외부 생성형서비스의 코드를 현재 프로젝트에 적용할 때, 고려할 요소와 해결방법을 참고해.

1. 환경 설정 통일 및 중앙 관리

가장 먼저 해결해야 할 **CORS, API** 연동 문제입니다**.**

1. **CORS** 문제 해결 **(2**가지 접근법**)**
    - **(**추천**)** 백엔드에서 허용**:** 가장 근본적인 해결책입니다. 로컬 프론트엔드 주소(`http://localhost:3000` 등)를 백엔드 서버의 CORS 정책에 추가하여 요청을 허용해 주세요.

Node.js (Express) 예시**:**
JavaScript
```
const cors = require('cors');
const app = express();

const corsOptions = {
  origin: 'http://localhost:3000', // 프론트엔드 개발 서버 주소
  credentials: true, // 쿠키 등 자격 증명 허용
};

app.use(cors(corsOptions));
```

-
    - 프론트엔드 프록시**(Proxy)** 설정**:** 백엔드 수정이 어렵다면, 프론트엔드 개발 서버에 프록시를 설정하여 브라우저를 '속이는' 방법을 사용할 수 있습니다. 프론트엔드에서 `/api` 로 시작하는 모든 요청을 실제 백엔드 서버(`http://localhost:8080`)로 전달하게 만드는 것입니다.

React (`package.json`) 예시**:**
JSON
```
"proxy": "http://localhost:8080"
```

-

Vite (`vite.config.js`) 예시**:**
JavaScript
```
export default defineConfig({
  // ...
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:8080',
        changeOrigin: true,
      },
    },
```

```
  },
});
```

-

2. **API** 요청 로직 모듈화 **(JWT** 문제 해결**)** AI가 생성한 `fetch` 코드를 그대로 사용하지 마세요. 프로젝트의 중앙 **API** 클라이언트 모듈(보통 `axios` 인스턴스)을 통해 API를 요청하도록 코드를 수정해야 합니다.

`axios` 인터셉터 활용 예시 **(`utils/api.js`):**

JavaScript

```
import axios from 'axios';

const apiClient = axios.create({
  baseURL: process.env.REACT_APP_API_URL, // .env 파일에서 API 주소 관리
});

// 요청 인터셉터: 모든 요청에 JWT 헤더를 자동으로 추가
apiClient.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken'); // 토큰 가져오기
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
}, (error) => {
  return Promise.reject(error);
});

export default apiClient;
```

-

AI 생성 코드 수정**:**

JavaScript

```
// AI가 생성한 코드
// fetch('https://api.lovable.ai/data', { headers: ... })

// 수정된 코드
import apiClient from './utils/api';

const fetchData = async () => {
  const response = await apiClient.get('/data'); // 중앙 모듈 사용
  console.log(response.data);
};
```

-

3. 환경 변수 사용 (**.env**)**:** API 서버 주소, 외부에 노출되면 안 되는 키 값 등은 반드시 `.env` 파일을 통해 관리하고, AI가 생성한 코드에 하드코딩된 부분이 있다면 즉시 환경 변수로 교체하세요.

2. 점진적인 통합 및 리팩터링

AI가 만든 코드를 한 번에 전부 붙여넣지 말고, 단계적으로 통합해야 합니다.

1. 구조**(Structure)**와 스타일**(Style)** 분리**:**
   - 먼저 AI가 생성한 코드에서 **JSX/HTML** 구조와 **CSS** 스타일만 가져옵니다.
   - 데이터를 가져오거나 상태를 변경하는 로직(logic) 부분은 일단 제외합니다.
2. 기존 로직에 연결**:**
   - 가져온 JSX/HTML 구조에 프로젝트의 기존 상태 관리**(Redux, Context API** 등**)**와 **API** 호출 함수를 연결합니다.
   - 이렇게 하면 데이터 흐름의 일관성을 유지할 수 있습니다.
3. 리팩터링 및 컴포넌트 분리**:**
   - 가져온 코드가 너무 크다면, 프로젝트의 아키텍처에 맞게 여러 개의 작은 컴포넌트로 분리하여 재사용성을 높입니다.
   - `ESLint`, `Prettier`를 실행하여 코드 스타일을 통일합니다.

📌 핵심 요약

문제점	해결책
CORS 오류	**1. (**추천**)** 백엔드 CORS 설정에 프론트엔드 주소 추가 **2. (**차선책**)** 프론트엔드 개발 서버에 프록시(Proxy) 설정
JWT 인증 오류 **(401/403)**	`axios` 인터셉터와 같은 중앙 **API** 클라이언트 모듈을 만들어 JWT 헤더를 자동으로 삽입하도록 하고, AI 생성 코드가 이 모듈을 사용하도록 수정
하드코딩된 **API** 주소**/**키	`.env` 파일을 사용하여 환경 변수로 관리하고, 코드에서는 `process.env`로 참조
의존성 및 라이브러리 충돌	AI가 사용한 라이브러리를 확인하고, 프로젝트의 기존 라이브러리로 기능을 대체하거나 필요한 경우에만 신중하게 추가
코드 스타일 불일치	생성된 코드를 붙여넣은 후 즉시 **`ESLint --fix`**, **`Prettier`** 와 같은 포매터를 실행하여 스타일 통일
구조적 이질성	UI(JSX/CSS)와 로직(JS)을 분리하여 가져온 후, 프로젝트의 데이터 흐름과 컴포넌트 설계에 맞게 리팩터링