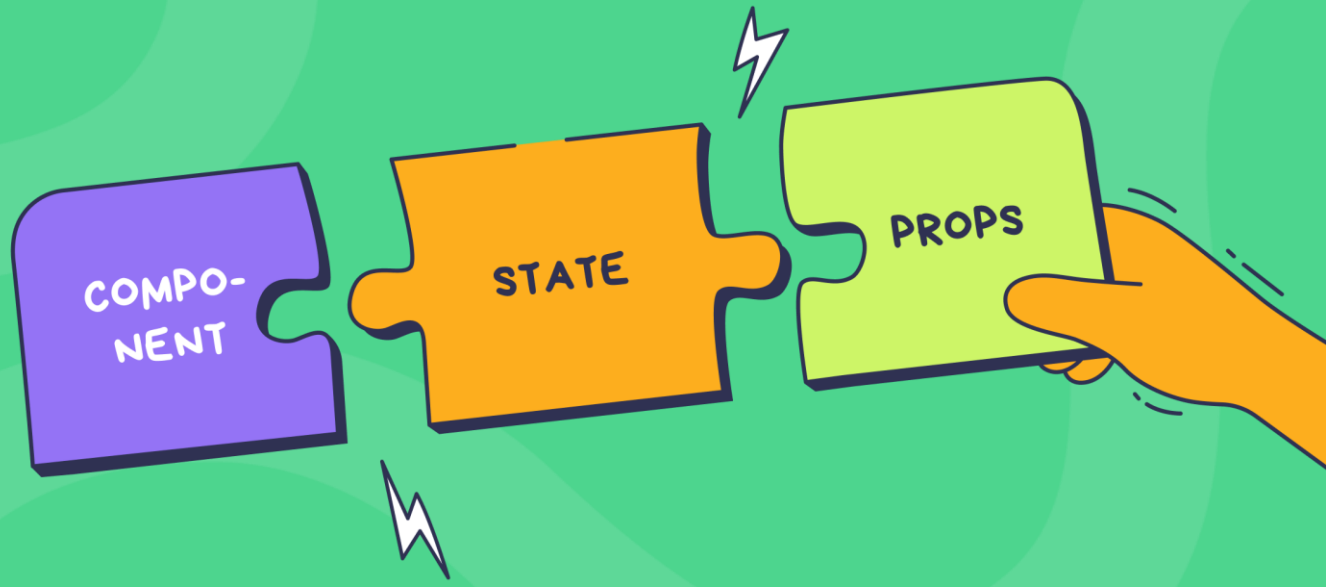
JAVARUSH

Путь к карьере Frontend Fullstack разработчика

MODULE 3. React
Уровень 9 Маршрутизация с react-router-dom и динамические маршруты



JAVARUSH

Что такое маршрутизация в SPA?

Single-Page Applications (SPA), создаваемые с помощью React, загружают одну HTML-страницу, а контент меняется динамически с помощью JavaScript, создавая иллюзию перехода между "страницами".

Проблема. Как обеспечить уникальные URL для разных "экранов" или "секций" SPA и позволить пользователю использовать кнопки "назад/вперед" браузера, а также делиться ссылками?

Решение. Клиентская маршрутизация (Client-Side Routing).
Это техника, при которой JavaScript-код в браузере отслеживает изменения URL (или его части) и рендерит соответствующие React-компоненты без полной перезагрузки страницы с сервера.

JAVARUSH

Основные компоненты react-router-dom

react-router-dom - это самая популярная и де-факто стандартная библиотека для реализации клиентской маршрутизации в React-приложениях.

React-router-dom v6 использует следующие ключевые компоненты:

<BrowserRouter> (или другой тип роутера, например, HashRouter):
- Компонент-обертка верхнего уровня. Использует HTML5 History API для синхронизации UI с URL.
- Оборачивает ту часть вашего приложения, где нужна маршрутизация (обычно все приложение).

<Routes>:
- Контейнер для одного или нескольких <Route>.
- "Смотрит" на текущий URL и рендерит первый дочерний <Route>, чей path совпадает. (Аналог Switch из v5).

<Route path="путь" element={<Компонент />}>:
- Определяет соответствие между URL-путем (path) и React-компонентом (element), который должен быть отрендерен.
- Может быть вложенным для создания иерархических маршрутов.

<Link to="путь">Text</Link> / <NavLink to="путь">Text</NavLink>:
- Компоненты для создания навигационных ссылок, которые изменяют URL без перезагрузки страницы.
- <NavLink> умеет стилизовать активную ссылку.

Хуки: useNavigate(), useParams(), useLocation(), useOutletContext() и др. для программной навигации и доступа к информации о маршруте из компонентов.

JAVARUSH

Установка

```
# Установка
npm install react-router-dom
```

Обернуть приложение в <BrowserRouter>. Это делается в корневом файле вашего приложения (обычно src/main.tsx или src/index.tsx).

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { BrowserRouter } from 'react-router-dom';
ReactDOM.createRoot(document.getElementById('root')!).render(
   <React.StrictMode>
       {/* 2. Оборачиваем наш корневой компонент App */}
       <BrowserRouter>
           <App />
       </BrowserRouter>
   </React.StrictMode>
);
```

JAVARUSH

Создание базовых маршрутов

Вы указываете, какой компонент должен отображаться для какого URL. Это делается с помощью компонентов **<Routes>** и **<Route>**.

Логика работы:

1. **<Routes> (Контейнер):** оборачивает все ваши определения <Route>. Его задача – найти первый дочерний <Route>, чей path совпадает с текущим URL в браузере, и отрендерить его element.
2. **<Route path="url-путь" element={<КомпонентСтраницы />} /> (Определение маршрута):**
   a. **path:** строка, определяющая URL-путь (например, /, /about, /products/:id).
   b. **element:** JSX-элемент, который будет отрендерен, если path совпадет (обычно это компонент-страница).

```
import { Routes, Route } from 'react-router-dom';
import HomePage from './pages/HomePage';
import AboutPage from './pages/AboutPage';

// Можно вынести маршруты в отдельный компонент
function AppRoutes() {
    return (
        <Routes>
            <Route path="/" element={<HomePage />} />
            <Route path="/about" element={<AboutPage />} />
            {/* Другие маршруты... */}
            {/* <Route path="*" element={<NotFoundPage />} />
// Для 404 */}
        </Routes>
    );
}
```

JAVARUSH

Навигация

Для создания навигационных ссылок, которые изменяют URL без перезагрузки страницы, react-router-dom предоставляет компоненты **<Link>** и **<NavLink>**.

<Link to="/your-path">Текст ссылки</Link>:

- Основной компонент для навигации.
- Принимает проп to со строкой пути (или объектом пути).
- При клике обновляет URL в браузере через HTML5 History API. react-router-dom отслеживает это изменение и рендерит соответствующий <Route>.

<NavLink to="/your-path" className={({isActive}) => isActive ? 'active-link' : ''}>...</NavLink>:

- Специальная версия <Link>, которая "знает", является ли ссылка активной (соответствует текущему URL браузера).
- Позволяет легко стилизовать активную ссылку:
  - Проп className: может быть функцией, получающей { isActive, isPending }.
  - Проп style: также может быть функцией.
  - Атрибут end: если true, ссылка активна только при точном совпадении пути (полезно для ссылок типа /).

JAVARUSH

Динамические маршруты и параметры

Часто URL должен содержать динамическую часть, например, ID продукта или имя пользователя.

1. Определение динамического сегмента в <Route>. Используйте двоеточие : перед именем параметра в атрибуте **path**.

```
<Route path="/products/:productId" element={<ProductDetailPage />} />
<Route path="/users/:username/profile" element={<UserProfilePage />} />
```

Здесь **:productId** и **:username** – это имена параметров.

JAVARUSH

Хук useParams()

Получение значения параметра в компоненте: Хук useParams()

- Импорт: import { useParams } from 'react-router-dom';
- useParams() возвращает объект, где ключи – это имена параметров из path, а значения – их фактические значения из текущего URL (всегда строки).

```tsx
// ProductDetailPage.tsx
import React from 'react';
import { useParams } from 'react-router-dom';

interface ProductPageParams {
   productId: string; // Всегда строка из useParams
}

const ProductDetailPage: React.FC = () => {
   // useParams() вернет объект, например, { productId: "123" }
   const params = useParams<Readonly<ProductPageParams>>();
   // или const { productId } = useParams<Readonly<ProductPageParams>>();

   return (
       <div>
           <h2>Страница продукта</h2>
           <p>ID продукта из URL: {params.productId}</p>
           {/* Если productId - число, нужно его преобразовать: Number(params.productId) */}
       </div>
   );
};
```

JAVARUSH

Типизация параметров маршрута

Хук useParams можно (и нужно) типизировать для безопасности и удобства.

Шаги:

1. **Определите interface** или **type** для ожидаемых параметров маршрута. Помните, что useParams всегда возвращает значения параметров как **строки**.
2. **Передайте этот тип как generic в useParams<MyParamsType>().**

Автоматический вывод типов:

react-router-dom v6.4+ и **TypeScript 4.1+** могут лучше выводить типы параметров, если **path** в **<Route>** определен как строковый литерал с параметрами. Однако явное указание типа через **generic** остается надежной практикой, особенно для сложных или опциональных параметров.

JAVARUSH

Nested Routes

Вложенные маршруты позволяют создавать иерархическую структуру UI, где одна часть интерфейса (например, боковая панель или шапка раздела) остается неизменной, а основная контентная область меняется в зависимости от более глубокого (вложенного) пути URL.

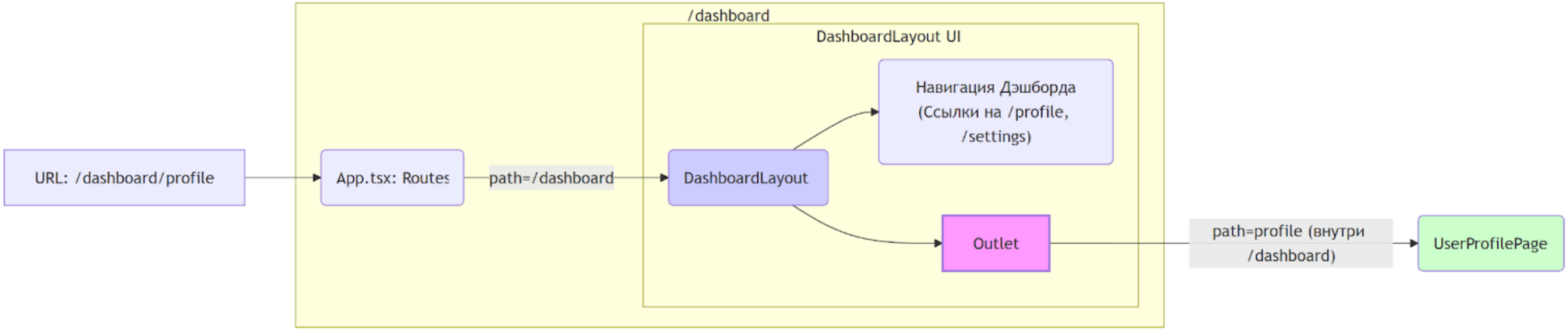
Как реализовать:

1. **Определение вложенных <Route>:** один <Route> (родительский) может содержать другие <Route> (дочерние) внутри себя.
2. **Layout-компонент:** родительский <Route> рендерит свой element – это обычно компонент-обертка или "layout" для вложенных страниц.
3. **Компонент <Outlet />:** внутри layout-компонента родителя используется специальный компонент <Outlet /> (из react-router-dom). <Outlet /> служит местом (placeholder), куда react-router-dom будет рендерить компонент, соответствующий активному дочернему маршруту.

Вложенные маршруты и Outlet

Компонент <Outlet /> – это ключ к работе вложенных маршрутов.

При переходе на **/dashboard/p**rofile, сначала рендерится **DashboardLayout**. Затем, в место, где находится **<Outlet />** внутри **DashboardLayout**, рендерится **UserProfilePage**. Навигация внутри **DashboardLayout** меняет только то, что рендерится в **<Outlet />**.



JAVARUSH

Программная навигация

Иногда переход на другую страницу нужно выполнить не по клику на **<Link>**, а программно, например, после отправки формы, выхода из системы или другого действия.

Для этого используется **хук useNavigate()**.

Импорт: **import { useNavigate } from 'react-router-dom';**

Использование:

1. Вызовите хук в вашем компоненте: **const navigate = useNavigate()**;
2. Функция **navigate** используется для выполнения перехода:
   a. **navigate('/new-path'):** дереход на указанный путь.
   b. **navigate(-1):** дереход на предыдущую страницу в истории (аналог кнопки "назад").
   c. **navigate(1):** дереход на следующую страницу (аналог "вперед").

Опции:

navigate(to, { replace?: boolean, state?: any })

- **replace: true:** заменяет текущую запись в истории браузера, а не добавляет новую (нельзя будет вернуться назад на эту страницу).
- **state:** позволяет передать некоторое состояние на целевую страницу (доступно там через useLocation().state).

JAVARUSH

Доступ к информации о маршруте

Помимо **useParams** и **useNavigate**, есть и другие полезные хуки:

useLocation():

- Возвращает объект location, представляющий текущий URL.
- **Содержит:**
  - **pathname:** текущий путь (например, /products/123).
  - **search:** строка query-параметров (например, ?sort=asc&filter=active).
  - **hash:** хеш-фрагмент URL (например, #section-details).
  - **state:** состояние, переданное через navigate(..., { state: ... }) или <Link state={...}>.
  - **key:** уникальный ключ для текущей записи в истории.
- Полезно для аналитики, условного рендеринга на основе пути, работы с query-параметрами.

useMatch(pattern: PathPattern | string):

- Проверяет, соответствует ли текущий URL заданному шаблону пути (pattern).
- Если соответствует, возвращает объект совпадения (включая params, pathname), иначе null.
- Может быть полезно для определения, активен ли определенный маршрут или его часть, если возможностей **<NavLink>** недостаточно, или для сложной логики на основе

JAVARUSH

Protected Routes

Часто в приложении есть страницы или целые разделы, доступ к которым должен быть ограничен только для аутентифицированных пользователей (или пользователей с определенными ролями/правами).

Общий подход к созданию "защищенных маршрутов":

1. **Создать компонент-обертку** (например, <ProtectedRoute /> или <AuthGuard />).
2. Этот компонент **проверяет статус аутентификации** текущего пользователя (например, используя контекст аутентификации, Redux store, или кастомный хук useAuth()).
3. **Логика рендеринга в обертке:**
    a. **Если пользователь аутентифицирован** (и, опционально, имеет нужные права): Компонент-обертка рендерит дочерний компонент (тот, который нужно защитить), переданный ему через props.children или проп element.
    b. **Если пользователь НЕ аутентифицирован.** Компонент-обертка перенаправляет пользователя на страницу входа (например, /login). Часто при этом сохраняется исходный URL, на который пользователь пытался зайти, чтобы вернуть его туда после успешного входа.

JAVARUSH

Типизация защищенных маршрутов

При типизации защищенных маршрутов основное внимание уделяется:

1. **Типизации пропсов самого компонента-обертки** (например, ProtectedRouteProps):
   - children: JSX.Element или ReactNode (если компонент передается как дочерний).
   - Опциональные пропсы, такие как redirectTo или roles для более сложной логики.

2. **Типизации хука аутентификации (useAuth)**
   Хук должен возвращать четко типизированное состояние (например, { isAuthenticated: boolean; user: UserType | null; isLoading: boolean; }).

3. **Совместимости пропсов**
   Убедиться, что компонент-обертка (ProtectedRoute) правильно передает все необходимые пропсы защищаемому компоненту, если это требуется. Часто защищаемый компонент не получает никаких дополнительных пропсов от ProtectedRoute, а получает их от самого роутера (через useParams) или из контекста/Redux.

В большинстве случаев, если ваш **хук useAuth** и компоненты страниц хорошо типизированы, типизация самого **ProtectedRoute** сводится к описанию его **children** или **element**.

JAVARUSH

Домашнее задание

Уровень 9 Маршрутизация с react-router-dom и динамические маршруты

MODULE 3. REACT