📌 React 비동기 처리와 Axios

🚀 1. 비동기 처리란?

📖 개념

- 비동기 처리(Asynchronous)는 **작업이 완료될 때까지 기다리지 않고, 다른 작업을 병행하여 처리하는 방식**입니다.
- React에서는 주로 **API 요청, 파일 업로드, 타이머, 애니메이션 등**에서 비동기 로직이 필요합니다.

✅ JavaScript 비동기 처리 방법

1. **콜백 함수 (callback)**
2. **Promise**
3. **async / await** ← 현대 React에서 가장 널리 사용

💡 2. Axios란?

📖 개념

- `Axios`는 **Promise 기반의 HTTP 클라이언트 라이브러리**로, React에서 외부 API와 통신할 때 주로 사용됩니다.
- 브라우저와 Node.js 환경 모두에서 작동하며, 사용법이 간단하고 다양한 기능(인터셉터, 타임아웃, 헤더 설정 등)을 지원합니다.

✅ 설치

```
npm install axios
```

✅ 기본 사용법

```
import axios from 'axios';

axios.get('/api/users')
  .then(response => console.log(response.data))
  .catch(error => console.error(error));
```

🛠️ 3. React에서 비동기 처리하기

📌 useEffect + async/await 조합

```
import { useEffect, useState } from 'react';
import axios from 'axios';

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        const response = await axios.get('https://jsonplaceholder.typicode.com/users');
        setUsers(response.data);
      } catch (e) {
        setError(e);
      } finally {
        setLoading(false);
      }
    };
    fetchUsers();
  }, []);

  if (loading) return <p>로딩 중...</p>;
  if (error) return <p>에러 발생: {error.message}</p>;

  return (
    <ul>
      {users.map(user => <li key={user.id}>{user.name}</li>)}
    </ul>
  );
}
```

✅ `useEffect` 내부에서 async 함수를 정의하고 즉시 호출하는 패턴이 일반적입니다.

🔁 4. GET / POST / PUT / DELETE 요청

📌 GET 요청

```
axios.get('/api/posts');
```

📌 POST 요청

```
axios.post('/api/posts', { title: 'React', body: '비동기 학습' });
```

📌 PUT 요청

axios.put('/api/posts/1', { title: '수정된 제목' });

📌 DELETE 요청

axios.delete('/api/posts/1');

🧠 5. 에러 처리 및 로딩 처리 패턴

✅ try-catch-finally 구조

- 비동기 함수 안에서 예외를 안전하게 다루고, 로딩 상태까지 반영
- UI에서 로딩 중 메시지, 에러 메시지 등을 분기 처리

✅ 공통 요청 처리 로직으로 분리

```
const fetchData = async (url, setState, setLoading, setError) => {
  try {
    setLoading(true);
    const response = await axios.get(url);
    setState(response.data);
  } catch (e) {
    setError(e);
  } finally {
    setLoading(false);
  }
};
```

✅ 반복되는 비동기 처리 로직을 함수로 추출하여 재사용성 향상

📦 6. Axios 인스턴스 생성

📌 공통 설정을 위한 커스터마이징

```
const api = axios.create({
  baseURL: 'https://api.example.com',
  timeout: 1000,
  headers: { 'Authorization': 'Bearer TOKEN' },
});
api.get('/posts');
```

✅ 인스턴스를 만들어두면 API 요청 시 반복되는 설정을 피할 수 있음

⚠️ 7. 주의사항

- `useEffect` 내부에서 직접 async 함수를 넣는 건 불가능 → 내부에 정의 후 호출해야 함
- 상태 변화가 많은 요청에서는 **로딩 상태, 에러 상태, 데이터 상태**를 모두 고려해야 함
- `axios.get`은 기본적으로 캐싱하지 않음 → 수동 캐시 또는 SWR, React Query 등 라이브러리와 병행 가능

🏁 8. 마무리

✅ 핵심 요약

- React에서 비동기 처리는 주로 `useEffect + async/await + axios` 조합으로 사용됨
- 로딩 상태, 에러 처리, 결과 데이터를 세분화하여 UI 구성 필요
- 반복되는 요청 코드는 함수로 분리하거나, Axios 인스턴스로 구성하여 유지보수성 향상
- 대규모 프로젝트에서는 React Query, SWR 등 상태 및 캐시 관리 도구 활용 고려