AI 관리형 광고 운영 시스템 - 프로젝트 전체 구조 및 기능 개요

> **문서 목적**: 이 문서는 프로젝트를 처음 인수받는 개발자가 전체 시스템의 구조, 책임 분리, 데이터 흐름을 빠르게 이해할 수 있도록 작성되었습니다.

---

1. 프로젝트 전체 개요

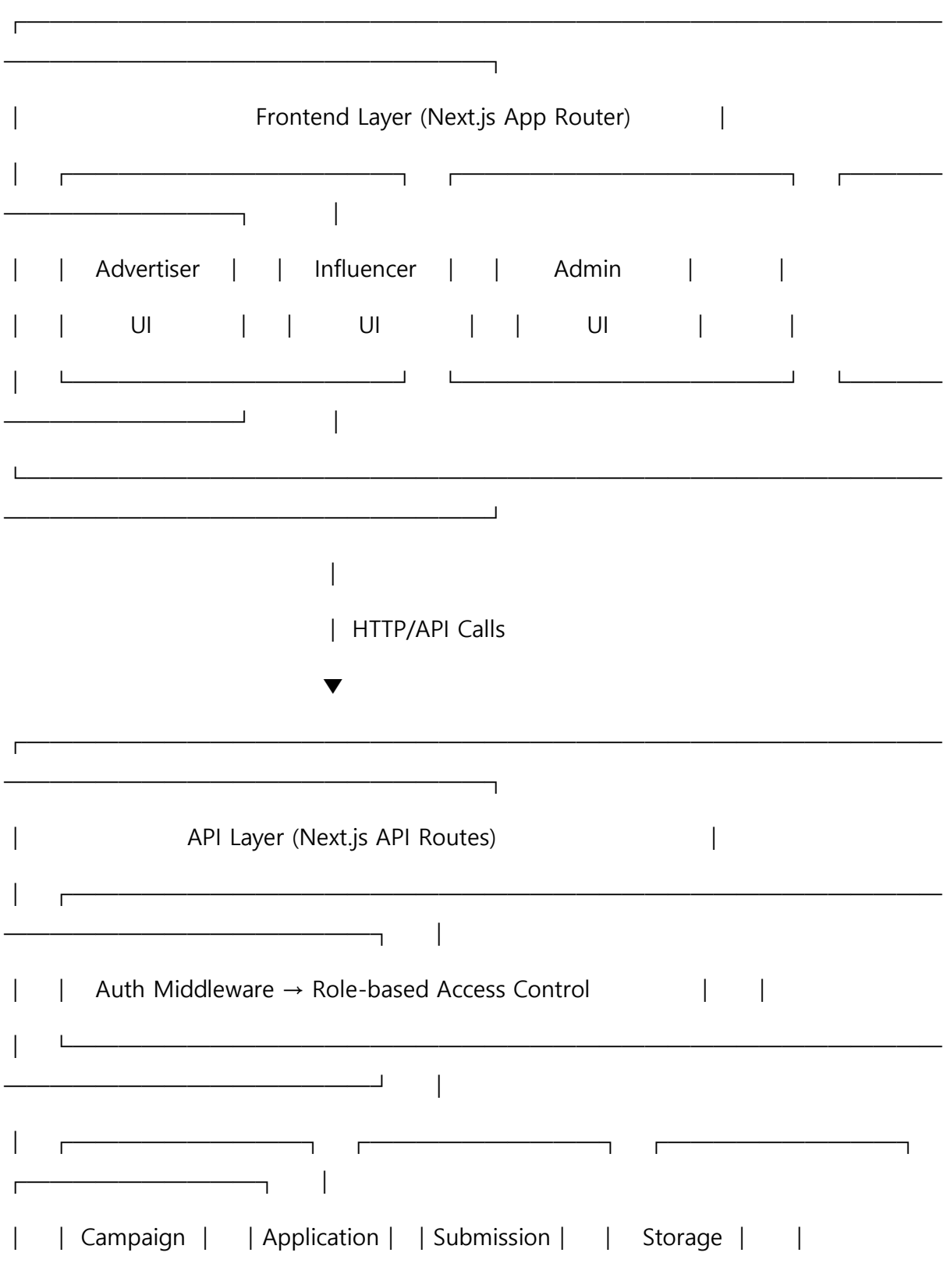
1.1 프로젝트 목적

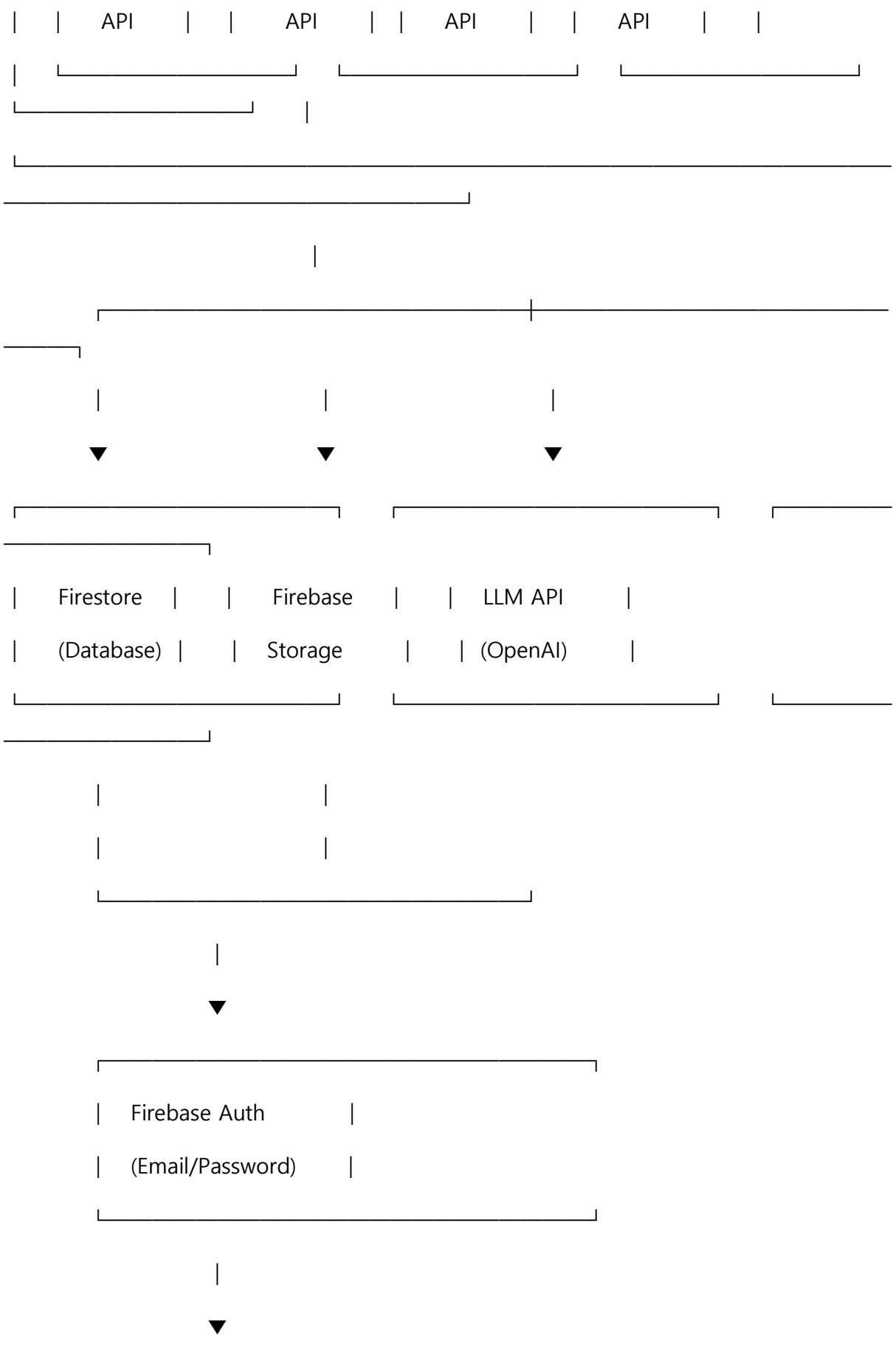
AI 관리형 광고 운영 시스템은 자연어 입력만으로 광고 캠페인을 생성하고, LLM이 실행 가능한 기획서를 자동 생성하며, 플랫폼이 모집/선정/집행/증빙/리포트까지 전 과정을 자동화하여 운영하는 시스템입니다.

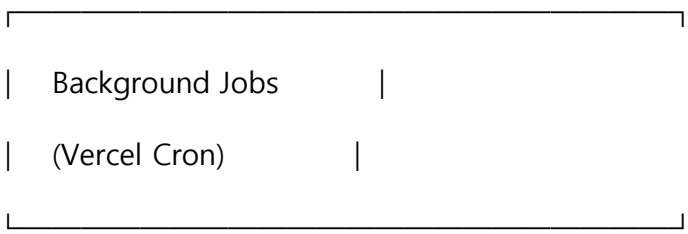
핵심 가치 제안:

- 광고 전문 지식 없이도 자연어로 캠페인 요청 가능
- AI가 전문적인 광고 기획서 자동 생성
- 광고주-인플루언서 간 직접 소통 차단 (플랫폼 중심 운영)
- 역할 기반 접근 제어 (광고주/인플루언서/운영자)

1.2 시스템 아키텍처 개요




API
API
API
API
Firestore
(Database)
Firebase
Storage
LLM API
(OpenAI)
Firebase Auth
(Email/Password)




```

### 1.3 레이어별 역할 요약

#### Frontend Layer (Next.js App Router)

- **책임**: 사용자 인터페이스 렌더링, 클라이언트 사이드 상태 관리

- **기술**: React 18, Next.js 14 App Router, TypeScript, Tailwind CSS

- **특징**: 역할별 Route Groups로 UI 분리, Server/Client Components 혼용

#### API Layer (Next.js Route Handlers)

- **책임**: 인증/인가, 비즈니스 로직 처리, 외부 서비스 연동

- **기술**: Next.js API Routes, Firebase Admin SDK

- **특징**: 미들웨어 기반 인증, 역할 기반 접근 제어, 통일된 에러 응답

#### Data Layer (Firebase Services)

- **Firestore**: 구조화된 데이터 저장 (campaigns, users, applications, submissions, events)

- **Firebase Storage**: 증빙 파일 업로드 (Signed URL 방식)

- **Firebase Auth**: 사용자 인증 및 역할 관리

#### AI/External Services
```

- **OpenAI GPT-4**: 자연어 → 구조화된 캠페인 스펙 변환
- **Vercel Cron**: 스케줄링된 백그라운드 작업 (리마인더, 리포트 생성, 상태 전환)

---

2. 폴더 단위 구조화

2.1 루트 디렉토리

```
ads/
├── .env.local                # 환경변수 (로컬 개발용, Git 무시)
├── .gitignore                # Git 무시 파일 목록
├── package.json              # 프로젝트 의존성 및 스크립트
├── tsconfig.json             # TypeScript 설정
├── next.config.js            # Next.js 설정 (이미지 도메인 등)
├── tailwind.config.ts        # Tailwind CSS 설정
├── firebase.json             # Firebase CLI 설정
├── firestore.rules           # Firestore 보안 규칙
├── firestore.indexes.json    # Firestore 인덱스 정의
├── vercel.json               # Vercel 배포 설정 (Cron Jobs 포함)
├── public/                   # 정적 파일 (이미지 등)
├── src/                      # 소스 코드
├── docs/                     # 프로젝트 문서
└── README.md                 # 프로젝트 개요 및 시작 가이드
```

```

**주요 설정 파일 설명**:

- `vercel.json`: Vercel Cron Jobs 스케줄 정의 (매일 리마인더, 매시간 상태 전환 등)

- `firestore.rules`: 현재 모든 접근 차단 (보안 규칙 미구현 상태 - 주의 필요)

- `next.config.js`: Firebase Storage 이미지 도메인 허용 설정

---

### 2.2 `/src` - 소스 코드 루트

#### `/src/app` - Next.js App Router (페이지 및 API)

**책임**:

- 사용자 인터페이스 페이지 렌더링

- API 엔드포인트 제공

- 라우팅 및 레이아웃 관리

**구조**:

```
src/app/
 ├─── layout.tsx   # 루트 레이아웃 (전역 메타데이터, 폰트 설정)
 ├─── page.tsx   # 랜딩 페이지 (로그인 체크 후 리다이렉트)
 ├─── globals.css   # 전역 스타일 (Tailwind CSS, CSS 변수)
 |
```
```

```
├── (auth)/                    # 인증 관련 페이지 (Route Group)
│   ├── login/page.tsx         # 로그인 페이지
│   └── signup/page.tsx        # 회원가입 페이지
│
├── (advertiser)/              # 광고주 전용 페이지 (Route Group)
│   ├── layout.tsx             # 광고주 레이아웃 (네비게이션 포함)
│   └── campaigns/
│       ├── new/page.tsx       # 새 캠페인 생성 (자연어 입력)
│       └── [id]/
│           └── clarify/       # 확인 질문 답변 (빈 폴더 - 미구현)
│
├── (influencer)/              # 인플루언서 전용 페이지 (Route Group)
│   └── layout.tsx             # 인플루언서 레이아웃
│
├── admin/                     # 운영자 페이지 (Route Group 아님)
│   ├── layout.tsx             # 운영자 레이아웃
│   ├── dashboard/page.tsx     # 전체 캠페인 모니터링 대시보드
│   └── campaigns/
│       └── [id]/page.tsx      # 캠페인 상세 관리
│
├── campaigns/                 # 공통 캠페인 페이지
│   ├── layout.tsx             # 캠페인 공통 레이아웃
│   ├── page.tsx               # 캠페인 리스트 (역할별 다른 뷰)
│   └── [id]/
│       ├── page.tsx           # 캠페인 상세 보기
```

```
|           └─── review/page.tsx # 제안서 검토 페이지
|
└─── api/                        # API Route Handlers
    ├─── auth/
    |    ├─── me/route.ts       # 현재 사용자 정보 조회
    |    └─── signup/route.ts # 회원가입 처리
    |
    ├─── campaigns/
    |    ├─── route.ts          # GET: 캠페인 리스트, POST: 캠페인 생성
    |    ├─── generate/route.ts # POST: LLM으로 캠페인 생성
    |    ├─── open/route.ts     # GET: 오픈 캠페인 리스트 (인플루언서용)
    |    └─── [id]/
    |         ├─── route.ts     # GET: 캠페인 상세, PATCH: 캠페인 수정
    |         ├─── approve/route.ts # POST: 캠페인 승인
    |         ├─── applications/
    |         |    ├─── route.ts # GET: 지원 목록, POST: 지원하기
    |         |    └─── [appId]/select/route.ts # POST: 인플루언서 선정
    |         └─── submissions/
    |              ├─── route.ts # GET: 제출 목록, POST: 증빙 제출
    |              └─── [subId]/review/route.ts # POST: 제출 검토
    |
    ├─── cron/                   # 백그라운드 작업 (Vercel Cron)
    |    ├─── deadline-reminder/route.ts     # 매일 9시: 마감 리마인더
    |    ├─── overdue-detection/route.ts     # 매일 9시 5분: 지연 감지
    |    ├─── generate-reports/route.ts      # 매일 18시: 리포트 생성
```

```
|       └──── status-transition/route.ts        # 매시간: 상태 자동 전환
|
└──── storage/
        └──── upload/route.ts # POST: 파일 업로드 Signed URL 생성
```

Route Groups 설명:

- `(auth)`, `(advertiser)`, `(influencer)`: Next.js Route Groups로 URL에는 포함되지 않지만 레이아웃을 분리
- `admin/`: Route Group이 아니므로 `/admin/*` URL로 접근

API Routes 패턴:

- RESTful 구조: `/api/campaigns/{id}/applications/{appId}/select`
- 모든 API는 인증 미들웨어를 통해 토큰 검증
- 역할 기반 접근 제어 (`requireRole` 미들웨어)

`/src/components` - React 컴포넌트

책임: 재사용 가능한 UI 컴포넌트 제공

구조:

```
src/components/
```

```
├─── ui/                          # shadcn/ui 기반 기본 UI 컴포넌트
|      ├─── button.tsx             # 버튼 컴포넌트 (variant, size 지원)
|      ├─── card.tsx               # 카드 컴포넌트
|      ├─── input.tsx              # 입력 필드
|      ├─── textarea.tsx           # 텍스트 영역
|      └─── badge.tsx              # 뱃지 컴포넌트
|
└─── campaigns/
     └─── CampaignsList.tsx     # 캠페인 리스트 컴포넌트 (역할별 다른 뷰)
```

특징:
- `ui/`: Radix UI 기반 접근성 있는 컴포넌트
- `campaigns/`: 도메인별 비즈니스 컴포넌트
- 현재 컴포넌트 구조가 단순함 (확장 필요 시 도메인별 폴더 분리 권장)

`/src/lib` - 공통 라이브러리 및 유틸리티

책임: 비즈니스 로직, 외부 서비스 연동, 공통 기능 제공

구조:
```
src/lib/
```

```
├─── firebase/
|      ├─── admin.ts              # Firebase Admin SDK 초기화 및 싱글톤
|      |                          # - getAdminApp(): Firebase App 인스턴스
|      |                          # - getAdminFirestore(): Firestore 인스턴스
|      |                          # - getAdminAuth(): Auth 인스턴스
|      |                          # - getAdminStorage(): Storage 인스턴스
|      |
|      ├─── auth.ts               # 클라이언트 사이드 Firebase Auth
|      |                          # - getFirebaseAuth(): 클라이언트 Auth 인스턴스
|      |
|      └─── firestore.ts          # Firestore 헬퍼 함수
|                                 # - createCampaign(): 캠페인 문서 생성
|                                 # - createCampaignSpec(): 스펙 버전 생성
|                                 # - updateCampaign(): 캠페인 업데이트
|                                 # - createEvent(): 이벤트 기록
|
├─── auth/
|      ├─── middleware.ts         # API 인증/인가 미들웨어
|      |                          # - verifyAuth(): Firebase 토큰 검증
|      |                          # - requireRole(): 역할 기반 접근 제어
|      |                          # - verifyCronSecret(): Cron 작업 인증
|      |
|      └─── roles.ts              # 역할 타입 정의 및 상수
|
├─── llm/
```

```
|   ├── client.ts              # LLM API 클라이언트 (OpenAI)
|   |                          # - generateCampaignSpec(): 자연어 → 캠페인 스펙
|   |                          # - 재시도 로직 포함 (최대 3회)
|   |
|   ├── prompts.ts             # LLM 프롬프트 템플릿
|   |                          # - SYSTEM_PROMPT: 시스템 프롬프트
|   |                          # - getUserPrompt(): 사용자 입력 프롬프트
|   |                          # - getRetryPrompt(): 재시도 프롬프트
|   |
|   └── schema.ts              # Zod 스키마 (LLM 응답 검증)
|                              # - LLMResponseSchema: JSON 응답 스키마
|
└── utils/
    ├── cn.ts                  # className 유틸리티 (Tailwind merge)
    └── constants.ts           # 상수 정의
                               # - CAMPAIGN_STATUS_LABELS: 상태 라벨
                               # - APPLICATION_STATUS_LABELS
                               # - SUBMISSION_STATUS_LABELS
```

핵심 파일 상세:

`lib/firebase/admin.ts`:
- Firebase Admin SDK 싱글톤 패턴으로 초기화
- 환경변수에서 서비스 계정 키 읽기

- 모든 서버 사이드 Firebase 작업의 진입점

`lib/auth/middleware.ts`:
- 모든 API Route의 첫 번째 진입점
- Authorization 헤더에서 Bearer 토큰 추출
- Firebase Admin Auth로 토큰 검증
- Firestore users 컬렉션에서 역할 정보 조회
- 역할 기반 접근 제어 (`requireRole`)

`lib/llm/client.ts`:
- OpenAI GPT-4 Turbo 사용
- JSON 모드로 응답 받기
- Zod 스키마로 응답 검증
- 실패 시 최대 3회 재시도 (에러 메시지 포함)

`/src/types` - TypeScript 타입 정의

책임: 도메인 모델 및 타입 정의 중앙화

구조:
```
src/types/
├── index.ts                    # 타입 재export (UserRole 등)
```

```
├─── user.ts                    # 사용자 타입
├─── campaign.ts                 # 캠페인 관련 타입
│                                # - CampaignStatus: 상태 enum
│                                # - Campaign: 캠페인 엔티티
│                                # - CampaignSpec: LLM 생성 스펙
│                                # - CampaignSpecVersion: 스펙 버전 관리
│
├─── application.ts             # 지원 관련 타입
├─── submission.ts               # 제출 관련 타입
└─── ...
```

특징:
- Firestore 문서 타입과 클라이언트 타입 분리 (`CampaignDocument` vs `Campaign`)
- Timestamp 변환 로직은 API 레이어에서 처리

2.3 `/public` - 정적 파일

```
public/
└─── images/
     ├─── hero.jpg               # 랜딩 페이지 히어로 이미지
     ├─── platform.jpg           # 플랫폼 소개 섹션 이미지
```

```
    └──── cta.jpg                  # CTA 섹션 이미지
```

접근 방법: `/images/hero.jpg` (Next.js가 자동으로 `/public`을 루트로 매핑)

2.4 `/docs` - 프로젝트 문서

```
docs/
├──── ARCHITECTURE.md             # 아키텍처 설계 문서
├──── FOLDER_STRUCTURE.md         # 폴더 구조 상세
├──── FIRESTORE_SCHEMA.md         # Firestore 스키마
├──── API_SPEC.md                 # API 명세서
├──── LLM_PROMPT_DESIGN.md        # LLM 프롬프트 설계
├──── BACKGROUND_JOBS.md          # 백그라운드 작업 설계
├──── AUTH_FLOW.md                # 인증 플로우
└──── PROJECT_PROGRESS.md         # 프로젝트 진행 상황
```

---

3. 핵심 흐름 설명

3.1 캠페인 생성 플로우 (자연어 → AI 기획서)

```
1. 사용자 액션
    └──> (advertiser)/campaigns/new/page.tsx
        └──> 사용자가 자연어 입력 후 "생성" 버튼 클릭

2. API 호출
    └──> POST /api/campaigns/generate
        └──> Body: { naturalLanguageInput: "..." }

3. 인증/인가
    └──> lib/auth/middleware.ts::requireRole(['advertiser'])
        └──> Authorization 헤더에서 토큰 추출
        └──> Firebase Admin Auth로 토큰 검증
        └──> Firestore users/{uid}에서 role 확인
        └──> role이 'advertiser'인지 검증

4. LLM 호출
    └──> lib/llm/client.ts::generateCampaignSpec()
        └──> lib/llm/prompts.ts::getUserPrompt()로 프롬프트 생성
        └──> OpenAI GPT-4 Turbo API 호출
        └──> JSON 응답 파싱
        └──> lib/llm/schema.ts::LLMResponseSchema로 검증
        └──> 실패 시 최대 3회 재시도
```

5. 데이터 저장

    └──> lib/firebase/firestore.ts::createCampaign()

        └──> Firestore campaigns/{id} 생성 (status: 'GENERATED')

    └──> lib/firebase/firestore.ts::createCampaignSpec()

        └──> Firestore campaigns/{id}/specs/{versionId} 생성

    └──> lib/firebase/firestore.ts::updateCampaign()

        └──> 제목, 마감일 등 업데이트

    └──> lib/firebase/firestore.ts::createEvent()

        └──> events/{eventId} 생성 (audit trail)

6. 응답 반환

    └──> { success: true, data: { campaignId, proposalMarkdown, spec, status } }

7. 프론트엔드 처리

    └──> 제안서 마크다운 렌더링

    └──> 확인 질문 표시

    └──> 승인/수정 옵션 제공

```

**주요 포인트**:

- LLM 응답은 Zod 스키마로 검증하여 타입 안정성 보장

- 스펙 버전 관리로 이력 추적 가능

- 모든 상태 변경은 events 컬렉션에 기록 (audit trail)
```

3.2 인플루언서 지원 플로우

```
1. 인플루언서 액션
    └──> campaigns/page.tsx (CampaignsList 컴포넌트)
        └──> "오픈 캠페인" 목록 조회

2. API 호출
    └──> GET /api/campaigns/open
        └──> Authorization: Bearer {token}

3. 인증/인가
    └──> lib/auth/middleware.ts::requireRole(['influencer'])
        └──> role이 'influencer'인지 검증

4. 데이터 조회
    └──> Firestore campaigns 컬렉션 쿼리
        └──> where('status', '==', 'OPEN')
        └──> where('openedAt', '<=', now())
        └──> orderBy('openedAt', 'desc')

5. 지원하기
    └──> POST /api/campaigns/{id}/applications
```

```
        └──> Body: { message: "..." }
        └──> Firestore campaigns/{id}/applications/{appId} 생성
        └──> status: 'APPLIED'

6. 광고주 선정
   └──> POST /api/campaigns/{id}/applications/{appId}/select
        └──> requireRole(['advertiser'])
        └──> application.status = 'SELECTED'
        └──> campaign.selectedInfluencerIds에 추가
        └──> campaign.status = 'MATCHING' → 'RUNNING'
```

3.3 백그라운드 작업 플로우 (Cron Jobs)

```
1. Vercel Cron 트리거
   └──> vercel.json에 정의된 스케줄에 따라 자동 호출
        └──> 예: "0 9 * * *" (매일 9시 UTC)

2. Cron 인증
   └──> lib/auth/middleware.ts::verifyCronSecret()
        └──> x-cron-secret 헤더와 CRON_SECRET 환경변수 비교
```

3. 작업 실행

    └──> /api/cron/deadline-reminder/route.ts

        └──> Firestore campaigns 쿼리

        └──> where('deadlineDate', '<=', tomorrow)

        └──> where('status', 'in', ['OPEN', 'MATCHING', 'RUNNING'])

        └──> 각 캠페인에 대해 알림 생성 또는 이메일 발송

4. 상태 전환

    └──> /api/cron/status-transition/route.ts

        └──> 매시간 실행

        └──> 조건에 따라 자동 상태 전환

            └──> APPROVED + openedAt <= now() → OPEN

            └──> RUNNING + deadlineDate < now() → COMPLETED

```

---

## 4. 기술 스택 및 의존성 요약

### 4.1 프론트엔드

| 기술 | 버전 | 용도 |

|---|---|---|

| Next.js | 14.0.4 | React 프레임워크 (App Router) |

| React | 18.2.0 | UI 라이브러리 |
```

| TypeScript | 5.3.3 | 타입 안정성 |

| Tailwind CSS | 3.3.6 | 유틸리티 기반 CSS |

| Radix UI | ^1.0.x | 접근성 있는 UI 컴포넌트 |

| Lucide React | ^0.294.0 | 아이콘 라이브러리 |

4.2 백엔드

| 기술 | 버전 | 용도 |

|---|---|---|

| Next.js API Routes | 14.0.4 | 서버 사이드 API 엔드포인트 |

| Firebase Admin SDK | 12.0.0 | 서버 사이드 Firebase 연동 |

| Firebase Client SDK | 10.7.1 | 클라이언트 사이드 Firebase 연동 |

4.3 데이터 계층

| 서비스 | 용도 |

|---|---|

| Firebase Firestore | 구조화된 데이터 저장 (NoSQL) |

| Firebase Storage | 파일 저장 (증빙 이미지 등) |

| Firebase Authentication | 사용자 인증 (Email/Password) |

4.4 AI/External Services

| 서비스 | 용도 |

|---|---|

| OpenAI GPT-4 Turbo | 자연어 → 구조화된 캠페인 스펙 변환 |

| Vercel Cron | 스케줄링된 백그라운드 작업 |

4.5 개발 도구

| 도구 | 용도 |

|------|------|

| ESLint | 코드 품질 검사 |

| TypeScript | 타입 체크 (`tsc --noEmit`) |

| Zod | 런타임 스키마 검증 (LLM 응답) |

4.6 배포/인프라

| 서비스 | 용도 |

|--------|------|

| Vercel | 프론트엔드/API 배포, Cron Jobs |

| Firebase | 데이터베이스, 스토리지, 인증 |

---

5. 현재 구조의 특징

5.1 잘 분리된 부분

✅ **인증/인가 레이어**

- `lib/auth/middleware.ts`에서 중앙화된 인증 로직
- 역할 기반 접근 제어가 일관되게 적용
- API Route에서 재사용 가능한 미들웨어 패턴

✅ **Firebase 연동**
- `lib/firebase/`에서 모든 Firebase 관련 로직 분리
- Admin/Client SDK 분리
- 싱글톤 패턴으로 인스턴스 관리

✅ **LLM 통합**
- `lib/llm/`에서 LLM 관련 로직 분리
- 프롬프트 템플릿 분리 (`prompts.ts`)
- 스키마 검증 분리 (`schema.ts`)

✅ **타입 정의**
- `types/`에서 도메인 모델 중앙화
- Firestore 문서 타입과 클라이언트 타입 분리

✅ **Route Groups 활용**
- 역할별 UI를 Route Groups로 분리
- 레이아웃 중복 제거

5.2 결합도가 높은 부분 (주의 필요)

⚠️ **API Route와 비즈니스 로직 결합**

- 현재 API Route Handler에 비즈니스 로직이 직접 포함
- 예: `/api/campaigns/generate/route.ts`에서 LLM 호출, Firestore 저장을 모두 처리
- **개선 방향**: Service 레이어 도입 고려

  ```
  src/lib/services/
  ├── campaign.service.ts      # 캠페인 생성, 업데이트 로직
  ├── application.service.ts # 지원 처리 로직
  └── submission.service.ts   # 제출 처리 로직
  ```

⚠️ **Firestore 쿼리 로직 분산**

- 각 API Route에서 Firestore 쿼리를 직접 작성
- **개선 방향**: Repository 패턴 도입 고려

  ```
  src/lib/repositories/
  ├── campaign.repository.ts
  ├── application.repository.ts
  └── submission.repository.ts
  ```

⚠️ **에러 처리 일관성**

- API Route마다 에러 응답 형식이 약간씩 다를 수 있음
- **개선 방향**: 공통 에러 핸들러 유틸리티

5.3 확장/리팩토링 시 주의사항

🔴 **Firestore 보안 규칙 미구현**

- `firestore.rules`가 현재 모든 접근을 차단
- **즉시 수정 필요**: 프로덕션 배포 전 보안 규칙 구현 필수

🟡 **상태 관리**

- 현재 클라이언트 사이드 상태 관리는 React hooks만 사용
- 복잡한 상태가 늘어나면 Context API 또는 상태 관리 라이브러리 고려

🟡 **컴포넌트 구조**

- 현재 `components/` 구조가 단순함
- 도메인이 늘어나면 도메인별 폴더 분리 권장

```
components/
├── ui/                  # 기본 UI 컴포넌트
├── campaigns/           # 캠페인 관련 컴포넌트
├── applications/        # 지원 관련 컴포넌트
└── shared/              # 공통 컴포넌트
```

🟡 **테스트**

- 현재 테스트 코드 없음

- API Route, LLM 통합, Firestore 쿼리 등에 대한 테스트 추가 권장

🟡 **환경변수 관리**

- `.env.local` 파일이 Git에 올라가지 않지만

- `.env.example` 파일이 없어서 필요한 환경변수를 파악하기 어려움

- **개선**: `.env.example` 파일 추가 권장

---

6. 데이터 모델 요약

6.1 Firestore 컬렉션 구조

```
users/{uid}
  - email: string
  - role: 'advertiser' | 'influencer' | 'admin'
  - createdAt: Timestamp
  - updatedAt: Timestamp

campaigns/{campaignId}
  - advertiserId: string
  - status: CampaignStatus
  - title: string
```

- naturalLanguageInput?: string
- createdAt: Timestamp
- updatedAt: Timestamp
- approvedAt?: Timestamp
- openedAt?: Timestamp
- completedAt?: Timestamp
- deadlineDate?: Timestamp
- estimatedDuration?: number
- currentSpecVersionId?: string
- selectedInfluencerIds?: string[]

campaigns/{campaignId}/specs/{specVersionId}

- version: number
- proposalMarkdown: string
- specJson: CampaignSpec (구조화된 스펙)
- createdAt: Timestamp
- createdBy: string

campaigns/{campaignId}/applications/{applicationId}

- influencerId: string
- status: 'APPLIED' | 'REJECTED' | 'SELECTED'
- message?: string
- createdAt: Timestamp

campaigns/{campaignId}/submissions/{submissionId}

- influencerId: string
- applicationId: string
- status: 'SUBMITTED' | 'NEEDS_FIX' | 'APPROVED'
- contentUrl?: string
- screenshotUrls?: string[]
- metrics?: object
- createdAt: Timestamp

events/{eventId}
- campaignId: string
- actorId: string
- actorRole: string
- type: string
- payload?: object
- createdAt: Timestamp
```

---

## 7. 보안 고려사항

### 7.1 현재 구현된 보안

✅ **인증**
- 모든 API Route에서 Firebase 토큰 검증
```

- 서버 사이드에서만 토큰 검증 (클라이언트 검증 의존 안 함)

✅ **인가**

- 역할 기반 접근 제어 (`requireRole` 미들웨어)
- 사용자는 자신의 데이터 또는 허가된 데이터만 접근

✅ **환경변수**

- 민감한 정보 (Firebase 서비스 계정 키, OpenAI API 키)는 환경변수로 관리
- `.gitignore`에 포함

7.2 보안 개선 필요 사항

🔴 **Firestore 보안 규칙**

- 현재 모든 접근 차단 상태
- **즉시 구현 필요**: 역할별 읽기/쓰기 규칙 정의

🟡 **API Rate Limiting**

- 현재 LLM API 호출에 대한 Rate Limiting 없음
- 비용 및 악용 방지를 위해 Rate Limiting 추가 권장

🟡 **입력 검증**

- 일부 API에서 입력 검증이 부족할 수 있음
- Zod 스키마로 모든 입력 검증 강화 권장

---

8. 배포 및 운영

8.1 배포 환경

- **프론트엔드/API**: Vercel
- **데이터베이스**: Firebase Firestore
- **스토리지**: Firebase Storage
- **인증**: Firebase Authentication

8.2 환경변수

필수 환경변수 목록:

- `NEXT_PUBLIC_FIREBASE_*`: 클라이언트 Firebase 설정
- `FIREBASE_ADMIN_*`: 서버 사이드 Firebase Admin 설정
- `OPENAI_API_KEY`: OpenAI API 키
- `CRON_SECRET`: Cron 작업 인증 시크릿
- `NEXT_PUBLIC_APP_URL`: 앱 URL

8.3 Cron Jobs

Vercel Cron으로 스케줄링된 작업:

- `deadline-reminder`: 매일 9시 UTC - 마감 리마인더
- `overdue-detection`: 매일 9시 5분 UTC - 지연 감지
- `generate-reports`: 매일 18시 UTC - 리포트 생성

- `status-transition`: 매시간 - 상태 자동 전환

---

9. 개발 워크플로우

9.1 로컬 개발

```bash
# 의존성 설치
npm install

# 환경변수 설정
cp .env.example .env.local  # (현재 .env.example 없음 - 생성 필요)

# 개발 서버 실행
npm run dev

# 타입 체크
npm run type-check

# 린트
npm run lint
```

9.2 브랜치 전략

- `main`: 프로덕션 브랜치
- `develop`: 개발 브랜치
- `features/*`: 기능 개발 브랜치

---

10. 향후 개선 제안

10.1 단기 (즉시 필요)

1. **Firestore 보안 규칙 구현** (최우선)
2. **`.env.example` 파일 생성**
3. **에러 처리 표준화**

10.2 중기 (3-6개월)

1. **Service 레이어 도입** (비즈니스 로직 분리)
2. **Repository 패턴 도입** (데이터 접근 로직 분리)
3. **컴포넌트 구조 개선** (도메인별 폴더 분리)
4. **테스트 코드 추가** (API Route, LLM 통합 등)

10.3 장기 (6개월+)

1. **실시간 업데이트** (Firestore 실시간 리스너 활용)

2. **캐싱 전략** (Next.js 캐싱, Redis 등)

3. **모니터링/로깅** (Sentry, LogRocket 등)

4. **성능 최적화** (이미지 최적화, 코드 스플리팅 등)

부록: 주요 파일 참조

핵심 진입점

- **랜딩 페이지**: `src/app/page.tsx`
- **캠페인 리스트**: `src/components/campaigns/CampaignsList.tsx`
- **캠페인 생성 API**: `src/app/api/campaigns/generate/route.ts`
- **인증 미들웨어**: `src/lib/auth/middleware.ts`
- **LLM 클라이언트**: `src/lib/llm/client.ts`
- **Firebase Admin**: `src/lib/firebase/admin.ts`

설정 파일

- **Next.js 설정**: `next.config.js`
- **Tailwind 설정**: `tailwind.config.ts`
- **TypeScript 설정**: `tsconfig.json`
- **Vercel Cron**: `vercel.json`
- **Firestore 규칙**: `firestore.rules`

문서 작성일: 2025-12-26

작성자: 시니어 엔지니어/아키텍트

문서 버전: 1.0