🗒️ 프로젝트명 (김채윤)

📖개요

프로젝트 목적 및 간단 설명

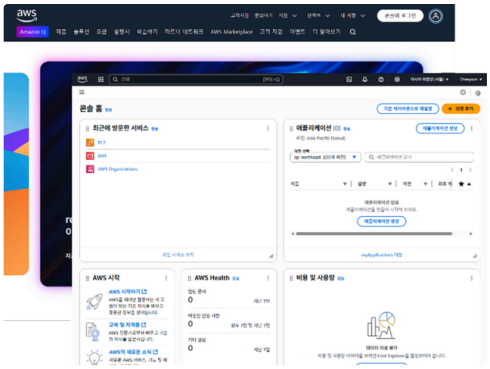
- **Elasticsearch를 이용한 기술 키워드 기반 오픈 데이터 검색 & 트렌드 분석 플랫폼**

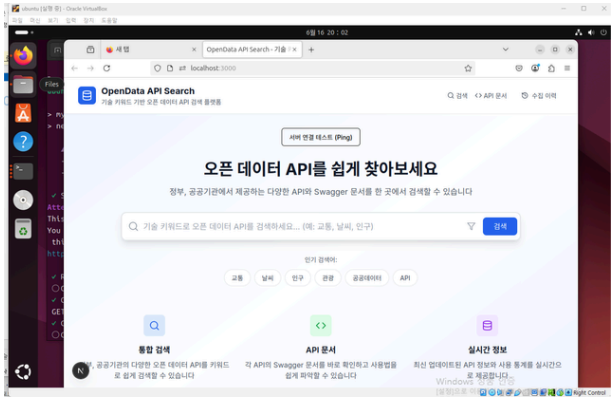
🗣️주요기능구현

기능명	설명	설정	완료여부
AWS EC2 서버 구축	초기 서버 배포 환경	• Ubuntu 20.04 LTS (EC2 인스턴스) • 포트 설정 및 보안 그룹 설정 • Spring Boot/Next.js+React • Nginx	❗이슈 발생
VirtualBox Ubuntu Linux 서버 이전	EC2 대신 로컬 리소스를 활용한 서버 환경 구축. Docker로 컨테이너 운영 가능	• Ubuntu 20.04 LTS (리눅스) • Docker 및 Docker Compose • MongoDB, Elasticsearch 설치 • Spring Boot/Next.js+React	✅완료
프론트 구성	React + Next.js 기반 회원가입, 로그인, 비밀번호 찾기 페이지 구성. Zustand로 상태 관리	• TypeScript + Tailwind CSS • Zustand 사용	✅완료
비밀번호 찾기 기능 백엔드 연결	입력 → 인증 코드 확인 → 새 비밀번호 입력 3단계 UI. Spring Boot + 이메일 인증 API 연동	• DB설정(mySql)	✅완료
소셜 로그인 (Google, Kakao, Naver) 백엔드 연결	OAuth2 인증 → JWT 발급 및 쿠키 저장 → 프론트 상태 연동. 로그인 후 기존 창 상태 변경 처리	• 소셜 로그인용 클라이언트 아이디 설정	✅완료

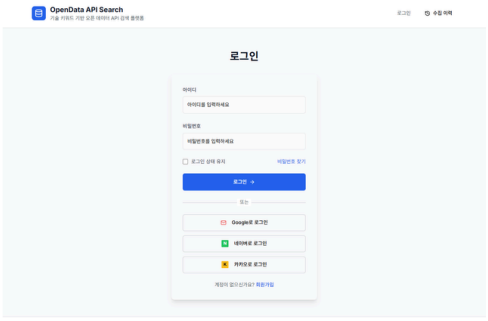
이미지 및 시각자료

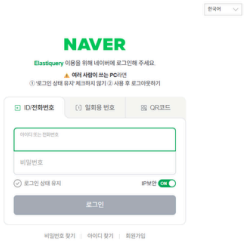
- AWS EC2 서버 구축 → Ubuntu virtual Box 서버 이전



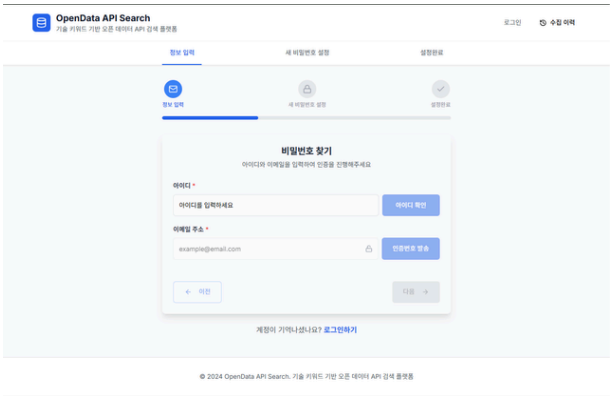


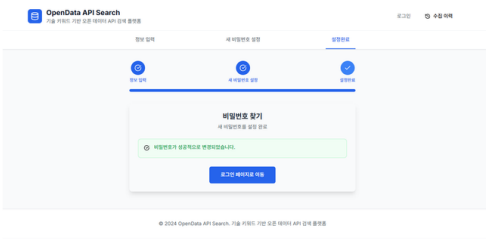
- 로그인



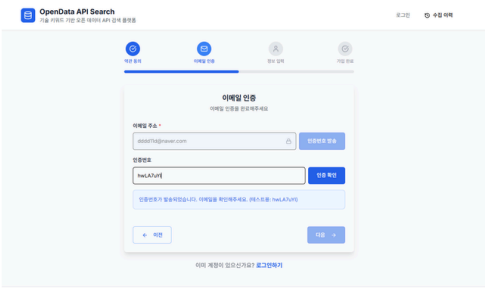


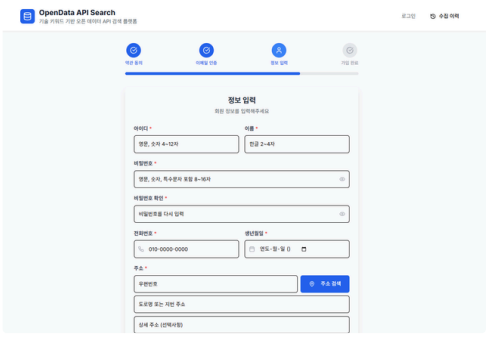
- 비밀번호 찾기

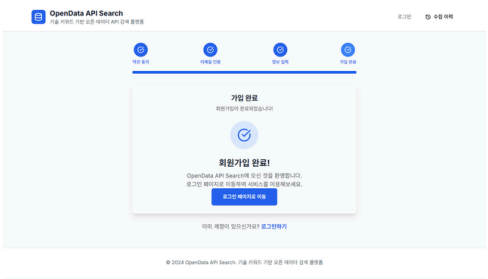




- 회원 가입







---

4. 코드 및 코드 설명

4.1 SPA + FormData + CSR(Client-Side Rendering) 기반 구조

```
export default function PasswordRecoveryPreview() {
  const [activeStep, setActiveStep] = useState<1 | 2 | 3>(1)
    const [userId, setUserId] = useState("")
    const [form, setForm] = useState<FormData>({
      userEmail: "",
      userId: "",
      userName: "",
      userPw: "",
      pwdConfirm: "",
    })
```

1. activeStep을 기반으로 한 3단계 구성(입력 → 인증 → 완료) : 전체 페이지 새로고침 없이 빠르게 렌더링 됨, 현재 단계에 따라 보여지는 UI가 유연하게 전환되어 사용자 흐름이 자연스러움
2. 입력 상태를 FormData 객체로 통합 관리 : FormData가 상태로 유지되기 때문에 다른 단계로 이동했다가 돌아와도 이전 값이 보존됨

3. CSR 중심 구조로 사용자 경험 강화 : 특정 필드 입력 여부에 따라 다음 단계 버튼 활성화, 에러 표시 등을 쉽게 제어 가능(상태 기반 조건부 렌더링)

4.2 Zustand 상태 관리 도입

```
const Header = () => {

   const isLoggedIn = useAuthStore((state) => state.isLoggedIn);
   const loading = useAuthStore((state) => state.loading);
   const logoutUser = useAuthStore((state) => state.logoutUser);
```

useAuthStore로 로그인 상태 제어

→ 컴포넌트 간 상태 공유와 추후 유지 보수가 용이

- useAuthStore : 인증관련 상태 저장소를 사용하는 훅

4.3 백엔드 API 연동 / 로직 분리 및 API 통신 처리 예외

```
<Button
  type="button"
  onClick={async () => {
    if (form.userPw === form.pwdConfirm && form.userPw.length >= 8) {
      try {
        const res = await changePassword(form.userId, form.userPw);
        console.log("전송된 userId:", form.userId);
        console.log("전송된 userPw:", form.userPw);
        if (res.success) {
          setPasswordChanged(true);
          setActiveStep(3); // 성공하면 다음 단계로 이동
        } else {
          alert("비밀번호 변경에 실패했습니다. 다시 시도해주세요.");
        }
      } catch (err: any) {
        console.error("비밀번호 변경 에러:", err);
        alert("서버 오류로 인해 비밀번호를 변경할 수 없습니다.");
      }
    } else {
      alert("비밀번호가 일치하지 않거나 형식이 잘못되었습니다.");
    }
  }}
  className="w-full h-12 bg-blue-600 hover:bg-blue-700 text-white"
>
```

1. 백엔드 API 연동을 통한 실시간 인증 및 데이터 처리
2. 로직 분리 및 API 통신 예외 처리

→ 실시간 인증, 에러 처리, 사용자 반응 속도에서 우수. 새로고침 x, 프론트와 백 각각 독립 유지보수 가능, API 단위로 테스트 쉬움

4.3 Tailwind CSS

```
  {activeStep === 1 && (
                  <Button
```

```
                    type="button"
                    onClick={() => setActiveStep(2)}
                    disabled={!emailVerified} // 이메일 인증되지 않으면 비활성화
                    className={`h-12 px-6 ${
                      emailVerified
                        ? "bg-blue-600 hover:bg-blue-700 text-white"
                        : "bg-gray-300 text-gray-500 cursor-not-allowed"
                    }`}
                  >
                    다음
                    <ArrowRight className="w-4 h-4 ml-2" />
                  </Button>
                )}
              </div>
            )}
```

1. 유틸리티 클래스 기반으로 한 줄의 클래스만으로 크기, 모양, 색상, 정렬 등 모두 설정 가능
2. sm:, md:, lg와 같은 접두어를 붙여 반응형 적용 가능
3. 디자인 시스템을 내장하고 있어 디자인 일관성 확보

✅ 이슈

- [ ] 초기 AWS 로 서버 배포를 완료했으나 빌드 과정에서 메모리 부족으로 인한 프로세스 중단 또는 컨테이너/인스턴스 비정상 종료 현상 발생 → Ubuntu Virtualbox로 서버 이전