트러블 슈팅

OSSProj “돈쭐” 프론트엔드 배포 트러블슈팅
작성자: [서희정]
작성일시: 25. 06. 06.

● useSearchParams() 빌드 오류 발생

- 현상
  - /review 및 /submit_store 페이지에서 useSearchParams() 훅을 사용했을 때,
  - next build 명령어 실행 중 정적 사전 렌더링 단계에서 빌드 실패 발생
- 원인
  - 해당 페이지는 "use client"가 선언되어 있었지만,
  - next export 과정에서 useSearchParams()는 클라이언트 전용 훅이므로 서버에서 처리할 수 없음
  - Next.js는 next export 시 모든 페이지를 정적으로 사전 렌더링하려 하며, 이와 클라이언트 전용 훅이 충돌
- 해결
  - 각 페이지의 핵심 클라이언트 로직을 ClientXXXPageContent로 분리
  - 외부에서 <Suspense fallback="페이지를 로딩 중입니다...">로 감싸 렌더링을 지연 처리
  - /review: WriteReviewPage → ClientReviewPageContent
  - /submit_store: VerifyPage → ClientVerifyPageContent
- 최종 결과
  - 로컬 환경에서 빌드 성공 확인 (npm run build)

● Tailwind 관련 모듈 불러오기 실패

- **현상:**
  - Netlify 빌드 과정 중 다음과 같은 오류 발생:

    `Cannot find module 'tailwindcss'`

또는 `postcss` 관련 모듈 불러오기 실패

- **원인:**
  - `tailwindcss`, `autoprefixer`, `postcss` 모듈이 `devDependencies`에 포함되어 있었음
  - Netlify에서는 **정적 빌드 시점에도 해당 모듈이 필요**한데, 일부 환경에서는 `devDependencies`가 설치되지 않거나 생략되는 경우가 있음
  - 특히 `NODE_ENV=production`인 환경에서는 `npm install --production`처럼 devDependencies를 무시할 수 있음
- **해결:**
  - 아래 모듈들을 `devDependencies` → `dependencies`로 이동
    - `tailwindcss`
    - `autoprefixer`
    - `postcss`
  - 이후 Netlify에서 정상적으로 모듈 인식 및 빌드 성공
- **최종 구조:**

```
"dependencies": {
  ...
  "tailwindcss": "^3.4.17",
  "autoprefixer": "^10.4.21",
  "postcss": "^8.5.4"
}
```

- **참고:**
  - 해당 이슈는 Stack Overflow 및 GitHub 이슈에서 일부 Netlify 사용자들이 동일하게 겪은 사례로 보고되었으며, `tailwind` 공식 문서에서도 `production 빌드에 필요 시 dependencies로 이동할 것`을 권장

● Webpack alias 미설정으로 인한 모듈 경로 인식 실패

- **현상:**
  - 컴포넌트 import 시 `@/components/XXX` 형태로 작성된 코드에서 모듈을 찾지 못하는 오류 발생

- 오류 예: `Module not found: Can't resolve '@/components/Header'`

- **원인:**
  - `@` alias를 사용하고 있었지만, `next.config.ts`의 webpack 설정에 `@`에 대한 alias 지정이 빠져 있었음
  - Next.js는 기본적으로 `@` alias를 제공하지 않으므로 명시적으로 경로 설정이 필요
- **해결:**
  - `next.config.ts`에 `webpack` 설정을 추가하여 `@ → src` 경로 alias 등록

```
import path from "path";

const nextConfig = {
  webpack(config) {
    config.resolve.alias = {
      ...(config.resolve.alias || {}),
      "@": path.resolve(__dirname, "src"),
    };
    return config;
  },
};

export default nextConfig;
```

- **결과:**
  - 이후 `@/components/...`, `@/store/...` 등 상대경로 없이 편리하게 import 가능
  - 프로젝트 내 경로 관리가 통일되어 유지보수성이 향상됨

● TypeScript 인식 오류로 Netlify 빌드 실패

- **현상:**
  - Netlify 배포 중 TypeScript 관련 모듈을 찾을 수 없다는 오류 발생
  - 에러 로그 예시:

```
Cannot find module 'typescript'
```

- **원인:**
  - Netlify는 루트 디렉토리에서만 `typescript`를 인식하는데,

    현재 프로젝트 구조는 `src/frontend` 하위에 TypeScript 관련 설정과 코드가 존재함
  - 이로 인해 루트 기준으로 `node_modules`를 설치하는 Netlify 빌드 환경에서 `tsc` 실행 시 오류 발생
- **해결:**
  - **루트 디렉토리에 `typescript` 설치**

```
npm install typescript --save-dev
```

- **결과:**
  - `typescript` 인식 오류 해결
  - Netlify에서 정상적으로 Next.js + TypeScript 프로젝트 빌드 완료

● Mixed Content 오류로 인한 API 호출 실패

- **현상:**
  - HTTPS 환경(Netlify)에서 브라우저가 HTTP API 요청을 차단
  - 콘솔 에러 예시:

```
Mixed Content: The page at 'https://donzzul.netlify.app' was loaded over HTTPS,
but requested an insecure resource 'http://13.125.255.84/users/login'. This request has been blocked.
```

- **원인:**
  - Netlify는 기본적으로 HTTPS를 사용하지만, EC2 백엔드 서버는 HTTP만 지원
  - 브라우저 보안 정책 상 HTTPS → HTTP 요청은 Mixed Content로 간주되어 차단됨
  - 기존에는 `NEXT_PUBLIC_API_URL`을 통해 직접 EC2 URL 호출 (`http://13.125.255.84`)

    → 모든 요청에서 차단 발생

- **해결:**
  1. **Netlify 프록시 기능(`netlify.toml`) 사용하여 HTTPS 터널링 구성**

     ```
     [[redirects]]
     from = "/api/*"
     to = "http://13.125.255.84:80/:splat"
     status = 200
     force = true
     ```

  2. **모든 fetch 요청 경로를 `/api/...`로 변경**
     - 기존:

       ```
       fetch(`${process.env.NEXT_PUBLIC_API_URL}/users/login`)
       ```

     - 변경:

       ```
       fetch("/api/users/login")
       ```

  3. `.env` **내** `NEXT_PUBLIC_API_URL` **제거 및 관련 코드 정리**
- **결과:**
  - 브라우저 Mixed Content 오류 해소
  - 프론트 → Netlify 프록시 → EC2 구조로 안전하게 HTTPS 통신 가능
  - 환경변수 없이도 일관된 API 요청 가능