього рядка вилучіть `q`. Після зміни підтвердженого пошуку скиньте `skip` до 0, очистіть `products` і `total` та завантажте першу сторінку нового пошуку. Організуйте це так, щоб зміна пошуку запускала один запит першої сторінки.

Наприклад, після введення watch і паузи адреса стає `/products?q=watch`, а каталог показує результати watch. `replace()` змінює поточний запис історії, тому кнопка «Назад» не перебирає кожен введений пошуковий рядок.

7.3. Обробіть повернення до того самого пошуку

Після негайного скасування запиту користувач може повернути попередній текст. URL тоді не зміниться, а запит уже скасований. У цьому випадку після паузи повторіть запит через `refetch()`. Самого `router.replace()` або спостерігача за `route.query.q` недостатньо.

Перевірте цей випадок із затримкою мережі:

1. Введіть phone, зачекайте 400 мс і переконайтеся в Network, що запит phone почався та ще очікує відповіді.
2. Змініть поле на watch і поверніть phone. Обидві зміни виконайте до завершення нової паузи 400 мс.

3. Після паузи має початися новий запит phone, а каталог має показати його результат. Адреса залишається `/products?q=phone`.

Debounce має повідомляти і про завершення такого редагування, хоча кінцевий рядок збігається з попереднім. У прикладі лекції для цього використано `useDebouncedValue(() => ({ query: query.value }), 400)`. Новий об'єкт позначає нове редагування. Можна використати власне рішення з такою самою поведінкою.

7.4. Відновіть стан після навігації

Якщо маршрут змінено посиланням або кнопками «Назад» і «Вперед», відновіть поле з його `q` та завантажте відповідний каталог. Це відновлення не повинно запускати відкладений запис старої чернетки назад у URL. Відрізняйте його від нового введення користувача.

Відкрийте `/products?q=phone`, перейдіть до товару й поверніться через крихту «Каталог». Повторіть повернення кнопками історії браузера. В обох випадках поле й результати мають відповідати `q` з адреси. Достатньо заново завантажити першу сторінку; зберігати накопичені сторінки та позицію прокручування не потрібно.

8. Перевірка поведінки та адаптивності

Підготуйте таблицю з колонками «Сценарій», «Очікуваний результат», «Фактичний результат», «URL» і «Кількість запитів товару». Виконайте перевірки:

1. / перенаправляє на /products; каталог зберігає поведінку роботи 5.
2. Картка відкриває товар без нового HTML-документа, q зберігається.
3. Прямі адреси огляду й відгуків працюють у новій вкладці.
4. Огляд і відгуки перемикаються без додаткового запиту товару.
5. Крихти ведуть до огляду та каталогу; Tab і Enter дозволяють пройти їх без миші.
6. Під час завантаження показано поточний id, після відповіді — його назву.
7. Зміна id оновлює дані в тому самому екземплярі ProductLayout; старий запит не перезаписує новий результат.
8. Некоректний id не створює запиту, відсутній товар показує свій стан, невідомий шлях показує сторінку 404; /about і 404 мають власну крихту поточної сторінки.
9. Offline показує помилку; після відновлення мережі повтор запиту показує товар.
10. Пошук із URL, зміна поля, повернення до каталогу та історія браузера узгоджені; повернення до phone після початого запиту q=phone не залишає скасований запит без повтору.
11. За 360×800, 768×1024, 1440×900 і 844×390 сторінка та крихти не мають горизонтального переповнення; довга назва переноситься.

Для перевірки зміни id без демонтажу тимчасово додайте в оболонку кнопку з router.push до іншого числового id, зберігши поточний розділ і query.

Увімкніть затримку мережі й виконайте два переходи поспіль. Після перевірки приберіть діагностичну кнопку. Перевірку порожніх відгуків виконайте окремо, передавши ProductReviewsView копію товару з reviews: []. Після цього відновіть звичайне передавання отриманого товару.

Збережіть знімки, перелічені в deliverables. У Network розрізняйте HTTP-запит JSON товару й завантаження коду представлення. Для перевірки кількості запитів відфільтруйте URL DummyJSON поточного id.

9. Виробнича збірка та завершення

Зупиніть dev-сервер через Ctrl+C, виконайте збірку й відкрийте її через preview:

```
npm run build
npm exec vite preview -- --port 5176 --strictPort
```

Повторіть пряме відкриття й reload для /products/12 та /products/12/reviews за адресою http://localhost:5176. Перевірте готове посилання з q. Vite preview перевіряє локальну збірку; вимоги виробничого сервера з createWebHistory поясніть у README.

Зупиніть preview через Ctrl+C. Оновіть README деревом маршрутів, таблицею відповідальностей та правилами метаданих крихт. Перевірте, що node_modules і dist не потрапляють до Git. Збережіть роботу у своєму репозиторії з каталогу його кореня:

```
git add lab6
git commit -m "Лабораторна 6: Vue Router і крихти"
git push
```

Що здається

Здаються артефакти з поля deliverables у lab.yaml. Застосовуються загальні правила звіту й репозиторію курсу. Каталог роботи — lab6.

Вимоги до змісту та оформлення звіту

- Файл звіту завантажується до СЕЗН Moodle у форматі PDF.
- Звіт містить титульний аркуш і деталі виконання роботи.
- Звіт містить посилання на репозиторій GitHub, Bitbucket або GitLab із роботою в каталозі `lab1`, `lab2`, … за номером роботи.
- Звіт містить команди, які виконуються під час роботи.
- Звіт містить знімки екрана, з яких зрозумілий контекст роботи та видно персональні дані автора: адресний рядок, дані студента й відомості про середовище.

Крім переліченого, за цією роботою здається:

- Вихідний код lab6 із конфігурацією маршрутів, оболонками, представленнями товару, Breadcrumbs і useBreadcrumbs та package-lock.json
- README з деревом маршрутів, відповідальністю модулів і таблицею метаданих крихт; пояснення розподілу стану між query, params і завантаженим товаром
- Таблиця сценаріїв з очікуваним і фактичним результатом, URL та кількістю запитів товару під час переходів між його розділами
- Знімки каталогу з q, огляду й відгуків із крихтами, вузької сторінки, відсутнього товару та сторінки 404; знімок Network із запитом товару й результатом перемикання розділів
- Результат npm run build і перевірки прямих адрес через vite preview

Контрольні питання

1. Яку роль виконують RouterLink і RouterView у маршрутизованому застосунку?
2. Чим відрізняються params.id і query.q в адресі /products/12/reviews?q=phone?
3. Чому запиту товару лише в onMounted недостатньо для переходів між різними id?
4. Чому хлібні крихти будуються за route.matched, а не за route.meta?
5. Для чого в конфігурації товару потрібен порожній дочірній шлях?
6. Чому перехід між оглядом і відгуками не повинен створювати новий запит товару?

Література до роботи

1. Vue Router. Getting Started — https://router.vuejs.org/guide/
2. Vue Router. Named Routes — https://router.vuejs.org/guide/essentials/named-routes.html
3. Vue Router. Dynamic Route Matching — https://router.vuejs.org/guide/essentials/dynamic-matching.html
4. Vue Router. Nested Routes — https://router.vuejs.org/guide/essentials/nested-routes.html
5. Vue Router. Programmatic Navigation — https://router.vuejs.org/guide/essentials/navigation.html
6. Vue Router. Route Meta Fields — https://router.vuejs.org/guide/advanced/meta.html
7. Vue Router. Navigation Guards — https://router.vuejs.org/guide/advanced/navigation-guards.html

8. Vue Router. Data Fetching — https://router.vuejs.org/guide/advanced/data-fetching.html
9. Vue Router. Composition API — https://router.vuejs.org/guide/advanced/composition-api.html
10. Vue Router. RouterView slot — https://router.vuejs.org/guide/advanced/router-view-slot.html
11. Vue Router. History modes — https://router.vuejs.org/guide/essentials/history-mode.html
12. MDN. Working with the History API — https://developer.mozilla.org/en-US/docs/Web/API/History_API/Working_with_the_History_API
13. WAI-ARIA APG. Breadcrumb Pattern — https://www.w3.org/WAI/ARIA/apg/patterns/breadcrumb/
14. DummyJSON. Products — https://dummyjson.com/docs/products
15. Vue Router. Active links — https://router.vuejs.org/guide/essentials/active-links.html
16. Vue Router. Passing Props to Route Components — https://router.vuejs.org/guide/essentials/passing-props.html
17. Vue Router. Redirect and Alias — https://router.vuejs.org/guide/essentials/redirect-and-alias.html
18. Vue Router. Navigation Failures — https://router.vuejs.org/guide/advanced/navigation-failures.html
19. Vue. Watchers — https://vuejs.org/guide/essentials/watchers.html
20. MDN. Using the Fetch API — https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch