hillel IT school

Lesson 35

План заняття

- React Routing

React Router

React Router дозволяє «маршрутизацію на стороні клієнта».

На традиційних веб-сайтах веб-переглядач запитує документ із веб-сервера, завантажує та оцінює ресурси CSS і JavaScript і відображає HTML, надісланий із сервера. Коли користувач натискає посилання, процес починається заново для нової сторінки.

Маршрутизація на стороні клієнта дозволяє вашій програмі оновлювати URL-адресу після натискання посилання без повторного запиту на інший документ із сервера. Замість цього ваша програма може негайно відтворити новий інтерфейс і зробити запити даних за допомогою fetch, щоб оновити сторінку новою інформацією.

Це забезпечує швидшу взаємодію з користувачем, оскільки браузеру не потрібно запитувати повністю новий документ або повторно оцінювати ресурси CSS і JavaScript для наступної сторінки. Це також забезпечує більш динамічний досвід користувача за допомогою таких речей, як анімація.

З базовими концепціями ви можете ознайомитись тут https://reactrouter.com/en/main/start/concepts

Routers

Хоча ваша програма використовуватиме лише один маршрутизатор, доступно кілька маршрутизаторів залежно від середовища, у якому працює ваша програма.

- createBrowserRouter
- createMemoryRouter
- createHashRouter
- createStaticRouter

Для будь яких web проектів використовуйте createBrowserRouter.

createBrowserRouter

Він використовує повну URL-адресу замість хеш-адрес (#this/stuff), поширених у веб-додатках до стандартизації history.pushState. Повні URL-адреси кращі для пошукової оптимізації, кращі для відтворення на сервері та просто більш сумісні з рештою веб-платформи.

Якщо ви розміщуєте свою програму на статичному файловому сервері, вам потрібно буде налаштувати її так, щоб усі запити надсилалися на ваш index.html, щоб уникнути отримання помилок 404.

Якщо з якоїсь причини ви не можете використати повну URL-адресу, createHashRouter — наступна найкраща річ.

Якщо вас не цікавлять API даних, ви можете продовжувати використовувати <BrowserRouter> або, якщо ви не можете використовувати повні URL-адреси, <HashRouter>.

Повертає об’єкт router для RouterProvider.

Детальніше тут https://reactrouter.com/en/main/routers/create-browser-router

createBrowserRouter

Синтаксис:

```
createBrowserRouter(
  routes: RouteObject[],
  opts?: {
    basename?: string;
    future?: FutureConfig;
    hydrationData?: HydrationState;
    window?: Window;
  }
)
```

createBrowserRouter

routes - Масив об’єктів Route із вкладеними маршрутами у дочірній власності.

basename - Базова назва програми для ситуацій, коли ви не можете розгорнути в кореневій частині домену, а в підкаталозі.

future - Додатковий набір майбутніх прапорів для ввімкнення для цього маршрутизатора. Ми рекомендуємо якомога швидше ввімкнути нещодавно випущені майбутні прапорці, щоб полегшити ваш можливий перехід на v7.

hydrationData - Під час серверного рендерингу та відмови від автоматичної гідратації параметр hydrationData дозволяє передавати дані гідратації з вашого серверного рендерингу. Це майже завжди буде підмножина даних із значення StaticHandlerContext, яке ви отримуєте з handler.query

createHashRouter

Цей маршрутизатор стане в нагоді, якщо ви не можете налаштувати веб-сервер, щоб спрямувати весь трафік до програми React Router. Замість використання звичайних URL-адрес для керування "URL-адресою програми" використовуватиметься хеш (#) URL-адреси.

Не рекомендується використовувати хеш-адреси.

createMemoryRouter

Замість того, щоб використовувати історію браузера, маршрутизатор пам’яті керує власним стеком історії в пам’яті. Це в першу чергу корисно для інструментів тестування та розробки компонентів, таких як Storybook, але також може використовуватися для запуску React Router у будь-якому небраузерному середовищі.

Детальніше тут https://reactrouter.com/en/main/routers/create-memory-router

createStaticRouter

createStaticRouter використовується, коли ви хочете використовувати маршрутизатор даних для візуалізації на вашому сервері (тобто Node або інше середовище виконання Javascript).

Докладніше тут https://reactrouter.com/en/main/routers/create-static-router

RouterProvider

Усі об’єкти маршрутизатора даних передаються цьому компоненту для відтворення вашої програми та ввімкнення решти даних API.

Синтааксис:

```
<RouterProvider
    router={router}
    fallbackElement={<Example  />}
    future={{ v7_startTransition:  true }}
/>
```

RouterProvider

FallbackElement - Якщо ви не використовуєте SSR, який рендерить вашу програму, createBrowserRouter ініціює всі відповідні завантажувачі маршрутів під час монтування. Протягом цього часу ви можете надати запасний елемент, щоб надати користувачеві деяку інформацію про те, що програма працює. Зробіть цей статичний хостинг TTFB рахунком!

Route

Маршрути є, мабуть, найважливішою частиною програми React Router. Вони поєднують сегменти URL-адрес з компонентами, завантаженням даних і мутаціями даних. Завдяки вкладеності маршрутів складні макети програм і залежності даних стають простими та декларативними.

Ви також можете оголосити свої маршрути за допомогою JSX і createRoutesFromElements, властивості елемента ідентичні властивостям об’єктів маршруту.

```
interface RouteObject {
  path?: string;
  index?: boolean;
  children?: React.ReactNode;
  caseSensitive?: boolean;
  id?: string;
  loader?: LoaderFunction;
  action?: ActionFunction;
  element?: React.ReactNode | null;
  hydrateFallbackElement?: React.ReactNode | null;
  errorElement?: React.ReactNode | null;
  Component?: React.ComponentType | null;
  HydrateFallback?: React.ComponentType | null;
  ErrorBoundary?: React.ComponentType | null;
  handle?: RouteObject["handle"];
  shouldRevalidate?: ShouldRevalidateFunction;
  lazy?: LazyRouteFunction<RouteObject>;
}
```

Route.path

Шаблон шляху для порівняння з URL-адресою, щоб визначити, чи цей маршрут відповідає URL-адресі, URL-адресі href або дії форми.

Наприклад: "/" або "home"

Якщо сегмент шляху починається з : тоді він стає "динамічним сегментом". Коли маршрут збігається з URL-адресою, динамічний сегмент буде проаналізовано з URL-адреси та надано як параметри іншим API маршрутизатора.

Наприклад: "user/:id"

Ви можете зробити сегмент маршруту необов’язковим, додавши ? до кінця відрізка.

Наприклад: "user/:id?/details" або "users?/details"

Також відомий патерн як сегменти "star". Якщо шаблон шляху маршруту закінчується на /*, тоді він відповідатиме будь-яким символам, що йдуть після /, включаючи інші символи /.

Наприклад: "*" або "categories/*"

Route.index

Визначає, чи маршрут є індексним. Індексні маршрути відображаються в батьківському Outlet за URL-адресою батьківського (як дочірній маршрут за замовчуванням).

Докладніше про інтекси тут https://reactrouter.com/en/main/start/concepts#index-routes

Route.children

Саброути які будуть вкладені до батьківського шляху. Мають такий самий інтерфейс як і Route.

Route.loader

Завантажувач маршруту викликається перед рендерингом маршруту та надає дані для елемента через useLoaderData. Якщо ви не використовуєте маршрутизатор даних, наприклад createBrowserRouter, це нічого не дасть.

Докладніше тут https://reactrouter.com/en/main/route/loader

Route.loader

Завантажувач маршруту викликається перед рендерингом маршруту та надає дані для елемента через useLoaderData. Якщо ви не використовуєте маршрутизатор даних, наприклад createBrowserRouter, це нічого не дасть.

Докладніше тут https://reactrouter.com/en/main/route/loader

Route.action

Дія маршруту викликається, коли відправлення надсилається до маршруту з форми, збирача або подання. Якщо ви не використовуєте маршрутизатор даних, наприклад createBrowserRouter, це нічого не дасть.

Докладніше тут https://reactrouter.com/en/main/route/action

Route.element || Component

Елемент/компонент React для візуалізації, коли маршрут збігається з URL-адресою.

Якщо ви хочете створити елемент React, використовуйте element.

Наприклад: <Route path="/" element={<HomePage/>} />

В іншому випадку використовуйте Component, і React Router створить React Element для вас

Наприклад: <Route path="/" Component={Component} />

Ви повинні використовувати API компонентів лише для маршрутизації даних через RouterProvider. Використання цього API для <Route> всередині <Routes> призведе до деоптимізації здатності React повторно використовувати створений елемент у різних рендерах.

Route.errorElement

Коли маршрут створює виняткову ситуацію під час відтворення, у завантажувачі або в дії, цей елемент/компонент React відображатиметься замість звичайного елемента/компонента.

Якщо ви хочете створити елемент React самостійно, використовуйте errorElement. В іншому випадку використовуйте ErrorBoundary, і React Router створить React Element для вас.

Докладніше тут https://reactrouter.com/en/main/route/error-element

Route.lazy

Щоб ваші пакети програм були невеликими та підтримували розподілення коду ваших маршрутів, кожен маршрут може надавати асинхронну функцію, яка вирішує невідповідні частини визначення маршруту (завантажувач, дія, компонент/елемент, ErrorBoundary/errorElement, тощо).

Кожна відкладена функція зазвичай повертає результат динамічного імпорту.

Докладніше тут https://reactrouter.com/en/main/route/lazy

<Link/>

<Link> — це елемент, який дозволяє користувачеві переходити на іншу сторінку. У react-router-dom <Link> рендерить доступний елемент <a> зі справжнім href, який вказує на ресурс, на який він посилається. Це означає, що такі речі, як клацання правою кнопкою миші <Link>, працюють так, як ви очікували. Ви можете використати <Link reloadDocument>, щоб пропустити маршрутизацію на стороні клієнта та дозволити веб-переглядачу нормально виконувати перехід (як якщо б це був <a href>).

Відносне значення <Link to> (яке не починається з /) розв'язується відносно батьківського маршруту, що означає, що воно будується на шляху URL-адреси, який відповідає маршруту, який відобразив це <Link>. Він може містити .. для посилань на маршрути, що знаходяться далі в ієрархії. У цих випадках .. працює так само, як функція командного рядка cd; кожний .. видаляє один сегмент батьківського шляху.

Синтаксис:

<Link to=".." relative="path">Go to</Link>

<Link to="/">Go to</Link>

<Link/> attributes

- relative - За замовчуванням посилання є відносними до ієрархії маршрутів (relative="route"), тому .. підніметься на один рівень Route вище від поточного контекстного маршруту. Іноді ви можете виявити, що у вас є відповідні шаблони URL-адрес, які не має сенсу бути вкладеними, і ви віддаєте перевагу використанню маршрутизації відносного шляху з поточного шляху контекстного маршруту. Ви можете вибрати таку поведінку за допомогою relative="path"
- preventScrollReset - Якщо ви використовуєте <ScrollRestoration>, це дає змогу запобігти скиданню позиції прокручування до верхньої частини вікна під час натискання посилання.
- replace - Властивість replace можна використовувати, якщо ви хочете замінити поточний запис у стеку історії через history.replaceState замість використання за замовчуванням history.pushState.
- state - Властивість стану можна використовувати для встановлення значення стану для нового розташування, яке зберігається в стані історії. Згодом це значення можна отримати за допомогою useLocation().
- reloadDocument - Властивість reloadDocument можна використовувати, щоб пропустити маршрутизацію на стороні клієнта та дозволити браузеру нормально обробляти перехід (як якщо б це був <a href>).

<NavLink/>

<NavLink> — це особливий тип <Link>, який знає, чи є воно «active», «panding» чи «transitioning». Це корисно в кількох різних сценаріях:

- Під час створення навігаційного меню, наприклад навігаційної ланцюжка або набору вкладок, де ви хочете показати, яка з них наразі вибрана
- Надає корисний контекст для допоміжних технологій, таких як програми зчитування з екрана
- Надає значення «transitioning», щоб надати вам більш детальний контроль над переходами перегляду. Доклаадніше про transition тут https://developer.mozilla.org/en-US/docs/Web/API/View_Transitions_API

Синтаксис:

```
<NavLink
  to="/messages"
  className={({ isActive, isPending, isTransitioning }) =>
    isPending ? "pending" : isActive ? "active" : ""
  }
>
  Messages
</NavLink>;
```

<NavLink/> attributes

За замовчуванням активний клас додається до компонента <NavLink>, коли він активний, тому ви можете використовувати CSS для його стилізації.

- className - працює як звичайний className, але ви також можете передати йому функцію, щоб налаштувати застосовані класи на основі активного та незавершеного стану посилання.
- style - працює як звичайний style, але ви також можете передати їй функцію, щоб налаштувати застосовані стилі на основі активного та незавершеного стану посилання.
- end - End prop змінює логіку відповідності для активного та очікуваного станів, щоб відповідати лише "кінцю" шляху NavLink to. Якщо URL-адреса довша ніж to, вона більше не вважатиметься активною.

Наприклад:

<NavLink to="/tasks" end /> => /tasks/123 => false

<NavLink to="/tasks" /> => /tasks/123 => true

- caseSensitive - Додавання властивості caseSensitive змінює логіку відповідності, щоб зробити її чутливою до регістру.
- aria-current - Коли NavLink активний, він автоматично застосовуватиме <a aria-current="page"> до основного тегу прив’язки. Докладніше тут https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Attributes/aria-current

<NavLink/> attributes

- reloadDocument - Властивість reloadDocument можна використовувати, щоб пропустити маршрутизацію на стороні клієнта та дозволити браузеру нормально обробляти перехід (як якщо б це був <a href>).

<Outlet/>

<Outlet> слід використовувати в батьківських елементах маршруту для відтворення їхніх дочірніх елементів маршруту. Це дозволяє вкладеному інтерфейсу користувача відображатися під час візуалізації дочірніх маршрутів. Якщо батьківський маршрут точно збігається, він відтворить дочірній індексний маршрут або нічого, якщо індексного маршруту немає.

useLocation()

Цей хук повертає поточний об’єкт розташування. Це може бути корисним, якщо ви бажаєте виконувати якийсь побічний ефект щоразу, коли змінюється поточне розташування. Повертає об’єкт location, має такі властивості:

- hash - Хеш поточної URL-адреси.
- key - Унікальний ключ від цієї локації.
- pathname - Шлях поточної URL-адреси.
- search - Рядок запиту поточної URL-адреси.
- state - Значення стану розташування, створене <Link state> або навігацією.

useNavigate()

Хук useNavigate повертає функцію, яка дозволяє здійснювати програмну навігацію, наприклад, у ефекті.

Функція навігації має двы сыгнатури:

- Або передайте значення To (того самого типу, що й <Link to>) із необов’язковим другим аргументом параметрів (подібно до атрибутів, які ви можете передати в <Link>), або
- Передайте дельту, яку ви хочете перенести в стек історії. Наприклад, navigate(-1) еквівалентно натисканню кнопки «Go back».

За документацією краще використовувати redirect в loader і action, ніж цей хук.

useParams()

Хук useParams повертає об’єкт із парами ключ/значення динамічних параметрів із поточної URL-адреси, які відповідають <Route path>. Дочірні маршрути успадковують усі параметри своїх батьківських маршрутів.

useSearchParams()

Хук useSearchParams використовується для читання та зміни рядка запиту в URL-адресі поточного розташування. Як і власний хук React useState, useSearchParams повертає масив із двох значень: параметри пошуку поточного розташування та функцію, яка може бути використана для їх оновлення. Як і хук useState React, setSearchParams також підтримує функціональні оновлення. Тому ви можете надати функцію, яка приймає searchParams і повертає оновлену версію.

Синтаксис:

```
const [searchParams, setSearchParams] = useSearchParams();
```

useLoadData()

Цей хук надає значення, яке повертає loader маршрутів.

Після виклику дій маршруту дані будуть повторно перевірені автоматично та повернуть останній результат із вашого завантажувача.

Зверніть увагу, що useLoaderData не ініціює вибірку. Він просто зчитує результат вибірки, якою React Router керує внутрішньо, тож вам не потрібно турбуватися про його повторне отримання під час повторного рендерингу з причин, які не пов’язані з маршрутизацією.

Це також означає, що повернуті дані стабільні між візуалізаціями, тож ви можете безпечно передавати їх у масиви залежностей у хуках React, як-от useEffect. Він змінюється лише тоді, коли завантажувач викликається знову після дій або певних навігацій. У цих випадках ідентифікатор буде змінено (навіть якщо значення не зміняться).

Ви можете використовувати цей хук у будь-якому компоненті або будь-якому спеціальному хуку, а не лише в елементі Route. Він поверне дані з найближчого маршруту в контексті.

Щоб отримати дані з будь-якого активного маршруту на сторінці, перегляньте useRouteLoaderData.

useRouteLoadData()

Цей хук робить дані на будь-якому відтвореному маршруті доступними будь-де в дереві. Це корисно для компонентів глибоко в дереві, яким потрібні дані з маршрутів, розташованих набагато вище, а також для батьківських маршрутів, яким потрібні дані дочірніх маршрутів, розташованих глибше в дереві.

React Router зберігає дані внутрішньо з детермінованими автоматично згенерованими ідентифікаторами маршрутів, але ви можете надати власний ідентифікатор маршруту, щоб полегшити роботу з цим хуком. Єдиними доступними даними є маршрути, які зараз відображаються. Якщо ви запитуєте дані з маршруту, який наразі не відображається, хук поверне undefined.

Дякую за увагу