중첩 라우팅

부모-자식 관계를 가진 라우트 구조

부모 라우트가 공통 레이아웃을 제공하고, 자식 라우트가 그 안에서 렌더링되는 구조

왜 중첩 라우팅을 사용하나요?

1. 코드 재사용

여러 페이지가 공통 레이아웃을 공유할 때, 레이아웃 코드를 중복 작성하지 않아도 됩니다.

2. 일관된 UI

같은 섹션의 페이지들이 동일한 네비게이션과 레이아웃을 가집니다.

3. 유지보수 용이

레이아웃을 변경할 때 한 곳만 수정하면 모든 자식 라우트에 적용됩니다.

4. URL 구조화

논리적인 URL 구조를 만들 수 있습니다:

- `/dashboard` → 대시보드 홈
- `/dashboard/expenses` → 지출 목록
- `/dashboard/settings` → 설정

중첩 라우팅 구현

부모 라우트(`Layout`)가 공통 레이아웃을 제공하고, 자식 라우트들이 그 안에서 렌더링됩니다.

Outlet은 React Router에서 자식 라우트를 렌더링할 위치를 지정하는 컴포넌트입니다.

기본 사용법 // Layout.jsx

```
import { Outlet } from 'react-router-dom';
function Layout() {
     return (
          <div>
               <header>공통 헤더</header>
               <nav>공통 네비게이션</nav>
               {/* 자식 라우트가 여기에 렌더링됨 */}
               <main>
                    <Outlet />
               </main>
               <footer>공통 푸터</footer>
          </div>
     );
}
```

실습 1. 중첩 라우팅 구현하기

1. 현재 App.jsx의 구조를 중첩 라우팅으로 변경하세요
2. NavigationBar를 Layout 안으로 이동하세요
3. Outlet을 사용하여 자식 라우트를 렌더링하세요

실습 2. 로그인-회원가입 구현하기

API URL : http://13.220.93.143:8080

- 사용자 회원가입 기능 **엔드포인트** `POST /api/auth/signup`
- 사용자 로그인 기능 엔드포인트 `POST /api/auth/login`

요청 본문 (LoginRequest)

```
{
	"email": "string", // 필수
	"password": "string" // 필수
}
```

응답 (AuthResponse):

```
{
	"token": "string", // JWT 토큰
	"email": "string",
	"name": "string",
	"userId": 123, // integer
	"message": "string"
}
```

실습 2. 로그인-회원가입 구현하기

- 인증 상태 관리
- 보호된 라우트 구현
- 로그인 상태 유지

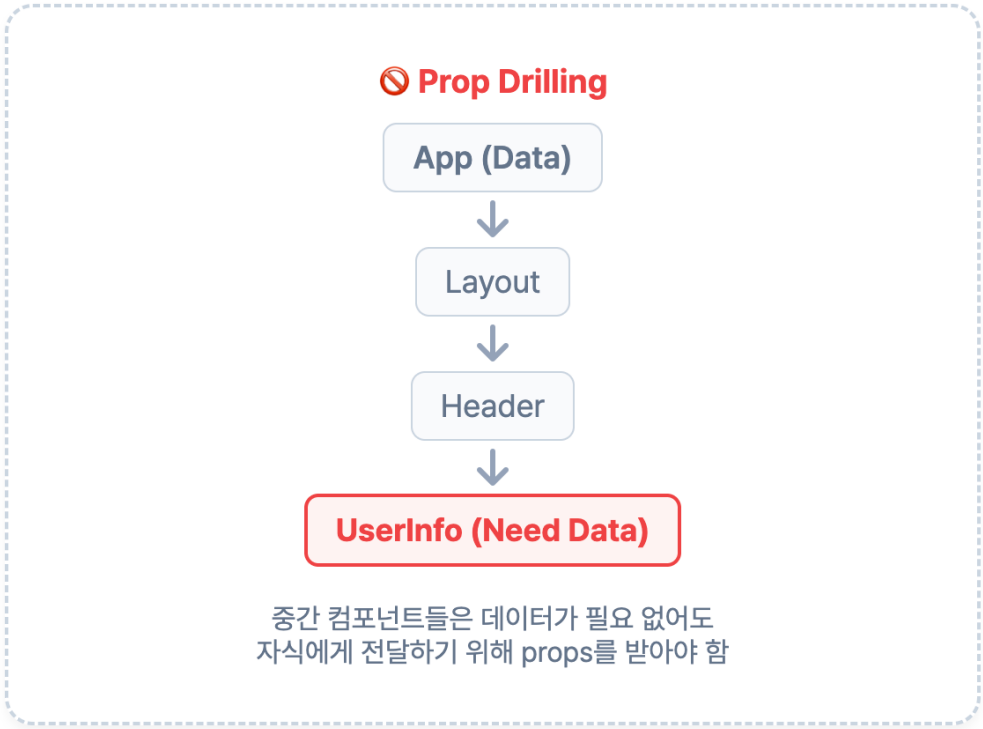
1. 모든 API 요청을 처리하는 공통 함수를 만들고, 회원가입/로그인 API 함수를 구현

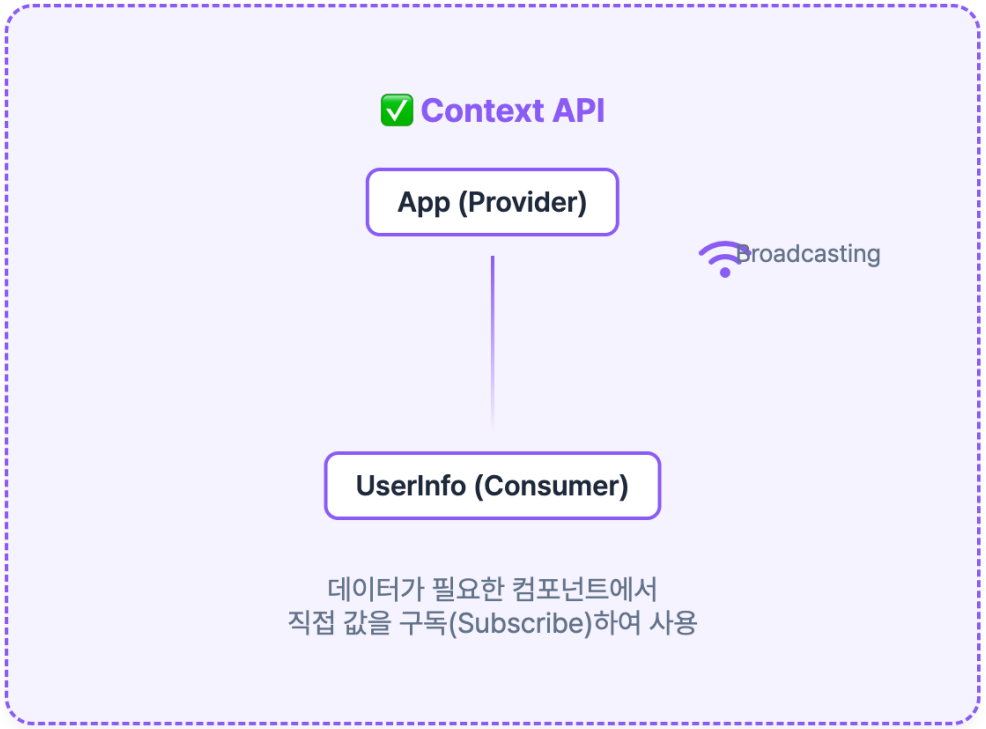
src/utils/api.js

```
async function apiRequest(endpoint, options = {}) {
  const url = `${API_BASE_URL}${endpoint}`;

  const defaultHeaders = {
    'Content-Type': 'application/json',
  };

  // 토큰이 있으면 헤더에 추가
  const token = localStorage.getItem('token');
  if (token) {
    defaultHeaders['Authorization'] = `Bearer ${token}`;
  }

  const config = {
    ...options,
    headers: {
      ...defaultHeaders,
      ...options.headers,
    },
  };
```

Context API의 필요성

Prop Drilling 문제 해결





Context API로 전역 인증 상태 관리

Step 1 Context 생성 (Create)

Step 2 범위 지정 및 값 공급 (Provide)

Step 3 데이터 사용 (Consume)

```
import { createContext, useContext, useState, useEffect } from 'react';
import { login as loginAPI, signup as signupAPI } from '../utils/api';

const AuthContext = createContext(null);
```

```
export function AuthProvider({ children }) {
  // ============================================
  // 1단계: useState - 상태 초기화
  // ============================================
  const [user, setUser] = useState(null);
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [loading, setLoading] = useState(true);
```

인증 상태 관리 – AuthProvider

```jsx
export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [loading, setLoading] = useState(true);

  // 컴포넌트 마운트 시 localStorage에서 인증 정보 확인
  useEffect(() => {
    const token = localStorage.getItem('token');
    const userData = localStorage.getItem('user');

    if (token && userData) {
      try {
        const parsedUser = JSON.parse(userData);
        setUser(parsedUser);
        setIsAuthenticated(true);
      } catch (error) {
        console.error('사용자 데이터 파싱 오류:', error);
        localStorage.removeItem('token');
        localStorage.removeItem('user');
      }
    }
    setLoading(false);
  }, []);
```

1. 상태 초기화 (Persistence)

새로고침 시 JS 메모리는 초기화되지만, `localStorage`는 유지됩니다. `useEffect`를 통해 앱 시작 시 토큰 유무를 확인하고 로그인 상태를 복구합니다.

2. 로그인 액션 (Encapsulation)

단순히 API만 호출하는 것이 아니라, **응답 처리 → 토큰 저장 → 상태 업데이트**까지 한 번에 처리하여 컴포넌트 코드를 단순화합니다.

JWT 인증 시스템 구현

1.로그인 요청: Client (React) → POST /api/auth/login → Server (Spring Boot)

2.토큰 발급: Server 검증 성공 → JWT Token 반환

3.토큰 저장: Client → localStorage에 Token 저장

(실무에서는 쿠키/HttpOnly 권장이나 학습용으로 로컬스토리지 사용)

4. 인증 요청: 이후 모든 API 요청 헤더에 Authorization: Bearer {Token} 추가

5. 라우트 보호: AuthContext가 토큰 유무 확인 → 없으면 로그인 페이지로 강제 리다이렉트

```
// 토큰이 존재하면 자동으로 헤더에 주입 (Interceptors 역할)
const token = localStorage.getItem('token');
if (token) {
  defaultHeaders['Authorization'] = `Bearer ${token}`;
}
```