# 주간 보고서 시스템 개발

## 전체 코드는 다음과 같습니다.

https://github.com/hanseungye/board_poject

### 현재 진행사항

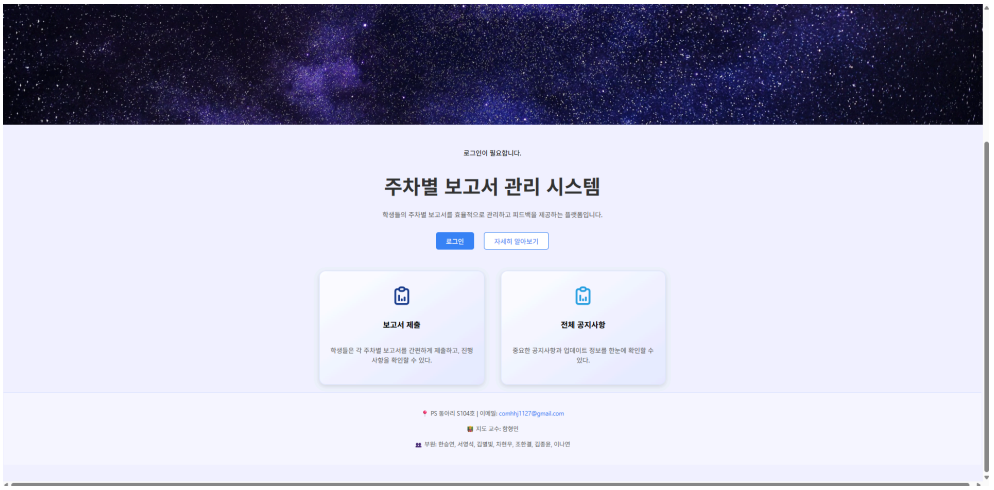
기존에는 Superbase를 데이터베이스로 사용하려고 하였으나, Superbase가 **IPv6를 지원하지 않아** Render(백엔드 배포 관리 클라우드)와의 연결이 원활하지 않았습니다.

이에 따라 데이터베이스를 **Neon(데이터베이스 관리 서비스)**으로 변경하였습니다.

또한, **클라우드 배포 환경은 Netlify를 사용**하고 있습니다.

# 1. 메인 페이지

- **구성:** 상단 내비게이션바, 공지사항/최근 글 미리보기, 로그인 진입 버튼
- **기능:**
  - 최신 게시글 목록 자동 로드
  - 로그인 전 사용자도 공지 확인 가능
  - 관리자용 “새 글 등록” 버튼은 로그인 후 표시됨
- **UI 포인트:**
  - 반응형 구조 (모바일/PC 자동 대응)
  - Tailwind 기반 카드 레이아웃 적용

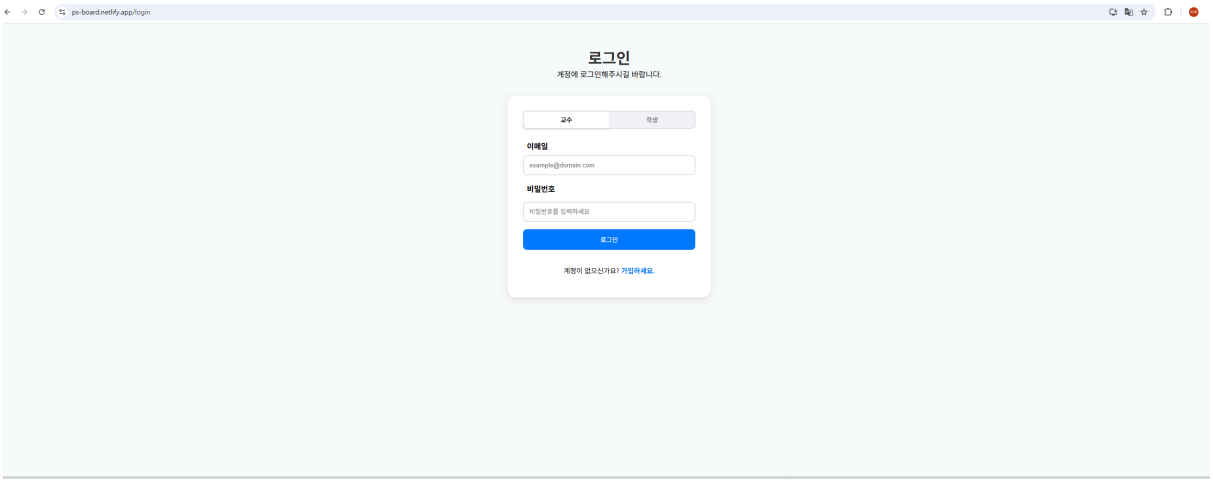



# 2. 로그인 페이지

- **기능 요약:**
  - 사용자 유형(교수 / 학생) 선택 후 로그인
  - PostgreSQL DB(`users` 테이블)에서 `role` 컬럼 기준으로 권한 구분
  - 로그인 성공 시 JWT 토큰 발급 → 세션 유지
- **교수(관리자)**

- 전체 게시글/회원 관리 가능

- **학생(일반 회원)**
  - 본인 작성 글만 수정/삭제 가능
- **보안 처리:**
  - 비밀번호 bcrypt 해시 저장
  - HTTPS 연결 enforced (Netlify + Render 간 TLS)

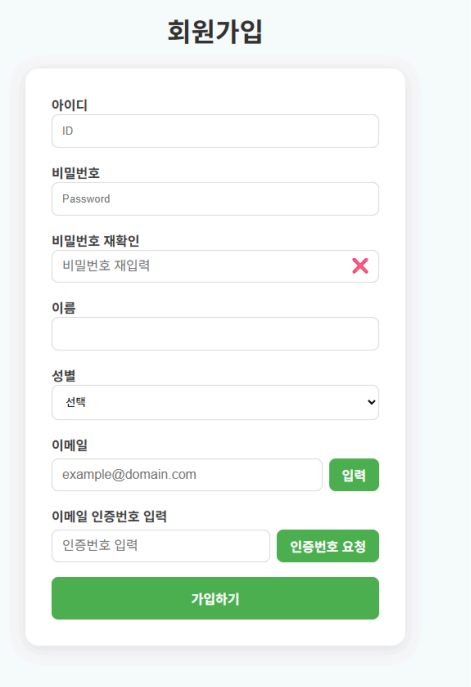



# 3. 회원가입 페이지

- **입력 항목:** 이름, 이메일, 비밀번호, 소속(학과)
- **권한 분류:**
  - 이메일 도메인( `@school.ac.kr` ) 기준으로 자동 role 설정
    - 교수: `professor@school.ac.kr` → `role='admin'`
    - 학생: 일반 도메인 → `role='user'`
- **검증:**
  - 중복 이메일 방지
  - 비밀번호 최소 길이 8자 이상 검증
- **DB 반영 예시:**

```
INSERT INTO users (name, email, password, role)
VALUES ('홍길동', 'hong@school.ac.kr', crypt('1234', gen_salt('bf')), 'user');
```

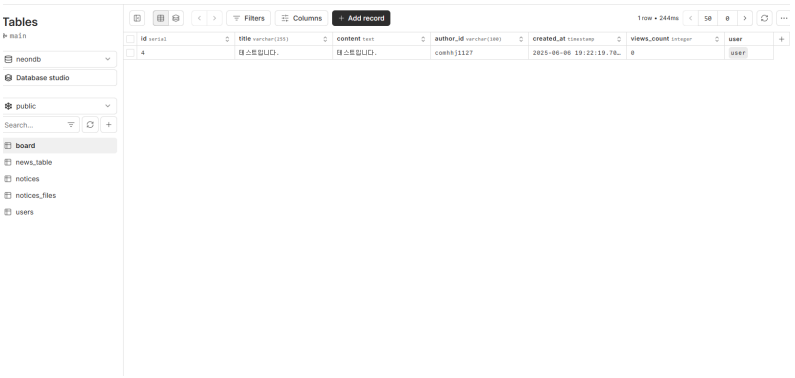
- **UX 포인트:**
  - 작성 완료 후 “게시글이 추가되었습니다.” Toast 알림
  - 관리자 페이지에서는 전체 글 조회/삭제 가능




## 게시판 페이지

- **구성:** 게시글 목록 + 작성 버튼
- **기능:**
  - “작성하기” 클릭 → `/posts/new` 진입
  - 작성 완료 시 DB(`posts` 테이블)에 자동 INSERT
  - 본인 글만 수정/삭제 가능 (권한 체크)
- **UX 포인트:**
  - 작성 완료 후 “게시글이 추가되었습니다.” Toast 알림
  - 관리자 페이지에서는 전체 글 조회/삭제 가능


# 5. 데이터베이스 및 게시판 기능 구현




# 1. 데이터베이스 설계 개요

- **DBMS:** PostgreSQL (Neon 클라우드 서비스 사용)
- **구조:** `public` 스키마 내에 총 5개 테이블(`users`, `board`, `news_table`, `notices`, `notices_files`)
- **역할 구분:**
  - `users`: 회원 정보 및 권한(role) 관리
  - `board`: 일반 게시판(학생/교수 공용)
  - `news_table`: 공지 뉴스 데이터
  - `notices`, `notices_files`: 공지사항 및 첨부파일 관리

## 🗂️ 주요 테이블 구조

| 테이블명 | 주요 컬럼 | 설명 |
|---|---|---|
| `users` | `id`, `email`, `password`, `role` | 회원 로그인 및 권한 구분(교수/학생) |
| `board` | `id`, `title`, `content`, `author_id`, `created_at`, `views_count` | 일반 게시판(글 작성 및 조회) |
| `news_table` | `id`, `title`, `content`, `date` | 뉴스성 게시물 관리 |
| `notices` | `id`, `title`, `content`, `created_at` | 공지사항 관리 |
| `notices_files` | `id`, `notice_id`, `file_path` | 공지사항 첨부파일 관리 |

# 2. 게시판 기능 상세

- **테이블:** `board`
- **테스트 데이터 예시:**

| id | title | content | author_id | created_at | views_count | user |
|---|---|---|---|---|---|---|
| 4 | 테스트입니다. | 테스트입니다. | comhjj1127 | 2025-06-06 19:22:19 | 0 | user |

- **기능 흐름:**
  1. 사용자가 로그인(교수/학생 구분)
  2. 게시판 페이지(`/posts`) 진입
  3. “작성하기” 버튼 클릭 시 게시글 입력 폼 호출
  4. `INSERT INTO board (...) VALUES (...)` 실행 → DB 저장
  5. 메인 게시판 목록에서 최신 글 자동 갱신
- **SQL 예시:**

```
CREATE TABLE board (
  id SERIAL PRIMARY KEY,
```

```
  title VARCHAR(255) NOT NULL,
  content TEXT,
  author_id VARCHAR(100),
  created_at TIMESTAMP DEFAULT NOW(),
  views_count INTEGER DEFAULT 0
);
```

- **조회 예시 쿼리:**

```
SELECT title, content, author_id, created_at
FROM board
ORDER BY created_at DESC;
```

---

## 3. 권한(Role) 기반 접근 제어

| 역할 | 기능 | 접근 가능 |
|---|---|---|
| **교수(admin)** | 전체 게시글 관리, 공지사항 등록/삭제 | ✅ |
| **학생(user)** | 일반 게시글 작성/조회/수정(본인 글만) | ✅ |
| **비회원** | 공지사항 및 메인 페이지 조회 | 🔷 제한적 |

- 로그인 시 `users.role` 값을 통해 접근 권한 분리
- 백엔드(API) 단에서 JWT 토큰 검증 후 `role` 확인

---

## 4. 데이터 연동 및 배포 환경

| 구분 | 서비스 | 내용 |
|---|---|---|
| **Frontend** | Netlify | React 기반, `/posts`, `/login` 라우팅 구성 |
| **Backend** | Render | Node.js(Express) 서버, REST API 구성 |
| **Database** | Neon(PostgreSQL) | Cloud 기반 영구 저장, `sslmode=require` 설정 |
| **Source Control** | GitHub | `ps-club/club-activity-report-2025` 저장소에 버전 관리 |

---

# 5. 한계점

현재 개발된 동아리 웹 보고서 시스템은 **React(프론트엔드)**, **Node.js(백엔드)**, **PostgreSQL(NeonDB)**로 구성되어 있으며,

배포 환경은 **Netlify(프론트엔드)**, **Render(백엔드)**, **NeonDB(데이터베이스)**의 **클라우드 무료 플랜**을 이용하고 있습니다.

그러나 개인 단위로 서버를 직접 운영하는 것은 고려해야 할 사항이 많고,

프론트엔드·백엔드·데이터베이스 등 다양한 인프라 요소를 동시에 관리하기 위해서는

보안, 서버 자원 관리, SSL 인증, 배포 자동화 등 폭넓은 지식이 요구됩니다.

무료 플랜의 경우 다음과 같은 **운영 한계**가 존재합니다.

| 구분 | 문제점 | 영향 |
|---|---|---|
| **Render 서버** | 20분간 요청이 없을 시 인스턴스 자동 슬립(다운) | 최초 접속 시 지연 발생 |
| **Netlify/Render 무료 트래픽 한도** | 일정 요청 수 이상 시 차단 또는 속도 저하 | 다수 사용 시 사이트 응답 지연 |
| **NeonDB 연결 제한** | 동시 접속 세션 수 제한 | 다중 사용자 환경에서 DB 연결 실패 가능 |
| **서버 모니터링 미비** | 자원 사용량 확인 도구 제한적 | 장애 발생 시 즉각 대응 어려움 |

따라서, **지속적이고 안정적인 서비스 운영**을 위해서는

무료 플랜 대신 **유료 클라우드 배포 서비스(AWS, Render Pro, Neon Pro 등)**를 활용하는 것이 필요합니다.

특히 교수님이 요청하신 **"동아리 활동 보고서 자동화 웹 시스템"**을 완전한 형태로 구축·운영하기 위해서는

- 지속 가동 가능한 백엔드 서버 (Render 유료 플랜 이상)
- 자동 백업 및 트래픽 모니터링 기능
- 안정적인 데이터베이스 인스턴스 (Neon 유료 플랜 또는 AWS RDS)
- 정적 호스팅의 빠른 응답 속도 확보 (Netlify Pro 또는 AWS S3+CloudFront)

  가 필수적이라고 판단됩니다.

요약하자면,

> 개인이 모든 서버를 직접 운영하기보다, 클라우드 서비스를 적극적으로 활용하여
> 유지보수 부담을 줄이고 서비스의 안정성을 확보하는 것이 장기적으로 더 효율적입니다.