

11강

LocalStorage

useNavigate()



LocalStorage





➤ LocalStorage 란?

- LocalStorage는 브라우저(Chrome, Edge, Safari 등)에 내장된 간단한 데이터 저장소이다.

특징	설명
저장 위치	사용자의 브라우저 내부(로컬 PC)
데이터 형식	<u>오직 문자열(string)만 저장 가능</u>
용량	<u>약 5MB 정도</u>
만료 시점	따로 설정하지 않으면 “영구적 저장” (쿠키와 달리 유효기간 없음)
사용 목적	로그인 정보, 테마 설정, 찜 목록, 장바구니 등 “유지되어야 하는 데이터” 저장



➤ LocalStorage 사용 방법

```
// 저장 (문자열만 가능)  
localStorage.setItem('키이름', '저장할값');  
  
// 불러오기  
const value = localStorage.getItem('키이름');  
  
// 삭제  
localStorage.removeItem('키이름');  
  
  
// 전체 초기화  
localStorage.clear();
```



➤ 기존에 작성한 wishlist에 LocalStorage 사용 하기

- wishlist가 배열이니까
- 문자열로 변환(JSON.stringify)
- 다시 객체로 복원(JSON.parse) 이 과정을 거쳐야 한다.

```
// 배열 저장
```

```
localStorage.setItem('wishlist', JSON.stringify(wishlist));
```

```
// 배열 불러오기
```

```
const savedList = JSON.parse(localStorage.getItem('wishlist')) || [];
```



➤ 기존에 작성한 wishlist에 LocalStorage 연결 하기 - 방법1

- WishlistContext.jsx 안에서
- 페이지가 처음 로드될 때 localStorage에서 기존 데이터 불러오기
- wishlist가 변경될 때마다 localStorage에 자동 저장

단, 다음 페이지에 작성한 방법-1번 방식으로 개발할 때 문제점은

- 1. 빈 배열 상태로 WishlistPage 렌더
 - 2. useEffect 실행 → localStorage 읽고 setWishlist()
 - 3. 다시 렌더링 → 이제 데이터 표시
-
- 그래서 실제로는 localStorage에는 값이 잘 들어있지만,
 - 첫 렌더링 시에는 빈 화면이 잠깐 보이거나,
 - 데이터가 표시되지 않는 것처럼 보일 수 있다는 점이다.



➤ 기존에 작성한 wishlist에 LocalStorage 연결 하기

```
// 찜 목록을 저장할 상태 변수 (초기값은 빈 배열)
const [wishlist, setWishlist] = useState([]);

// 1. 최초 렌더링 시 LocalStorage에서 불러오기
useEffect(() => {
  const saved = localStorage.getItem('wishlist');
  if (saved) {
    setWishlist(JSON.parse(saved)); //상태 갱신
  }
}, []);

// 2. wishlist가 바뀔 때마다 LocalStorage에 저장
useEffect(() => {
  localStorage.setItem('wishlist', JSON.stringify(wishlist));
}, [wishlist]);
```

LocalStorage



- 방법1 일 때 -> LocalStorage 개발자 모드 F12에서 확인하기

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the left sidebar, under 'LocalStorage', the '로컬 스토리지' item for the URL 'http://localhost:5173' is expanded. A red box highlights this item. In the main pane, a table displays the stored data:

키	값
wishlist	[{"id":1,"name":"노트북","price":1500000}]

- F5(새로 고침)함과 동시에 빈 배열이 들어감 -> 고로 F5를 눌러도 저장된 데이터 보이지 않는다.

The screenshot shows the same Chrome DevTools interface after a refresh (F5). The 'LocalStorage' entry for 'http://localhost:5173' is still expanded in the sidebar, but the table in the main pane now shows an empty array for the 'wishlist' key:

키	값
wishlist	[]



➤ 기존에 작성한 wishlist에 LocalStorage 연결 하기 - 방법 2

- 자연 초기화 방식으로 아래처럼 `useState(() => { ... })`를 다시 사용하면
- 렌더링 전에 딱 한 번만 `localStorage`에서 읽기 때문에
- 빈 배열 순간 자체가 없다.
- 이 방식이 React 공식 흄(문서)에서도 추천하는 방식이다.

```
// // 1. 최초 렌더링 시 LocalStorage에서 불러오기
// 기본 아래 방법을 추천
const [wishlist, setWishlist] = useState(() => {
  const saved = localStorage.getItem('wishlist');
  return saved ? JSON.parse(saved) : []; // 저장된 게 있으면 복원, 없으면 빈 배열
});

// 2, wishlist가 바뀔 때마다 LocalStorage에 저장
useEffect(() => {
  localStorage.setItem('wishlist', JSON.stringify(wishlist));
}, [wishlist]);
```

LocalStorage



➤ 방법2 일 때 -> LocalStorage 개발자 모드 F12에서 확인하기

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. On the left, there's a sidebar with '애플리케이션' (Application) section containing '매니페스트', 'Service workers', and '저장용량'. Below it is the '저장용량' (Storage) section, which is expanded and shows a list under '로컬 스토리지' (LocalStorage). One item in the list is highlighted with a red box and contains the URL 'http://localhost:5173'. To the right, the main panel displays the contents of the 'wishlist' key. A red box highlights the entire table row for the 'wishlist' key, which contains the value: `[{"id":1,"name":"노트북","price":1500000}, {"id":2,"name":"마우스","price":30000}]`.

키	값
wishlist	[{"id":1,"name":"노트북","price":1500000}, {"id":2,"name":"마우스","price":30000}]

➤ F5(새로 고침)를 누르고 다시 짐 목록을 클릭하고 확인해도 짐 목록이 그대로 저장되어 있는 걸 확인 할 수 있다.

This screenshot is identical to the one above, showing the same DevTools interface and Local Storage data. The 'wishlist' key still contains the same value: `[{"id":1,"name":"노트북","price":1500000}, {"id":2,"name":"마우스","price":30000}]`. The red boxes highlighting the 'LocalStorage' entry in the sidebar and the table row for 'wishlist' remain in place, demonstrating that the data is preserved across page reloads.

키	값
wishlist	[{"id":1,"name":"노트북","price":1500000}, {"id":2,"name":"마우스","price":30000}]



➤ LocalStorage 동작의 흐름

단계	동작 내용
(1)	페이지 처음 열 때, LocalStorage에 "wishlist"가 있으면 그걸 불러옴
(2)	사용자가 "찜하기" 클릭 → 상태(wishlist) 업데이트
(3)	상태가 바뀔 때마다 useEffect가 실행되어 LocalStorage에도 저장
(4)	페이지 새로고침해도 LocalStorage의 데이터가 다시 복원됨



➤ LocalStorage 삭제하기

- WishlistContext.jsx 에 삭제 함수 인 clearWishlist 생성한다.
- WishlistPage.jsx 페이지에 짐한 목록이 있을 때만 전체삭제 버튼 보이기로 작성하고 전체 삭제 버튼 클릭하면 LocalStorage에서 삭제되도록 작성한다.

```
/** 전체 짐 목록 삭제 (localStorage + 상태 모두 초기화) */
const clearWishlist = () => {
  setWishlist([]); // 상태 초기화
  localStorage.removeItem('wishlist'); // localStorage에서 삭제
};
```



➤ LocalStorage 삭제하기

- WishlistPage.jsx 페이지에 짐한 목록이 있을 때만 전체삭제 버튼 보이기로 작성하고 전체 삭제 버튼 클릭하면 LocalStorage에서 삭제되도록 작성한다.

```
/* 짐 목록이 있을 때만 전체삭제 버튼 보이기 */
{wishlist.length > 0 && (
  <button
    onClick={clearWishlist}
    style={{
      background: 'tomato',
      color: 'white',
      border: 'none',
      padding: '8px 12px',
      borderRadius: '8px',
      cursor: 'pointer',
      marginBottom: '15px',
    }}
  >
    전체 삭제
  </button>
)}
```

useNavigate()





➤ useNavigate() 란?

R, fbr ~ ...

- useNavigate는 자바스크립트 코드 안에서 페이지 이동(라우팅)을 제어할 때 사용하는 툥이다.
- 즉, 버튼 클릭, 로그인 성공 등 특정 조건일 때 자동으로 페이지를 이동시킬 수 있다.
- React Router v6 이후부터 useHistory() 대신 쓰는 최신 방식이다.
- Link는 **태그로 이동**,
- useNavigate()는 코드로 이동하는 방식이다.

➤ 기존 Link와의 차이점

비교	Link 컴포넌트	useNavigate()
형태	<Link to="/">홈</Link>	navigate('/')
사용 위치	JSX 안에서	JS 코드(이벤트, 조건문 등)
주 사용 시점	클릭 시 단순 이동	로그인 성공 후, 조건 충족 시, 함수 안에서 이동 등

useNavigate()



➤ Home.jsx

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

export default function HomePage() {
  const navigate = useNavigate();

  return (
    <div style={{ padding: '50px', backgroundColor: '#ffcccb' }}>
      <h1>🏠 홈페이지입니다</h1>
      <p>현재 경로: /</p>

      <button onClick={() => navigate('/about')} style={{ margin: '10px', padding: '10px' }}>
        소개 페이지로 이동
      </button>

      <button onClick={() => navigate('/contact')} style={{ margin: '10px', padding: '10px' }}>
        연락처 페이지로 이동
      </button>
    </div>
  );
}
```



➤ About.jsx

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

export default function AboutPage() {
  const navigate = useNavigate();

  return (
    <div style={{ padding: '50px', backgroundColor: '#add8e6' }}>
      <h1> 소개 페이지입니다 </h1>
      <p>현재 경로: /about</p>

      <button
        onClick={() => navigate('/')}
        style={{ margin: '10px', padding: '10px' }}
      >
        홈으로 이동
      </button>

      <button
        onClick={() => navigate('/contact')}
        style={{ margin: '10px', padding: '10px' }}
      >
        연락처 페이지로 이동
      </button>
    </div>
  );
}
```

useNavigate()



➤ Contact.jsx

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

export default function ContactPage() {
  const navigate = useNavigate();

  return (
    <div style={{ padding: '50px', backgroundColor: '#90ee90' }}>
      <h1>👋 연락처 페이지입니다</h1>
      <p>현재 경로: /contact</p>

      <button
        onClick={() => navigate('/')}
        style={{ margin: '10px', padding: '10px' }}
      >
        홈으로 이동
      </button>

      <button
        onClick={() => navigate('/about')}
        style={{ margin: '10px', padding: '10px' }}
      >
        소개 페이지로 이동
      </button>
    </div>
  );
}
```



➤ App.jsx

```
import AboutPage from './Example/About';
import ContactPage from './Example/Contact';
import HomePage from './Example/Home';

function App() {

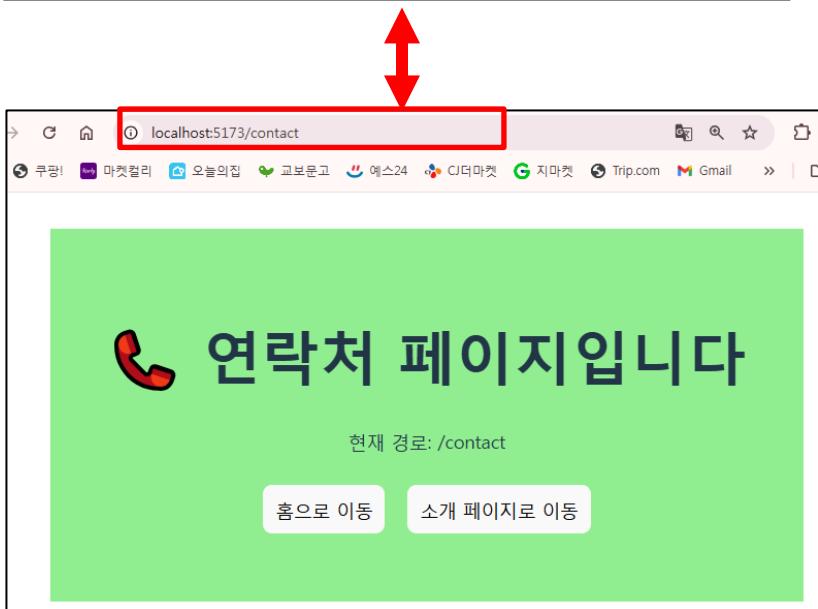
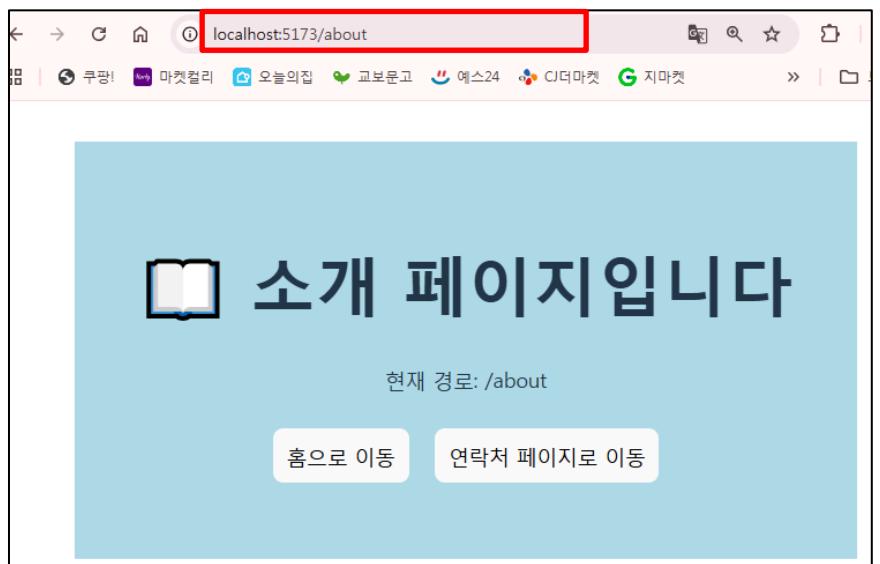
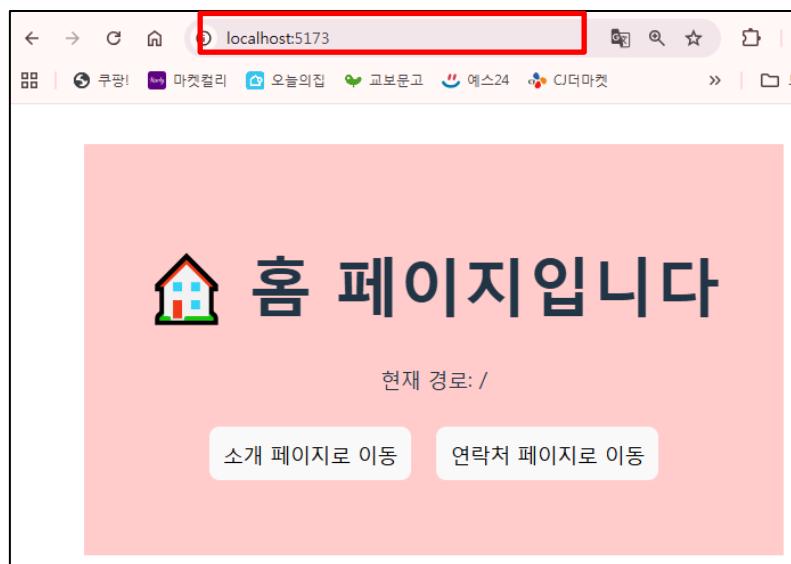
  <><
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<HomePage />} />
      <Route path="/about" element={<AboutPage />} />
      <Route path="/contact" element={<ContactPage />} />
    </Routes>
  </BrowserRouter>
</>
};

}
```

useNavigate()



➤ 출력 화면





➤ 여기서 잠깐

- ① useNavigate()로 로그인 하기
- ② 로그인 페이지에서 id/pw 입력 후 로그인 성공
- ③ useNavigate()로 메인 페이지(/home) 자동 이동
- ④ 메인 페이지 상단에 로그인한 사용자 이름 표시
- ⑤ Src -> LoginExample 폴더 생성 후 -> Context, Pages 폴더 생성 한다.
- ⑥ Context 폴더 생성 후 -> AuthContext.jsx 파일을 생성한다.
- ⑦ Pages 폴더 생성 후 -> HomePage.jsx, LoginPage.jsx 파일을 생성한다.

<초기 화면>

A screenshot of a web browser window. The address bar shows 'localhost:5173'. The main content area has a light gray background with the text '로그인 페이지' centered. Below it are two input fields: one labeled '아이디' and another labeled '비밀번호'. At the bottom is a blue button labeled '로그인'.

<로그인 성공 시 화면>

A screenshot of a web browser window. The address bar shows 'localhost:5173'. A modal dialog box is displayed with the text 'localhost:5173 내용:' at the top, followed by '로그인 성공!', and a red '확인' (Confirm) button. Below the modal is the text '로그인 페이지' and two input fields: 'admin' in the '아이디' field and '....' in the '비밀번호' field. At the bottom is a blue '로그인' button.

<로그인 성공 후 Home으로 자동 이동 됨>

A screenshot of a web browser window. The address bar shows 'localhost:5173/home'. The main content area displays the text '안녕하세요, admin님' (Hello, admin님) in a large font. At the bottom right is a blue '로그아웃' (Logout) button.



➤ AuthContext.jsx

```
import { createContext, useState } from 'react';

export const AuthContext = createContext();

export default function AuthProvider({ children }) {
  const [user, setUser] = useState(null); // 로그인한 사용자 정보 저장

  const login = (username) => {
    setUser(username);
  };

  const logout = () => {
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}
```



➤ LoginPage.jsx

```
import { useState, useContext } from 'react';
import { useNavigate } from 'react-router-dom';
import { AuthContext } from './Context/AuthContext';

export default function LoginPage() {
  const [id, setId] = useState("");
  const [pw, setPw] = useState("");
  const navigate = useNavigate();
  const { login } = useContext(AuthContext);

  const handleLogin = () => {
    if (id === 'admin' && pw === '1234') {
      login(id); // Context에 사용자 정보 저장
      alert('로그인 성공!');
      navigate('/home'); // 페이지 이동
    } else {
      alert('아이디 또는 비밀번호가 올바르지 않습니다.');
    }
  };
}
```

```
return (
  <div style={{ textAlign: 'center', marginTop: '50px' }}>
    <h2>로그인 페이지</h2>
    <input
      type="text"
      placeholder="아이디"
      value={id}
      onChange={(e) => setId(e.target.value)}
    />
    <br />
    <input
      type="password"
      placeholder="비밀번호"
      value={pw}
      onChange={(e) => setPw(e.target.value)}
    />
    <br />
    <button onClick={handleLogin}>로그인</button>
  </div>
);
```



➤ HomePage.jsx

```
import { useContext } from 'react';
import { AuthContext } from './Context/AuthContext';
import { useNavigate } from 'react-router-dom';

export default function HomePage() {
  const { user, logout } = useContext(AuthContext);
  const navigate = useNavigate();

  const handleLogout = () => {
    logout();
    navigate('/'); // 로그아웃 후 로그인 페이지로 이동
  };

  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      {user ? (
        <>
          <h2>안녕하세요, {user}님 </h2>
          <button onClick={handleLogout}>로그아웃</button>
        </>
      ) : (
        <>
          <h2>로그인이 필요합니다.</h2>
          <button onClick={() => navigate('/')}>로그인하러 가기</button>
        </>
      )}
    </div>
  );
}
```



➤ App.jsx

```
import AuthProvider from './LoginExample/Context/AuthContext';
import LoginPage from './LoginExample/Pages/LoginPage';
import HomePage from './LoginExample/Pages/HomePage';

function App() {

  <>
  <AuthProvider>
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<LoginPage />} />
        <Route path="/home" element={<HomePage />} />
      </Routes>
    </BrowserRouter>
  </AuthProvider>
  </>

};

}
```