컴파일룸 개인 테스트

"컴파일룸" 프로젝트 최종 계획서 (MacBook 개발 환경 기준)

1. 프로젝트 개요

- **프로젝트명:** 컴파일룸 (Compile-Room)
- **목표:** 학과 자료(강의, 과제, 족보)를 공유하는 웹 아카이브 구축
- **핵심 기술: Next.js 풀스택(Full-stack)**. 하나의 프로젝트에서 프론트엔드(React)와 백엔드(API)를 동시에 개발.
- **본 계획서의 목적:** 4인의 팀원이 각자의 **MacBook**에서 개발 환경을 통일하고, GitHub로 협업하며, 최종적으로 Ubuntu 서버에 배포하는 전 과정을 명확히 정의한다.

2. MacBook 개발 환경 세팅 (4인 팀원 모두)

팀원 4명 모두가 각자의 MacBook에 다음 환경을 준비해야 합니다. 이것이 우리의 "개발 도구"입니다.

1. **텍스트 에디터 (필수)**
   - VS Code: 현재 가장 표준적인 코드 에디터. (설치 필수)
2. **버전 관리 (필수)**
   - **Git**: MacBook에는 기본 설치되어 있을 수 있습니다. 터미널에서 `git --version` 으로 확인.
   - GitHub Desktop **(권장)**: 4인 협업 시 Git 명령어가 익숙하지 않다면, GUI 툴을 쓰는 것이 충돌을 줄이고 매우 편리합니다.
3. **개발 언어 및 환경 (필수)**
   - Node.js: Next.js를 실행하기 위한 JavaScript 런타임. (LTS 버전 설치)
   - MacBook에서는 **Homebrew** (`brew install node`)로 설치하는 것을 가장 권장합니다.

[중요] Nginx와 PM2는 MacBook에 설치할 필요가 없습니다. 이 두 도구는 개발이 다 끝난 후, 최종 'Ubuntu 서버'에만 설치하는 것입니다. MacBook에서는 `npm run dev` 명령어가 이 모든 것을 대신합니다.

3. 우리가 구현해야 할 것 (개발 To-Do List)

프로젝트를 크게 3개의 파트(UI, DB, API)로 나눕니다. 4인조가 이 작업들을 나눠서 진행할 수 있습니다.

A. 프론트엔드 (UI / 손님용 메뉴판)

- [ ] **Next.js 프로젝트 생성:**
  - `npx create-next-app@latest --tailwindcss` 명령어로 Next.js + Tailwind CSS 프로젝트를 생성합니다. (조장 1명만 생성 후 GitHub에 `push`)
- [ ] **기존 HTML → React 컴포넌트 변환:**
  - [ ] `컴파일룸_메인_페이지.html` → `app/page.js`
  - [ ] `컴파일룸_자료_업로드_페이지.html` → `app/upload/page.js`
  - [ ] `컴파일룸_자료_상세_페이지.html` → `app/materials/[id]/page.js` (동적 라우트)
- [ ] **공통 UI 분리:**
  - [ ] `Header` (헤더/네비게이션) 컴포넌트
  - [ ] `Footer` (푸터) 컴포넌트
  - [ ] `app/layout.js`에 공통 레이아웃 적용

B. 데이터베이스 설계 (주방 레시피북)

- [ ] **Prisma + SQLite 설치 및 설정:**
  - `npm install prisma` / `npx prisma init --datasource-provider sqlite`
- [ ] **데이터 모델 정의 (`prisma/schema.prisma` 파일):**
  - [ ] **`User` 모델:** (Auth.js와 연동) 사용자 정보 (이름, 이메일, 프로필 사진 등)
  - [ ] **`Material` 모델:** (게시글)
    - `id` (고유번호), `title` (제목), `description` (내용)
    - `subject` (과목), `type` (자료 종류)
    - `fileUrl` (★실제 파일 경로), `fileName` (파일명)
    - `authorId` (작성자 ID, User와 연결)
    - `createdAt` (작성일)
  - [ ] **`Comment` 모델:** (댓글)
    - `id`, `content` (내용)
    - `authorId` (작성자 ID, User와 연결)
    - `materialId` (게시글 ID, Material과 연결)
- [ ] **DB 파일 생성:**
  - `npx prisma migrate dev --name init` 명령어로 `prisma/dev.db` (SQLite 파일) 생성.
  - **(중요)** `.gitignore` 파일에 `*.db`를 추가하여 DB 파일이 GitHub에 올라가지 않도록 합니다. (각자의 Mac에서 테스트용 DB를 가짐)

C. 백엔드 API 구현 (주방 / 데이터 처리 창구)

Next.js의 Route Handlers (`app/api/.../route.js`)를 사용해 "내부 주방"을 만듭니다.

- [ ] **인증 API 구현:**
  - [ ] `npm install auth` (Auth.js 설치)
  - [ ] `Prisma Adapter`와 연동하여, 로그인/회원가입 시 `User` 모델(DB)에 자동 저장되도록 설정.
- [ ] **(핵심) 자료 업로드 API (`app/api/upload/route.js`):**
  - [흐름 1] 프론트엔드(`app/upload/page.js`)의 폼 데이터(FormData)를 받습니다.
  - [흐름 2] **파일 저장:** 받은 파일(file)을 MacBook의 로컬 폴더(예: `public/uploads/`)에 저장합니다.
  - [흐름 3] **DB 저장:** 파일 외 텍스트(제목, 설명)와 [흐름 2]에서 저장된 파일의 경로(`fileUrl`)를 `Prisma`를 이용해 `Material` 모델(DB)에 저장합니다.
- [ ] **자료 목록/검색 API (`app/api/materials/route.js`):**
  - [흐름] 메인 페이지에서 요청 시, `Prisma`를 사용해 `Material` 목록을 최신순/인기순으로 DB에서 조회하여 JSON 형태로 반환합니다.
- [ ] **댓글 API (`app/api/comments/route.js`):**
  - [흐름] 상세 페이지에서 요청 시, 특정 `materialId`에 해당하는 댓글을 DB에 추가하거나, DB에서 조회하여 반환합니다.

4. 4인 팀 협업 흐름 (MacBook ↔ GitHub)

이것이 4인조가 매일 개발할 때의 흐름입니다.

1. **(팀원 A)** 작업 시작 전:
   - `git pull` (GitHub에서 최신 코드를 내 Mac으로 받기)
   - `npm install` (다른 팀원이 추가한 라이브러리 설치)
2. **(팀원 A)** 로컬 서버 실행:
   - `npm run dev` (터미널에 입력)
   - 브라우저에서 `http://localhost:3000` 열기 (내 Mac에서만 보이는 테스트 서버)
3. **(팀원 A)** 코드 수정:
   - `app/upload/page.js`의 UI를 수정하거나, `app/api/upload/route.js`의 기능을 개발.
   - `http://localhost:3000/upload`에서 직접 테스트.
4. **(팀원 A)** 작업 완료 후:

- `git add .` (수정한 파일 추가)
- `git commit -m "feat: 자료 업로드 UI 완성"` (작업 내용 기록)
- `git push` (내 Mac의 변경사항을 GitHub로 올리기)

5. **(팀원 B, C, D):**
   - 1번부터 다시 반복... (팀원 A가 올린 코드를 `git pull`로 받음)

5. 최종 서버 배포 (MacBook → Ubuntu)

이 단계는 위 1~4단계가 모두 끝나고, MacBook의 `http://localhost:3000`에서 모든 기능이 완벽하게 작동할 때 **딱 한 번만** 진행합니다.

1. **Ubuntu 서버 접속** (`ssh` 이용)
2. **서버 환경 구축:** `sudo apt install nodejs npm nginx`, `sudo npm install -g pm2`
3. **프로젝트 복제:** `git clone [우리 GitHub 리포지토리 주소] /var/www/compileroom`
4. **프로젝트 세팅:**
   - `cd /var/www/compileroom`
   - `npm install`
   - `npx prisma migrate deploy` (★개발용이 아닌, 배포용 DB 생성)
   - `npm run build` (Next.js 프로젝트 빌드)
5. **Nginx 설정:** (리버스 프록시) `/etc/nginx/sites-available/`에 설정 파일 추가.
6. **PM2로 서버 실행:**
   - `pm2 start npm --name compileroom -- run start`
   - 이제 `http://[서버 IP 또는 도메인]`으로 실제 서비스가 시작됩니다.