

React Router v7 Navigation Guide

Student Notes with E-Commerce Analogies

1. Understanding Navigation Components & Functions

Navigation in React Router v7 is like the different ways customers move around an e-commerce store.

2. `<Link>` Component

What it does:

`<Link>` is a basic navigation element that takes users from one route to another without reloading the page. It renders as an `<a>` tag in HTML.

E-Commerce Analogy:

Think of `<Link>` as **clickable product category labels or breadcrumb navigation** in a store. When a customer clicks on "Electronics," they navigate to that section smoothly without the entire store refreshing.

Code Example:

```
javascript

import { Link } from 'react-router-dom';

export function ProductList() {
  return (
    <div>
      <h1>Shop Products</h1>
      <Link to="/products/electronics">Electronics</Link>
      <Link to="/products/clothing">Clothing</Link>
      <Link to="/products/books">Books</Link>
    </div>
  );
}
```

Key Features:

- **Lightweight and semantic** - Uses standard HTML links
- **Prefetching** - Can preload route data before navigation

- **No state management** - Simple client-side navigation
 - **SEO friendly** - Actual links that search engines can crawl
-

3. `<NavLink>` Component

What it does:

`<NavLink>` is an enhanced version of `<Link>` that automatically applies active styling when the current route matches the link's destination.

E-Commerce Analogy:

Imagine the **main navigation menu at the top of an e-commerce site**. When you're browsing "Shoes," the "Shoes" menu item is highlighted/underlined to show you where you are. This is exactly what `<NavLink>` does!

Code Example:

```
javascript
```

```

import { NavLink } from 'react-router-dom';

export function Navigation() {
  return (
    <nav className="navbar">
      <NavLink
        to="/"
        className={({ isActive }) => isActive ? "active-link" : ""}
      >
        Home
      </NavLink>

      <NavLink
        to="/products"
        className={({ isActive }) => isActive ? "active-link" : ""}
      >
        Products
      </NavLink>

      <NavLink
        to="/cart"
        className={({ isActive }) => isActive ? "active-link" : ""}
      >
        Cart
      </NavLink>

      <NavLink
        to="/account"
        className={({ isActive }) => isActive ? "active-link" : ""}
      >
        Account
      </NavLink>
    </nav>
  );
}

```

Key Differences from Link:

- **Active state detection** - Knows when it's the current page
- **Dynamic styling** - Can apply different CSS based on active status
- **Perfect for navigation menus** - Users always know where they are
- **Callback function** - Can receive `isActive` and `isPending` states

Active Styling Example:

css

```
.active-link {  
  color: #ff6b6b;  
  font-weight: bold;  
  border-bottom: 3px solid #ff6b6b;  
}
```

4. `<Form>` Component

What it does:

`<Form>` handles form submissions and integrates with route actions. Instead of traditional form submission, it coordinates with your route handlers.

E-Commerce Analogy:

Think of `<Form>` as **checkout process forms or login pages**. When a customer fills out their shipping address and clicks "Continue," the form captures their data, validates it, and processes it through your backend route handlers (actions) before moving forward.

Code Example:

javascript

```
import { Form, useActionData } from 'react-router-dom';

export function CheckoutForm() {
  const actionData = useActionData();

  return (
    <Form method="post" action="/checkout">
      <fieldset>
        <legend>Shipping Address</legend>

        <input
          type="text"
          name="firstName"
          placeholder="First Name"
          required
        />

        <input
          type="text"
          name="lastName"
          placeholder="Last Name"
          required
        />

        <input
          type="email"
          name="email"
          placeholder="Email Address"
          required
        />

        <textarea
          name="address"
          placeholder="Street Address"
          required
        ></textarea>

        <button type="submit">Complete Purchase</button>
      </fieldset>

      {actionData?.error && (
        <div className="error-message">{actionData.error}</div>
      )}
    </Form>
  );
}
```

```
</Form>

);
}
```

Route Action Handler (Backend Processing):

javascript

// This handles the form submission on the server/backend

```
export async function checkoutAction({ request }) {
  if (request.method !== 'POST') {
    return null;
  }

  const formData = await request.formData();
  const order = {
    firstName: formData.get('firstName'),
    lastName: formData.get('lastName'),
    email: formData.get('email'),
    address: formData.get('address')
  };

  try {
    const response = await fetch('/api/orders', {
      method: 'POST',
      body: JSON.stringify(order),
      headers: { 'Content-Type': 'application/json' }
    });

    if (!response.ok) {
      return { error: 'Order failed. Please try again.' };
    }

    const savedOrder = await response.json();
    return { success: true, orderId: savedOrder.id };
  } catch (error) {
    return { error: error.message };
  }
}
```

Key Features:

- **Method control** - Supports GET, POST, PUT, DELETE

- **Form data handling** - Automatic form data parsing
 - **Progressive enhancement** - Works even without JavaScript
 - **Action integration** - Automatically routes to route actions
 - **Error handling** - Can return error data to the component
-

5. `redirect()` Function

What it does:

`redirect()` programmatically sends users to a different route, typically used in route actions after form submission.

E-Commerce Analogy:

After a customer successfully completes their purchase, they're automatically redirected to an **"Order Confirmation" page**. This is like the store clerk saying "Go to window 5 to pick up your receipt."

Code Example:

```
javascript
```

```

import { redirect } from 'react-router-dom';

export async function loginAction({ request }) {
  const formData = await request.formData();
  const email = formData.get('email');
  const password = formData.get('password');

  try {
    const user = await authenticate(email, password);

    if (user) {
      // Successfully logged in, redirect to account page
      return redirect('/account');
    } else {
      return { error: 'Invalid credentials' };
    }
  } catch (error) {
    return { error: error.message };
  }
}

export function LoginPage() {
  const actionData = useActionData();

  return (
    <Form method="post" action="/login">
      <input type="email" name="email" placeholder="Email" required />
      <input type="password" name="password" placeholder="Password" required />
      <button type="submit">Login</button>
      {actionData?.error && <p className="error">{actionData.error}</p>}
    </Form>
  );
}

```

When to use redirect():

- After successful form submission
- User authentication/authorization checks
- Route protection (redirect unauthorized users)
- Data validation failures
- Session expiration

Route Configuration:

javascript

```
import { createBrowserRouter } from 'react-router-dom';

const router = createBrowserRouter([
  {
    path: '/login',
    element: <LoginPage />,
    action: loginAction
  },
  {
    path: '/account',
    element: <AccountPage />
  }
]);
```

6. `useNavigate()` Hook

What it does:

`useNavigate()` is a React hook that gives you programmatic control over navigation. It lets you navigate imperatively rather than declaratively.

E-Commerce Analogy:

Imagine a **"Go Back" button in the product details page** or an **"Add to Cart and Continue Shopping" button**. These require JavaScript logic to decide where to send the user. `useNavigate()` is like the store clerk pointing customers in different directions based on what they're doing.

Code Example:

javascript

```
import { useNavigate } from 'react-router-dom';

export function ProductDetails() {
  const navigate = useNavigate();

  const handleAddToCart = async (productId) => {
    try {
      // Add to cart logic
      await addToCart(productId);

      // After successful addition, navigate to cart
      navigate('/cart', { replace: true });
    } catch (error) {
      console.error('Failed to add to cart:', error);
    }
  };

  const handleGoBack = () => {
    navigate(-1); // Go back to previous page
  };

  return (
    <div>
      <button onClick={handleGoBack}><← Back to Products</button>
      <button onClick={() => handleAddToCart(123)}>Add to Cart</button>
    </div>
  );
}
```

Common Navigation Patterns:

javascript

// Navigate forward to a specific route

```
navigate('/products');
```

// Navigate with state data

```
navigate('/order-confirmation', {  
  state: { orderId: '12345' }  
});
```

// Navigate back in browser history

```
navigate(-1); // Go back one page
```

```
navigate(-2); // Go back two pages
```

// Replace current history entry (don't create back button)

```
navigate('/checkout', { replace: true });
```

// Navigate with search parameters

```
navigate('/products?category=electronics&sort=price');
```

Advanced Example - Search and Filter:

javascript

```
import { useNavigate, useSearchParams } from 'react-router-dom';

export function ProductFilter() {
  const navigate = useNavigate();
  const [searchParams] = useSearchParams();

  const handleCategoryChange = (category) => {
    navigate(`/products?category=${category}&sort=${searchParams.get('sort')}`);
  };

  const handleSearch = (query) => {
    navigate(`/products/search?q=${query}`);
  };

  return (
    <div>
      <input
        type="text"
        onChange={(e) => handleSearch(e.target.value)}
        placeholder="Search products..."
      />
      <button onClick={() => handleCategoryChange('electronics')}>
        Electronics
      </button>
    </div>
  );
}
```

When to use useNavigate():

- Complex conditional navigation
- Event-based navigation (button clicks)
- Navigation after API calls
- Dynamic navigation based on user input
- Navigation with state passing

7. Comparison Table

Feature	<Link>	<NavLink>	<Form>	redirect()	useNavigate()
Use Case	Simple navigation	Menu items	Form submission	After actions	Programmatic navigation

Feature	<Link>	<NavLink>	<Form>	redirect()	useNavigate()
HTML Output	<a> tag	<a> tag	<form> tag	N/A	N/A
Active State	✗	✓	N/A	N/A	N/A
Form Integration	✗	✗	✓	N/A	✗
Server-side Ready	✓	✓	✓	✓	✗
Client-side Only	✗	✗	✗	✗	✓

8. Full E-Commerce App Example

Router Setup:

```
javascript
```

```
import { createBrowserRouter } from 'react-router-dom';
import Layout from './Layout';
import HomePage from './pages/Home';
import ProductsPage, { productsLoader } from './pages/Products';
import CheckoutPage, { checkoutAction } from './pages/Checkout';
import OrderConfirmationPage from './pages/OrderConfirmation';

export const router = createBrowserRouter([
  {
    element: <Layout />,
    children: [
      {
        path: '/',
        element: <HomePage />
      },
      {
        path: '/products',
        element: <ProductsPage />,
        loader: productsLoader
      },
      {
        path: '/checkout',
        element: <CheckoutPage />,
        action: checkoutAction
      },
      {
        path: '/order-confirmation/:orderId',
        element: <OrderConfirmationPage />
      }
    ]
  }
]);
```

Navigation Layout:

javascript

```
import { NavLink, Outlet } from 'react-router-dom';

export default function Layout() {
  return (
    <div className="app">
      <header className="header">
        <nav className="navbar">
          <NavLink to="/"> 🏠 Home</NavLink>
          <NavLink to="/products"> 📦 Products</NavLink>
          <NavLink to="/checkout"> 🛒 Checkout</NavLink>
        </nav>
      </header>

      <main className="content">
        <Outlet />
      </main>
    </div>
  );
}
```

9. Key Takeaways for Students

1. `<Link>` = Simple, semantic navigation (like breadcrumbs)
2. `<NavLink>` = Link with active state styling (like the main menu)
3. `<Form>` = Form submission integrated with routes (like checkout forms)
4. `redirect()` = Programmatic server-side redirect (automatic page navigation)
5. `useNavigate()` = Client-side programmatic navigation (like custom buttons)

Best Practices:

- Use `<NavLink>` for navigation menus
- Use `<Link>` for inline navigation
- Use `<Form>` for all form submissions with route actions
- Use `redirect()` after successful server-side actions
- Use `useNavigate()` for complex client-side logic

Remember:

React Router v7 (Remix-like) prioritizes **data loading with loaders** and **data mutations with actions**, making

it easier to handle complex navigation scenarios in modern e-commerce applications.