무서핑 서비스 게시판 및 광고 수익화 기능 PRD

1. 서비스 개요 및 개발 목표

무서핑(Musurfing)은 AI가 생성한 유머/공포 콘텐츠를 제공하는 서비스로, 현재는 **React 기반 프론트엔드**만 Vercel에 배포되어 운영 중입니다. 사용자들은 회원가입 없이도 AI가 만든 귀신 캐릭터 등의 콘텐츠를 감상할 수 있으나, **사용자 참여도와 서비스 체류 시간**을 높일 필요가 있습니다. 따라서 다음 두 가지 신규 기능을 도입하고자 합니다:

- **익명 공개 게시판 기능:** 사용자들이 직접 AI 생성 콘텐츠를 공유하고 소통할 수 있는 커뮤니티 게시판을 추가합니다 (회원가입/로그인 불필요).
- **Google AdSense 광고 수익화:** 서비스 트래픽을 **광고 수익**으로 연결하기 위해 페이지 내 적절한 위치에 배너 광고를 통합합니다.

이 PRD는 위 두 가지 기능에 대한 **상세 요구사항과 구현 방안**을 다루며, 기술적인 타당성과 보완점, 운영 상의 고려사항까지 포괄적으로 제시합니다. 목표는 **기술적으로 구현 가능하고 확장 가능한 솔루션**을 설계하여 서비스 활성화와 수익 창출을 동시에 달성하는 것입니다.

2. 주요 기능 및 사용자 정책

2.1 익명 게시판 (공개 커뮤니티)

- **회원가입 없는 완전 익명성:** 사용자들은 별도의 인증 절차 없이 게시판에서 글을 쓰고 댓글을 달며, 좋아요 등의 피드백을 남길 수 있습니다. 모든 활동은 **익명 닉네임 또는 ID**로 표시되고, 개인 식별 정보(PII)는 수집하지 않습니다.
- **게시글 작성/수정/삭제:** 작성자는 자신이 쓴 글을 일정 시간 내에 수정하거나 삭제할 수 있습니다. 익명성을 유지하면서도 작성자에게만

수정/삭제 권한을 부여하는 메커니즘이 필요합니다 (아래 **5.1 사용자 식별**에서 상세 설명).

- **댓글 및 좋아요:** 사용자들은 게시글에 댓글을 달고, 콘텐츠에 공감 표시(좋아요)를 할 수 있습니다. 이 또한 로그인 없이 가능하지만, **중복 방지**와 **도배 방지**를 위한 기본 장치가 적용됩니다 (예: 동일 기기에서 같은 게시글에 여러 번 좋아요 금지 등).
- **커뮤니티 가이드라인:** 익명 게시판의 특성상 악용 사례를 막기 위한 **커뮤니티 이용 수칙**을 공지합니다. 욕설, 스팸, 불법정보 등의 게시를 금지하며, 위반 시 게시물 삭제 및 이용 제한 조치가 있을 수 있음을 명시합니다.

2.2 Google AdSense 광고 통합

- **광고 게재 위치:** 사용자 경험을 해치지 않으면서도 시선이 잘 머무는 영역에 광고를 배치합니다. 예를 들어:
  - **상단 배너:** 사이트 헤더 아래에 가로 배너 (데스크톱: 728×90, 모바일: 320×50 등).
  - **사이드바 배너:** 데스크톱 화면 우측 사이드바에 고정형 배너(300×600 등) - 모바일에서는 미표시.
  - **피드 내 광고:** 게시글 목록 중간중간 또는 긴 게시글 본문 중간에 **네이티브 광고** 형식으로 삽입 (예: 300×250 Rectangle).
- **반응형 및 자동 광고:** AdSense 의 **반응형 광고 코드**를 사용해 화면 크기에 맞는 광고를 제공하고, 필요에 따라 AdSense 의 **자동 광고(Auto Ads)** 기능을 활용하여 최적의 광고 종류/위치를 Google AI 가 자동으로 시험하도록 할 계획입니다.
- **정책 준수:** 광고가 게재되는 모든 페이지의 **콘텐츠 품질과 정책 적합성**을 유지해야 합니다. **Google 게시자 정책**에 위배되는 콘텐츠(예: 성인물, 혐오, 위험한 또는 불법 요소 등)가 노출될 경우 광고 계정이 제한될 수 있으므로, **UGC(User-Generated Content) 관리 방안**을 철저히 마련합니다 (자세한 내용은 **5.3 콘텐츠 필터링** 참조).
- **수익 목표 및 UX 고려:** 광고를 통한 월 수익 목표를 설정하되, **사용자 경험(UX)**을 해치지 않는 것을 최우선 원칙으로 합니다. 팝업이나 과도한 광고 배치는 지양하며, **콘텐츠 탐색을 방해하지 않는 범위**에서 광고 노출을 최적화합니다. 광고 로딩으로 사이트 속도가 저하되지 않도록 **지연 로딩(Lazy load)**, **비동기 로딩** 등 기술을 사용합니다.

3. 프론트엔드 UI/UX 설계

3.1 기술 스택 및 프레임워크

- **Next.js 14+ (React 기반):** 기존 프론트엔드가 Next.js 로 작성되어 있으므로 게시판도 Next.js 의 **App Router** 구조 하에 개발합니다. Next.js 는 SSR/SSG, ISR(Incremental Static Regeneration) 등을 지원하여 **정적 페이지 생성과 동적 API 라우트** 구현이 모두 가능하므로, 게시판 목록은 정적으로, 게시글 상세보기나 작성 기능은 동적으로 처리하는 식으로 활용합니다【1】.
- **UI 라이브러리: Tailwind CSS** 를 도입하여 빠르게 반응형 디자인을 구축합니다. Tailwind 는 Utility-first CSS 프레임워크로 개발 생산성이 높고, 스타일 일관성을 유지하기 쉽습니다【1】. 컴포넌트 단에서는 필요에 따라 Chakra UI 등의 컴포넌트를 참고하거나, Headless UI 같은 것을 활용할 수 있습니다.
- **상태 관리:** 데이터 패칭에는 React Query 또는 SWR 을 적용하여 **게시글/댓글 목록**을 캐싱하고 실시간 업데이트 시 사용합니다【1】. 전역 상태가 크지 않으므로 Redux 등은 도입하지 않고, React Context 와 훅으로 필요한 상태만 관리합니다.

3.2 UX 요소 및 사용자 편의 기능

- **이미지 업로드 및 미리보기:** 사용자가 AI 생성 이미지를 게시글에 첨부할 수 있도록 **파일 업로드 UI** 를 구현합니다. react-dropzone 같은 라이브러리를 통해 드래그앤드롭 업로드를 지원하며, 업로드 시 **브라우저 FileReader API** 로 즉시 클라이언트 측 미리보기를 제공해 사용자가 올린 이미지가 어떻게 보일지 확인할 수 있습니다【1】. 이는 잘못된 파일 업로드를 줄이고 게시물 품질을 높여줍니다.
- **반응형 디자인:** 모바일 Web 사용자가 많을 것을 고려하여 모든 페이지는 모바일 퍼스트로 디자인합니다. 게시글 목록은 모바일에서는 카드 형식의 단일 컬럼, 데스크톱에선 여러 컬럼 **Grid** 로 보여주고, 사이드바와 큰 배너는 **데스크톱 전용**으로 분기합니다.
- **글 작성 UX:** 게시글/댓글 작성 시 **마크다운** 간이 지원이나 **텍스트 서식 버튼**을 제공하여 사용자들이 콘텐츠를 꾸밀 수 있도록 합니다. (MVP 단계에서는 단순 텍스트+이미지 정도로 시작하고, 추후 에디터 고도화 고려).
- **피드 페이지네이션:** 한 번에 너무 많은 게시글을 불러오지 않도록 페이지네이션 또는 **무한 스크롤(Infinite scroll)**을 적용합니다. Next.js App Router 의 **Dynamic segment** 와 ?page= 파라미터로 페이지네이션 API 를 구현하거나, SWR 의 페이지 기능을 활용할 수 있습니다.
- **로딩/에러 처리:** 데이터 로딩 중에는 Skeleton UI 또는 스피너를 표시하고, 에러 발생 시 사용자에게 이해하기 쉬운 안내 메시지를 보여줍니다. 예: "게시글을 불러오지 못했습니다. 인터넷 연결을 확인해주세요."

4. 백엔드 아키텍처 및 데이터 설계

4.1 서버리스 백엔드 개요

- **Vercel Serverless Functions**: Next.js 의 /app/api 경로를 통해 **API 라우트**들을 구현합니다. Vercel 의 서버리스 함수는 필요 시 **Edge Function** 과 **Node.js(Serverless) Function** 중 선택 가능한데, 짧은 응답이 필요한 간단한 작업(예: 레이트 리미팅 체크)은 Edge 로, 데이터베이스 연동 등 **복잡한 작업**은 Node.js 런타임으로 작성합니다【1】. 이 분리를 통해 **콜드 스타트** 지연을 최소화하고 효율을 높입니다.
    - Edge Functions (V8 런타임): 초기화 지연이 거의 없어 **인증, 레이트 제한, 캐싱 확인** 등의 작업에 사용.
    - Node.js Serverless: DB 쿼리, 파일 업로드, AI 연동 등 **비즈니스 로직** 처리에 사용.
- **데이터베이스**: **Supabase (PostgreSQL)**를 주 데이터베이스로 사용합니다. Supabase 는 Postgres 기반의 서버리스 DB 로 **브라우저 클라이언트 SDK** 및 **REST API** 를 제공하여 Next.js 백엔드와 연동이 수월합니다【1】. 릴레이셔널 DB 이므로 **게시판 스키마**를 구성하기에 적합하며, RLS(Row-Level Security) 등 **권한 통제** 기능도 내장되어 있어 **익명 사용자 권한 관리**에도 활용합니다.
    - *대안:* 아주 단순한 구조의 데이터에는 Vercel KV(Upstash Redis 기반)나 Firebase Firestore 같은 NoSQL 도 고려했으나, **SQL 의 관계 및 쿼리 성능**이 요구되므로 Postgres 가 유리합니다. 또한 **구현의 용이성** 측면에서 검증된 솔루션인 Supabase 를 선택합니다 (만약 서버리스 DB 운영이 복잡해질 경우, 추후 Neon 또는 Planetscale 등으로도 교체 가능).
- **파일 스토리지**: 업로드된 이미지는 **Vercel Blob Storage** 또는 **Supabase Storage** 에 저장합니다【1】. 둘 다 서버리스에 친화적이며 브라우저 업로드를 지원합니다. 저장된 이미지의 URL 을 게시글 데이터에 포함시켜 관리합니다.
    - 향후 트래픽 증가 시를 대비해 **이미지 CDN** 을 설정합니다. Vercel 은 기본적으로 정적 자산에 Global CDN 캐시를 제공하며, 필요시 Cloudflare 같은 외부 CDN 도 연계합니다. 이를 통해 이미지 용량 최적화(WebP 변환, 압축)와 전송 속도를 개선하여 사용자 경험을 향상시킵니다【45】.
- **환경 변수 관리**: 데이터베이스 연결 정보, AdSense Client ID, API 키 등 민감한 값들은 Vercel 의 Environment Variables 에 저장하여 코드에 하드코딩하지 않습니다. CI/CD 배포 시 자동으로 주입됩니다.

4.2 데이터 모델링

게시글(Posts)과 댓글(Comments)을 위한 기본 테이블 구조는 다음과 같습니다 (PostgreSQL 기준):

- **Users (익명 사용자)**: *(선택)* 회원 시스템이 없으므로 필수는 아니지만, 디바이스 단위 익명 사용자 추적을 위해 테이블을 둘 수 있습니다. 예: anonymous_users(id UUID, device_fingerprint TEXT, created_at TIMESTAMP, last_active TIMESTAMP ...). 새 사용자가 활동할 때마다 fingerprint 로 조회/생성.
- **Posts (게시글)**: 게시판의 글 본문을 저장. 주요 컬럼:
  - id (UUID, Primary Key) – 게시글 고유 ID.
  - anonymous_user_id (UUID) – 작성자 익명 ID (Users 테이블 참조).
  - title (TEXT) – 제목 (1~200 자 제약).
  - content (TEXT) – 본문 내용 (최대 10,000 자 등 제약).
  - image_url (TEXT) – 첨부 이미지 경로 (없으면 NULL).
  - created_at (TIMESTAMP) – 생성 시각 (기본값 NOW()).
  - like_count (INT) – 좋아요 수 (기본 0).
  - status (TEXT) – 게시글 상태 (active, deleted, blocked 등).
  - edit_token (TEXT) – 수정 권한 토큰 (랜덤 32 바이트 hex 로 생성).
  - delete_token (TEXT) – 삭제 권한 토큰 (마찬가지로 생성).
- **Comments (댓글)**: 각 게시글에 대한 댓글 목록.
  - id (UUID) – 댓글 ID.
  - post_id (UUID) – 연관된 게시글 ID (Posts FK).
  - anonymous_user_id (UUID) – 작성자 익명 ID.
  - content (TEXT) – 댓글 내용 (제한 길이 설정).
  - created_at (TIMESTAMP).
  - status (TEXT) – 상태 (active, blocked 등).

권한 관리: Supabase 의 RLS(Row Level Security)를 활용하여 **토큰을 이용한 권한 검증** 정책을 설정합니다. 예를 들어, Posts 테이블에 대해 다음과 같은 정책을 적용할 수 있습니다:

```
-- 게시글 수정은 올바른 edit_token 을 가진 경우에만 허용
CREATE POLICY "Allow post edit with valid token"
ON posts FOR UPDATE
USING ( edit_token = current_setting('app.edit_token', true) );
```

```
-- 게시글 삭제는 delete_token 으로 검증
CREATE POLICY "Allow post delete with valid token"
ON posts FOR DELETE
USING ( delete_token = current_setting('app.delete_token', true) );
```

이런 방식으로, 클라이언트나 API 서버가 글 수정/삭제 요청을 처리할 때, 요청 헤더나 DB 세션 설정값에 app.edit_token 등을 주입하여 **해당 토큰이 일치할 경우에만** 동작하도록 만들 수 있습니다. 토큰 자체는 글 작성 시 백엔드에서 생성하여 응답으로 전달하고, 클라이언트는 이를 로컬 저장(쿠키 또는 localStorage)해뒀다가 수정/삭제 시 사용하게 합니다.

참고: 게시글 수정/삭제를 위한 대안으로 **"게시글 비밀번호"** 입력 방식을 고려할 수도 있습니다【7】. 사용자가 글 작성 시 직접 비밀번호를 정해두고, 나중에 수정/삭제하려면 그 비밀번호를 입력하도록 하는 전통적인 익명 게시판 방식입니다. 이는 토큰 분실 시 대비책이 되고 구현이 단순하지만, 사용자 경험 측면에선 번거로울 수 있습니다. 초기 구현에서는 자동 생성된 토큰을 사용하는 편의성을 택하고, 필요시 비밀번호 방식을 추가로 도입합니다.

4.3 인증 및 익명성 유지

핵심 과제: 로그인 없이도 특정 사용자가 쓴 글/댓글에 대한 수정 또는 삭제 권한을 **본인에게만** 부여하는 것입니다. 이를 위해 아래와 같은 다층적 익명 인증 전략을 적용합니다:

- **디바이스 식별 (기기 기반 세션):** 사용자의 브라우저 정보를 기반으로 **디바이스 핑거프린팅**을 수행합니다. 예를 들어 사용자 에이전트, 화면 해상도, 타임존, 언어, 쿠키 지원 여부 등의 조합으로 해시를 생성하여 *하나의 기기*를 식별합니다. 백엔드(Supabase)에 이 해시를 키로 한 anonymous_users 레코드를 만들고 고유 ID 를 발급합니다【1】. 이를 통해 로그인 없이도 동일 기기에서 오는 요청에 일정한 사용자 ID 를 부여할 수 있습니다.
- **세션 쿠키 발급:** 사용자 최초 방문 시 서버가 임의의 세션 ID(예: UUID)를 생성하여 **HttpOnly 쿠키**로 발급합니다【1】. 쿠키 유효기간은 30 일 등으로 설정하여 동일 브라우저에서의 지속성을 갖습니다. 이 세션 ID 는 anonymous_user_id 와 매핑되어 서버에서 관리되고, 추후 요청에서 쿠키를 보내면 해당 익명 사용자를 신원 확인할 수 있습니다. (쿠키가 없거나 만료된 경우, 디바이스 핑거프린트를 사용해 서버측에서 동일인 추정되면 기존 ID 를 이어쓰고, 그렇지 않으면 새 ID 생성).
- **게시글별 편집/삭제 토큰:** 앞서 언급한 edit_token, delete_token 은 **각 게시글당 별도로 생성되는 1 회용 비밀번호** 같은 개념입니다. 작성 성공

시 응답 payload 에 이 토큰들을 포함하여 클라이언트에 전달하고, 클라이언트는 이를 안전하게 저장해둡니다. 사용자가 나중에 해당 글을 수정/삭제하려 할 때, 이 토큰을 첨부해야만 요청을 승인합니다. 이 방법은 쿠키가 지워지거나 IP 가 바뀌어도 **토큰만 알고 있으면 권한을 유지**할 수 있다는 장점이 있습니다. 다만 사용자가 토큰을 잃어버리면 본인도 글을 지울 수 없게 되므로, 토큰은 노출되지 않도록 주의하며, 분실 시 대응은 어려운 점을 안내합니다.

이상의 복합 접근을 통해 **익명성을 깨지 않으면서도** 일정 수준의 사용자 식별과 권한 관리를 실현합니다. 물론 이러한 방법들은 완벽하지 않아서, 브라우저 쿠키를 지우거나 시크릿 모드를 사용하는 경우 동일인 식별이 어려워질 수 있습니다【2】. 또한 IP 기반 식별은 한계가 뚜렷합니다 (공용 IP 사용자 구분 불가 등)【3】. 따라서 **가능한 한 구현이 단순하고 확실한 방법**을 우선 적용하고, 추후 문제 발생 시 보완하는 방향으로 합니다. (예: 추후 특정 중요 기능에서는 선택적으로 OAuth 소셜 로그인 도입 검토).

4.4 좋아요 기능 구현 및 성능 고려

- **좋아요 중복 방지:** 익명이지만 동일 콘텐츠에 대해 한 사용자가 여러 번 좋아요를 누를 수 없도록 해야 합니다. 이를 위해 **클라이언트 측**에서 한 번 누른 좋아요에 대한 상태를 로컬에 저장하고, **서버 측**으로는 IP 나 세션 기준으로 중복 요청이 오면 무시합니다【1】. 더 정교하게는 PostLikes 같은 테이블을 두어 (post_id, anonymous_user_id)를 유니크 키로 저장해 중복을 원천 차단할 수 있습니다.
- **동시성 및 이벤트 처리:** 좋아요 수는 매우 빈번하게 변경될 수 있는 데이터입니다. 다수 사용자가 동시에 같은 게시글에 좋아요를 누를 경우, **데이터베이스의 동시 업데이트**로 락 경합이나 성능 문제가 생길 수 있습니다【9】. 이를 해결하기 위해 **낙관적 갱신 (Optimistic Update)** 및 **이벤트 큐/배치** 전략을 사용합니다:
    - 사용자가 좋아요를 누르면, 일단 클라이언트 UI 상 즉시 like_count 를 +1 증가시켜줍니다 (사용자 경험 상 즉각 반영).
    - 서버에서는 별도의 **좋아요 이벤트 로그 테이블**에 post_id, user_id, timestamp 등을 기록만 합니다 (INSERT). 그리고 Redis 등에 post:{id}:like_count_pending 과 같은 키에 증분을 해둡니다.
    - 일정 주기(예: 매 1 분 또는 특정 이벤트 수 누적 시)로 배치 작업이 실행되어, 로그를 집계하여 Posts 테이블의 like_count 를 한 번에 업데이트합니다 (예: 1 분 동안 같은 글에 좋아요 100 개 발생 시, 최종 +100 으로 합산하여 업데이트)【9】.
    - 이런 **Eventual Consistency (궁극적 일관성)** 모델에서는 실시간 정확한 좋아요 수와 약간의 차이가 있을 수 있지만, 대규모

트래픽에서도 DB 부하를 낮추고 성능을 높이는 데 매우 효과적입니다【9】. (유튜브나 인스타그램 등의 좋아요 숫자가 즉시 변동되지 않고 조금 지연돼서 정확해지는 것도 이 때문입니다).

- **정합성 vs 성능:** 실시간 정합성을 약간 희생하고 성능을 택하는 위 접근은 사용자 경험에 큰 문제를 일으키지 않으며【9】, 서비스 안정성을 높입니다. 초기 구현에서는 너무 복잡하게 가지 않고, **DB 트랜잭션으로 like_count 컬럼 +1 업데이트**하는 단순 방식으로 시작할 수 있습니다. 그러나 트래픽이 늘면 해당 레코드 락으로 인한 지연이 발생할 수 있으므로, 미리 **배치 집계 방식**을 고려해두고 단계적으로 전환합니다.

4.5 정렬 및 검색 기능 최적화

- **최신순/인기순 정렬:** 게시판 기본 정렬은 **최신순**이며, 사용자 선택에 따라 **인기순**(좋아요 많은 순)도 제공됩니다【13】. 이를 지원하기 위해 데이터베이스 인덱싱과 쿼리 최적화가 필요합니다:
  - Posts 테이블의 created_at 컬럼에 **인덱스**를 생성하여 최신순(ORDER BY created_at DESC) 조회를 빠르게 합니다【14】.
  - like_count 컬럼에도 인덱스를 생성하여 인기순(ORDER BY like_count DESC) 조회 시 사용합니다【14】. (단, like_count는 수시로 변동되므로 너무 자주 인덱스 리빌드가 일어나지 않도록 배치 업데이트 전략과 함께 고려).
  - 인덱스는 **다중 컬럼 인덱스**도 검토합니다. 예를 들어 특정 카테고리별 최신/인기 게시글 조회를 원하면 (category, created_at) 혹은 (category, like_count) 복합 인덱스를 둘 수 있습니다【15】. 자주 사용되는 쿼리에 맞춰 인덱스를 구성하여 WHERE + ORDER BY 조건이 인덱스만으로 처리가 되도록 최적화합니다【16】.
- **검색 기능:** 기본적으로 Next.js 클라이언트에서 키워드로 검색 시, Supabase Full-Text Search 기능을 이용하거나, 간단히 WHERE title ILIKE '%키워드%' OR content ILIKE '%키워드%' 형태로 구현합니다. 추후 게시글 수가 많아지면 ElasticSearch 연동이나 Supabase의 Full-Text GIN 인덱스를 활용하는 것을 고려합니다.
- **캐싱과 CDN:** 정렬된 게시글 목록이나 인기글 상위 n개 등은 빈번히 요청되므로, **Application 레벨 캐싱**을 적용합니다. 예를 들어 Upstash Redis(Vercel KV)를 사용해 posts:latest나 posts:top10 키로 JSON 결과를 저장하고, 사용자가 게시글을 추가/삭제하면 해당 캐시를 갱신(또는 무효화)하는 식입니다. 캐시 유효기간을 30~60초 정도로 짧게 설정해 **실시간성**과 **효율성**을 절충합니다.

5. 핵심 기술 과제 및 해결 방안

5.1 익명성 유지와 보안 강화

도전 과제: 익명 게시판에서는 사용자의 신원이 드러나지 않아 **악용 및 어뷰징**의 우려가 있습니다. 또한 익명으로 권한을 관리하기 때문에 **보안 취약점**이 생기기 쉽습니다. 다음과 같은 대책을 마련합니다:

- **쿠키/세션 기반 중복 방지의 한계:** 단순히 쿠키에 플래그를 저장하여 "이미 좋아요 누른 글"을 막거나, IP 로 제한하는 방법은 매우 쉽게 우회될 수 있습니다【2】【3】. 예를 들어 사용자가 브라우저 쿠키를 삭제하거나, 시크릿 모드를 쓰면 쿠키 기반 제약은 초기화됩니다. VPN 이나 프록시로 IP 를 바꾸면 IP 기반 차단을 피할 수 있습니다. 따라서 **클라이언트-서버 양측에서 중복 방지** 로직을 구현하되, 결정적으로 **서버 측 검증**에 무게를 둡니다. 서버는 (세션 or 디바이스 ID)와 (게시글 ID)를 묶어 이미 액션을 취한 이력이 있으면 idempotent 하게 처리합니다.
- **IP 레이트 리미팅:** 악의적인 사용자가 다량의 요청을 보내 서비스에 부담을 주는 것을 막기 위해 Rate Limiting 을 적용합니다. Upstash Redis 와 같은 키-값 DB 를 활용해 **슬라이딩 윈도우 알고리즘**으로 구현합니다【12】. 예를 들어:
  - 게시글 작성: 동일 IP 에서 **10 분에 5 건** 이상 글 작성 시 거부.
  - 댓글 작성: **1 분에 20 건** 이상 작성 시 캡차 요구 또는 일시 차단.
  - API 전체: **분당 1000 회** 이상 호출 시 해당 세션 차단.

  이러한 레이트 리미터는 Next.js Middleware 에서 Edge Function 으로 동작시켜, 본격적인 API 로직을 타기 전에 차단할 수 있도록 합니다. Redis 의 INCR 와 TTL 을 이용한 슬라이딩 윈도우 카운터 기법으로 정확도를 높입니다【12】.

- **VPN/프록시 탐지:** 추가로, MaxMind GeoIP 나 IPQualityScore API 등을 사용해 요청 IP 의 프록시/VPN 여부, 이상 트래픽 점수를 확인할 수 있습니다. 이를 통해 명백한 우회 시도가 보이면 일부 기능을 제한하거나 캡차를 요구합니다. (이 기법은 약간의 비용이 들지만, 악용 방지 효과를 높일 수 있습니다. 정확도가 99% 수준으로 알려짐【**】).
- **점진적 신뢰 등급:** 새로운 익명 사용자는 잠재적으로 악용 가능성이 있으므로, **처음에는 제한된 권한**을 부여하고 일정 이상 기여를 하면 제한을 완화하는 시스템을 고려합니다. 예:
  - 신규 사용자: 하루 게시글 3 개, 댓글 10 개 등 비교적 빡빡한 제한 + 매 게시글 작성 시 캡차 확인.
  - 활동 많은 사용자: 일정 기간 문제 행동 없이 이용한 경우 제한 상향 (하루 10 개 등) 및 캡차 면제.

- 신뢰 사용자: 커뮤니티에서 긍정적 참여가 많고 운영자가 인정한 경우, 자율권 부여 (예: 신고나 숨김 처리에 영향 미치게).

이 Trust Level 시스템은 추후 커뮤니티 규모가 커지면 도입하고, 초반에는 기본적인 레이트 제한과 수동 모니터링으로 충분히 커버합니다.

5.2 서버리스 환경의 효율성과 비용 관리

- **콜드 스타트 대책:** Vercel Serverless Functions는 일정 기간 트래픽이 없으면 인스턴스를 종료하고, 다음 요청 때 Cold Start 지연이 발생할 수 있습니다【29】. 이를 완화하기 위해 핵심 API는 Edge Function으로 전환하거나, 주기적인 *ping*으로 함수를 예열하는 방법도 검토합니다. 또한 Vercel이 제공하는 Provisoned Concurrenty(미리 함수 인스턴스 확보) 옵션을 사용할 수 있지만 비용 증가 요인이므로 신중히 판단합니다【30】. 함수 메모리 할당을 늘려 CPU 성능을 높이면 초기 부팅 시간을 줄일 수 있다는 보고도 있으니 튜닝합니다【31】.
- **외부 상태 저장:** 서버리스 함수는 **무상태(stateless)**이므로, 세션이나 캐시를 메모리에 저장할 수 없습니다. 이에 대한 대책으로 **Redis (Upstash)**를 세션/캐시 저장소로 사용하고, Supabase 등의 Persisted DB에 최종 데이터 저장을 합니다. 좋아요 중복 체크, 레이트 리밋 카운트, 세션 토큰 관리 등이 모두 Redis에 저장되어 여러 함수 인스턴스 간에 공유되도록 합니다. 이러한 외부화로 서버리스의 스케일 아웃 시에도 **일관된 상태 관리**가 가능합니다.
- **성능 모니터링:** Sentry와 같은 APM(Application Performance Monitoring)을 도입하여 함수 지연 시간, 오류, DB 쿼리 성능 등을 추적합니다. Core Web Vitals (LCP, FID 등)도 모니터링하여 광고 스크립트 등이 성능에 미치는 영향을 분석합니다【57】.
- **비용 예측 및 계획:** 서버리스 아키텍처는 초기에는 저렴하나 트래픽 증가 시 비용이 **선형 증가**합니다【32】. 예상 월간 운영비를 추산하면 다음과 같습니다 (월간 활성 사용자 1만 명 기준):
  - Vercel Pro 플랜: $20/월 (빌드/배포 및 기본 서버리스 호출 포함)【35】.
  - Supabase Pro 플랜: $25/월 (8GB DB, 250GB 대역폭 등)【36】.
  - Upstash Redis: $20/월 (요청량 기반 요금제 추산).
  - 이미지 CDN/스토리지: $10/월 (전송량 및 저장용량에 따라).
  - AI API (예: Google Gemini): $50~200/월 (사용량에 따라 유동).

  합계: 약 $115 ~ $265/월 예상. 이는 수익화로 얻을 광고 수입 내에서 충분히 감당 가능한 수준이지만, 항상 **무료 할당량 초과 여부**를 모니터링하고, 비용 증가 요인이 발생하면 적절히 인프라를 조정해야 합니다. 예를 들어 Supabase는 무료 플랜에 500MB까지 DB, 5GB까지 대역폭을 지원하지만【36】, 데이터가 쌓이거나 대시보드 조회가 많아지면

제한을 넘길 수 있으므로 적시에 Pro 로 업그레이드합니다【38】. Firebase 의 경우도 일정 이상 쓰면 Blaze 요금으로 전환되어 과금되니 주의합니다【39】【40】.

- **확장 대비:** 현재 구조는 AWS Lambda 기반이지만, 트래픽이 폭증하여 비용이 기하급수로 늘거나 성능 한계에 부딪힐 경우, **컨테이너 기반 (예: AWS ECS, Kubernetes)**로의 전환도 염두에 둡니다【32】. 다만 해당 임계점에 도달하기 전까지는 서버리스로 **개발 민첩성**과 **운영 편의성**을 최대한 활용하는 전략입니다.

5.3 콘텐츠 필터링 및 UGC 모더레이션

왜 필요한가: 익명 게시판은 사용자가 어떤 내용을 올릴지 예측하기 어렵습니다. 서비스 신뢰도와 광고 게재를 위해서는 사용자 생성 콘텐츠(UGC)에 대한 **강력한 모니터링 및 필터링**이 필수입니다【5】. 특히 AdSense 정책상 **게시자는 사이트의 모든 콘텐츠에 책임을 져야 하며**, 부적절한 콘텐츠가 노출되면 광고가 중지될 수 있습니다【5】.

5.3.1 욕설 및 유해 표현 필터링

- **기본 금칙어 필터:** 우선 댓글과 게시글 입력 시 **한국어 비속어 라이브러리**인 badwords-ko 와 영어 비속어 필터 bad-words 등을 적용합니다【1】. 사용자가 제출을 시도할 때 금칙어가 발견되면 해당 단어를 *** 등으로 치환하거나, 아예 **제출을 막고 경고 메시지**를 띄웁니다. 이 1 차 필터로 일반적인 욕설을 상당 부분 걸러낼 수 있습니다.
- **금칙어 필터 한계:** 사용자들은 필터를 피하기 위해 다양한 우회 방법을 사용합니다. 예를 들어:
  - 특정 글자를 특수문자나 숫자로 치환: *“ㅅ#ㄲ1”*, *“ㅆ1발”* 처럼【21】【22】.
  - 자모음을 분리하거나 일부 글자만 쓰는 형태: *“ㅈㄹ (지랄)”*, *“개ㅆ…”* 등.
  - 정상 단어를 포함한 애매한 표현: *“18 채널”*(십팔채널)처럼 숫자 ‘18’을 욕설로 오인하게 만드는 경우【20】.

  단순 치환으로는 이런 경우를 모두 잡아내기 어렵고, 과도하게 필터링하면 **오검열(false positive)** 문제가 발생합니다【20】. 예: “피에로”라는 단어에 “피**“로 필터가 걸린다든지.

- **고급 필터링 (AI 활용):** 이를 보완하기 위해 **딥러닝 기반 욕설 탐지 모델**을 도입 검토합니다. 한국어 욕설 탐지는 자모 단위 CNN 모델이나 한글 형태소 기반 LSTM 모델로 구현 사례가 있으며, 금칙어 대비 훨씬 높은 정확도(85~90% 이상)를 보입니다【21】. 이러한 모델은 금칙어 리스트에 없거나 변형된 욕설도 학습을 통해 잡아낼 수 있고, 맥락까지 고려할 수 있다는 장점이 있습니다【21】.
    - 현실적으로 자체 모델 개발이 어렵다면, Google Perspective API 같은 서비스를 활용하여 댓글/게시글의 toxicity score(악성 정도 점수)를 받을 수 있습니다. 일정 임계치 이상이면 자동 차단하거나 관리자 검토로 넘기는 **2단계 필터**를 구현합니다.
    - 예시 (의사 코드):

```
const content = "이 댓글 내용";
const basicFilter = cleanWithBadwordsKo(content);
const score = await perspective.analyze(content);
if (score.toxicity > 0.8) rejectPost();        // 욕설 심한 경우 블락
else if (score.toxicity > 0.5) flagForReview(); // 다소 애매하면 관리자 검토 대상으로
else acceptPost(basicFilter);
```

-
- **이미지 및 기타 콘텐츠:** 이미지에 대한 **NSFW 검출**도 고려합니다. 사용자가 업로드한 이미지가 과도한 폭력, 성인물을 포함하는지 판별하기 위해 Cloud Vision API 등의 Safe Search 기능이나, 오픈소스 NSFW 모델을 사용해 자동 분석합니다. 텍스트보다 훨씬 적은 빈도로 발생할 것이므로 우선은 **사용자 신고**에 의존하고, 추후 필요하면 도입합니다.

5.3.2 관리자 도구와 커뮤니티 자율 기능

- **신고 기능:** 모든 게시글과 댓글에는 “신고” 버튼을 제공합니다. 사용자가 부적절한 내용을 발견하면 사유와 함께 신고할 수 있고, 이는 서버 DB의 reports 테이블에 누적됩니다【27】. 일정 횟수 이상 신고가 모이면 자동으로 해당 콘텐츠를 임시 블라인드 처리하고 관리자가 검토하도록 플래그를 세웁니다.
- **관리자 모니터링:** 초창기에는 별도의 관리자 페이지가 없더라도, 신고 내역이나 필터에 걸린 차단 내역을 **주기적으로 확인**해야 합니다【1】. 예를 들어 신고/차단 로그를 Slack이나 이메일로 받아보는 것을 고려합니다. 추후 관리 페이지를 제작하여 실시간으로 게시글/댓글 리스트와 신고 상황, 차단 로그, 사용자 정보(익명 ID 기반) 등을 조회하고 **수동 제재**를 할 수 있도록 합니다【1】.

- **로그 기록:** 욕설이 검출되어 차단된 내용, 유저 ID 와 IP, 발생 시각 등을 모두 DB 나 로그 파일에 저장합니다【1】. 혹시 모를 분쟁이나 실수 차단에 대비해 증빙을 남기고, 필터 개선 자료로 활용합니다.
- **커뮤니티 가이드 준수:** 이용자가 자발적으로 가이드라인을 지키도록 **공지 및 교육**도 중요합니다. 첫 방문 시 커뮤니티 원칙을 안내하거나, 게시물 작성 전 가이드 요약을 보여주어 사전 예방을 합니다.
- **AdSense 정책과 연계:** Google AdSense 측에서는 **포럼, 댓글 등 UGC 영역을 특히 위험**요소로 봅니다【4】. "모든 게시자는 자신의 사이트 콘텐츠에 대해 책임을 져야 한다"고 명시되어 있기에【5】, 위 모든 모니터링 노력에도 불구하고 문제 발생 시 **Google 에 검토 요청**을 하는 절차도 알아둡니다. (예: 정책 오탐지로 광고가 제한될 경우 이의를 제출해 검토받는 프로세스)

5.4 Google AdSense 통합 및 정책 준수

5.4.1 AdSense 가입 및 승인 준비

- **도메인 소유 및 연령 요건:** 서비스 운영자는 만 18 세 이상이어야 하며, **자신의 도메인** 소유권을 증명할 수 있어야 합니다. 현재 Vercel 에 연결된 커스텀 도메인이 있다면 해당 도메인으로 신청하고, 없을 경우 우선 Vercel 제공 *.vercel.app 도메인으로도 테스트 가능하나, 최종 승인에는 1 차 도메인이 권장됩니다【1】.
- **콘텐츠 확보:** 광고 게재를 신청하기 전에 사이트에 **충분한 양의 독자적 콘텐츠**가 있어야 합니다【13】. 다른 곳에서 복사한 콘텐츠가 대부분이거나, 게시글 수가 너무 적으면 승인이 거절될 수 있습니다. 일반적으로 **최소 5~10 개의 양질의 게시물**이 필요하다는 사례가 많습니다【55】. 따라서 AI 생성 컨텐츠 + 사용자 게시판 글이 일정 수 이상 축적된 후 신청합니다.
- **광고 코드 삽입:** AdSense 승인 과정에서 제공되는 **광고 스크립트**(예: <script async src="...adsbygoogle.js?client=ca-pub-XXXX" ...>와 <ins class="adsbygoogle" ...>)를 Next.js 의 _app 혹은 특정 페이지에 삽입해야 합니다【1】. Next.js 에서는 next/script 컴포넌트를 이용하여 strategy="afterInteractive"로 스크립트를 로드하면 **페이지 렌더를 막지 않고** 광고를 불러올 수 있습니다【1】. 예:

```
// _app.tsx or layout.tsx
import Script from 'next/script';
function MyApp({ Component, pageProps }) {
  return (
    <>
```

```
      <Script
        id="adsense-init" strategy="afterInteractive"

src={`https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js?client
=${process.env.NEXT_PUBLIC_ADSENSE_CLIENT_ID}`}
        crossOrigin="anonymous"
      />
      <Component {...pageProps} />
    </>
  );
}
```

- 이 코드를 배포한 뒤, AdSense 측에서 사이트 크롤링을 통해 코드 삽입 여부를 확인하고 승인을 진행합니다.
- **정책 검수 통과:** Google의 **프로그램 정책 및 게시자 콘텐츠 가이드라인**을 철저히 준수해야 합니다【5】. 자동화된 크롤러뿐 아니라 사람이 사이트를 방문해 검수할 수 있으므로, **성인물, 저작권 위반, 도박, 마약, 무기, 혐오, 폭력 미화** 등 금지 콘텐츠가 없어야 합니다. 또한 사이트 UI 측면에서도 **광고 배치가 사용자를 기만하지 않는지** (예: 콘텐츠와 광고를 헷갈리게 배치하면 안 됨) 확인합니다【5】.

5.4.2 광고 배치 전략과 UX

- **권장 광고 위치 및 형태:** (앞서 2.2절과 중복되는 내용이지만 구체화)
  - 헤더 아래 **상단 배너** – 첫 화면에 노출되며, 콘텐츠를 밀어내리지 않도록 상단 여백 등에 자연스럽게 배치.
  - **게시글 리스트 사이** – 예를 들어 목록 5개마다 1개 광고 카드를 끼워 넣습니다. 이때 "스폰서" 레이블을 붙여 **콘텐츠와 광고를 명확히 구분**해야 합니다【5】.
  - **게시글 본문 중간** – 긴 글을 읽을 때 가운데 등장하는 300x250 정도의 광고. 하지만 문단과 문단 사이 충분한 여백을 주고, 광고임을 표시하여 **의도치 않은 클릭**을 유도하지 않도록 합니다【5】.
  - 데스크톱 우측 **사이드바 고정광고** – 스크롤해도 항상 보이는 위치에 배너를 두면 **노출 시간**이 늘어나 효과가 좋습니다. 다만 모바일에선 화면을 가릴 우려가 있어 숨깁니다.
- **광고 로드 최적화:** 광고 스크립트는 비동기로 불러오지만, 광고 슬롯이 많은 경우 초기 로딩 속도가 느려질 수 있습니다【58】. **레이지 로딩(Lazy Loading)** 기법을 적용해 현재 화면에 보이지 않는 광고는 나중에 로드합니다【49】. Intersection Observer API를 활용해 광고 요소가 뷰포트 근처에 오면 그때 adsbygoogle.push()로 로드하는 식입니다. 이를 통해 Core Web Vitals 지표(LCP 등)에 영향을 최소화합니다.

- **광고와 콘텐츠 구분:** 광고를 **메뉴나 링크처럼 보이게** 꾸미는 것은 금지입니다【5】. 예를 들어 "인기 글" 섹션에 광고를 섞어놓고 표시하지 않으면 오해의 소지가 있으므로, **"광고" 혹은 "스폰서"** 라벨을 명시하고, 스타일도 콘텐츠 카드와 약간 다르게 해서 사용자가 식별할 수 있도록 디자인합니다.
- **팝업 광고 미사용:** 사용자 37%가 광고 차단기를 사용하고 그 이유 중 48%가 성가신 광고 때문이라는 조사도 있습니다【54】. 무서핑 서비스는 **팝업/팝언더 광고, 자동 소리 재생 광고** 등 방해형 광고는 일절 사용하지 않습니다. 필요하다면 향후 **광고 미노출 유료 구독 모델** 같은 우회 수익 모델도 고려합니다【54】. 광고 차단기 사용자에 대해서는 별도의 제재는 하지 않고, **양질의 콘텐츠 제공과 절제된 광고**로 자발적 허용을 유도하는 방향을 취합니다.

5.4.3 AdSense 운영 및 정책 준수 방안

- **무효 트래픽 모니터링:** AdSense 측에서는 **무효 클릭 및 트래픽**을 엄격히 단속합니다【6】. 혹시 경쟁 사이트가 악의적으로 광고를 여러 번 클릭하거나 트래픽 봇을 통한 공격을 가할 수 있으므로, 자체적으로 이상 클릭 감지 로직을 둘 수 있습니다. 예를 들어 동일 IP 에서 짧은 시간에 반복적으로 광고 클릭 신호가 감지되면 경고 로그를 남기고 AdSense 에 알리는 등의 조치를 합니다. 또한 **절대 사용자에게 광고 클릭을 유도하지 않으며**, 인위적 트래픽 교환 프로그램 등을 사용하지 않습니다【6】.
- **UGC 정책 준수:** 앞서 5.3 절의 콘텐츠 필터링을 통해 광고 게재 페이지에 부적절한 내용이 나오지 않도록 지속 관리합니다. 특히 **실시간 채팅, 비공개 커뮤니케이션 영역은 광고 게재 불가** 정책도 있으므로【5】, 우리 서비스의 공개 게시판은 이에 해당하지 않지만, 혹시 추후 1:1 채팅 같은 기능이 생긴다면 해당 페이지엔 광고를 표시하지 않습니다.
- **정기 감사 및 교육:** 운영진은 주기적으로 AdSense 정책 업데이트를 확인하고 준수 여부를 점검합니다. 이용자들에게도 "클릭 유도 금지" 등의 규정을 안내하여 혹시 모를 **선의의 규정 위반**이 없도록 합니다. (예: 어떤 열성 팬이 "사이트 운영에 도움이 되길 바란다"며 일부러 광고를 여러 번 클릭하는 것도 정책 위반이므로, 이런 행위를 포럼 공지 등을 통해 자제시키도록 알립니다【5】.)

6. 배포 및 운영

6.1 CI/CD 및 배포 파이프라인

- **Vercel 연동 배포:** GitHub 저장소와 Vercel 을 연결하여 **Push -> Build -> Deploy** 자동화합니다【1】. main 브랜치에 푸시된 커밋은 자동으로 프로덕션 배포되며, 기타 브랜치는 Preview 환경으로 배포되어 기능 테스트에 활용합니다. Vercel 의 **미리보기 URL** 기능을 통해 PR 마다 배포 링크를 받아 디자이너/PM 이 바로 확인하고 피드백할 수 있습니다【1】.
- **환경별 설정:** .env.local, .env.production 파일을 분리하고 Vercel 프로젝트 설정에서 환경 변수 값을 관리합니다【1】. 개발 환경에선 테스트 AdSense 코드 (또는 광고 비활성화), 프로덕션엔 실제 코드 삽입 등 분기 처리도 코드 내에서 환경 변수를 참조하여 수행합니다.
- **테스트 자동화:** 주요 기능에 대해 Jest 로 단위 테스트, Cypress 로 E2E 테스트를 작성해 두고, GitHub Actions 에서 PR 시 자동 실행하도록 구성합니다【1】. 배포 전에 기본 시나리오들이 깨지지 않았는지 확인하여 품질을 높입니다.
- **모니터링:** 배포된 후 Sentry 를 통해 프론트/백엔드 오류를 수집하고, Cloudflare Analytics 나 Vercel Analytics, 혹은 자체 로그 수집을 통해 트래픽과 성능 데이터를 추적합니다. 이는 운영 중 이슈를 빠르게 발견하고 대응하는 데 도움을 줍니다.

6.2 운영 및 유지보수 계획

- **콘텐츠 모더레이션 팀:** 서비스 성장에 따라 **운영자/모더레이터** 역할이 중요해집니다. 초반에는 작은 팀(또는 1 인)이 모든 신고와 로그를 직접 모니터링하겠지만, 유저 베이스가 커지면 자원 봉사 모더레이터나 정식 운영 인력을 충원하는 계획을 세웁니다. 커뮤니티 신뢰도를 위해 **일관된 가이드라인** 하에 컨텐츠를 관리해야 합니다.
- **개인정보 보호:** 익명 서비스라고 하더라도 이용자의 IP, 디바이스 정보, 쿠키 등의 데이터는 수집될 수 있습니다. 이에 대한 **개인정보처리방침**을 수립하고 공개합니다. 또한, Google Analytics 와 같은 사용자 추적 도구는 **필요 최소한**으로 사용하며, 법적 요구에 따라 쿠키 배너 등을 검토합니다【1】.
- **지속적인 개선:** 사용자 피드백 창구(게시판 내 건의게시판 등)를 만들어 기능 개선 의견을 수렴합니다. UI/UX 개선이나 신규 기능 요청이 활발하면 우선순위를 정해 스프린트 단위로 반영합니다.
- **스케일 모니터링:** CPU/메모리 사용량, DB 쿼리 시간, Redis 용량 등을 지속 모니터링하여 **임계치 도달 전에** 용량을 늘리거나 구조를 개편합니다. 특히 DB 는 **인덱스 재구성**이나 **샤딩/리플리카** 등의 작업이 필요해질 수 있으므로 미리 대비합니다.

7. 개발 일정 및 로드맵

개발은 단계별로 진행하며, 각 단계 종료 후 다음 단계로 넘어갑니다. 예상 소요 기간과 주요 작업은 아래와 같습니다:

1. **1 단계: MVP 구현 (약 4~6 주) – 핵심 게시판 기능과 기본 수익화 탑재**
    - 게시글/댓글 작성, 조회, 삭제의 기본 사이클 구현 (익명 토큰 방식 적용).
    - 게시판 UI/UX 완성 및 이미지 업로드 기능 구현.
    - 기본 욕설 필터 적용 및 레이트 리미팅 (간단한 쿠키/IP 기반으로 시작).
    - AdSense 코드 스니펫 삽입 및 **광고 슬롯** UI 배치 작업. (초기엔 광고 게재가 안 되더라도 자리 확보)
    - **성과:** 커뮤니티 서비스 기본 오픈, 초기 사용자 확보 및 피드백 수집.
2. **2 단계: 보안 및 모더레이션 강화 (약 2~3 주) – 서비스 악용 대응 체계 구축**
    - **고급 욕설 필터링** 도입: 정규식 추가 보완, Perspective API 혹은 자체 ML 모델 연동 시험.
    - **사용자 신고 기능** 구현 및 신고 처리 프로세스 수립. 신고된 콘텐츠 관리자 확인 가능하도록 임시 UI 또는 로그 알람 구축.
    - **CAPTCHA 적용:** 특정 행동(짧은 시간 내 다수 글쓰기 등)에 reCAPTCHA v3 적용하여 봇 행위 방지.
    - **Adsense 정책 점검:** 실제 광고 게재 테스트를 통해 페이지 구성의 문제점 보완 (예: 광고 라벨 표시 등 미비사항 수정).
    - **성과:** 커뮤니티 환경이 비교적 안전해지고, 광고 게재 준비 완료 (콘텐츠 누적도 충분).
3. **3 단계: 성능 최적화 및 확장 준비 (약 2~3 주) – 트래픽 증가 대비 기술적 튜닝**
    - **캐싱 도입:** 인기 글, 최신 글 목록 Redis 캐싱 + ISR 적용으로 응답 속도 개선.
    - **좋아요 기능 개선:** 이벤트 큐 도입 및 배치 처리로 DB 락 감소, 대량 트래픽 견조성 상승.
    - **DB 최적화:** 인덱스 점검 및 쿼리 최적화. 필요한 경우 Supabase read replica 도입 검토.
    - **빌드/배포 최적화:** 트래픽이 늘면 빌드 아웃풋을 분리(코드-splitting)하고, 이미지 정적 최적화 등 Core Web Vitals 개선 작업.
    - **성과:** 1 만 DAU 규모까지 쾌적하게 운영 가능한 성능 기반 마련.
4. **4 단계: AI 기능 고도화 (약 3~4 주) – 차별화 요소 강화 및 비용 효율화**
    - **AI 콘텐츠 생성 API 연동:** OpenAI 대신 **Google Gemini API** 를 시범 도입하여 AI 콘텐츠 생성 비용 절감 및 안정성 테스트【1】. 한국어 지원 품질 확인.

- **AI 유머/공포 최적화:** 프롬프트 엔지니어링을 통해 서비스 특화된 AI 생성물의 품질을 향상. 예: 한국 밈(meme) 문화나 공포 소재 데이터 조금 fine-tuning 등.
- **AI 안전장치:** AI 생성 콘텐츠에도 유해표현 필터 적용, 잘못된 정보 생산 예방 등 정책 수립.
- **성과:** AI 기능과 게시판이 융합되어 사용자들이 **AI 와 상호작용하는 커뮤니티**로 진화. 비용구조도 최적화되어 수익률 개선.

총 개발 기간: 약 **11~16 주** 소요 예상 (3~4 개월). 팀 인원과 우선순위에 따라 변동 가능.
예상 출시 일정: 2024 년 1 분기 내 베타 오픈, 2 분기 정식 출시 목표.

8. 결론 및 권고사항

기술 평가 요약: 무서핑 서비스에 익명 게시판과 광고 수익화를 추가하는 것은 **충분히 구현 가능**하며, Modern Web 스택(Next.js, Supabase 등)으로 비교적 짧은 기간 내 출시할 수 있습니다. 다만 **익명성으로 인한 보안/운영 리스크**와 **UGC 광고 정책 준수**라는 두 가지 큰 과제를 간과해서는 안 됩니다.

- **익명성 vs 악용의 균형:** 완전 익명으로 열려있는 만큼 초기부터 **강력한 악용 방지 장치**를 넣어야 합니다. 레이트 리밋, 필터, 신고, 운영자 모니터링이 유기적으로 작동하도록 설계합니다. 동시에 선의의 사용자에게 너무 불편을 주지 않도록 **UX 밸런스**도 신경써야 합니다 (ex: 과도한 캡차는 이탈 원인). 커뮤니티는 **신뢰**를 바탕으로 성장하므로, 점진적 신뢰 등급 시스템 등 **커뮤니티 내적 자정 능력**을 키우는 방향도 추구합니다.
- **기술 선택의 원칙:** 여러 구현 방법 중 **가장 구현이 쉬운 쪽을 선택**하는 것을 권장합니다【1】. 예컨대 서버리스 환경에서 복잡한 데이터 일관성을 유지하려 애쓰기보다는, **가능하면 단순한 구조**로 시작하는 것입니다. Supabase 와 같은 서비스는 많은 기능을 제공하므로 별도 서버 개발 없이 활용하고, Vercel KV 나 Edge 기능도 처음엔 복잡도를 낮추는 선에서 사용합니다. 만약 특정 기술이 너무 어렵거나 제약이 많다면 **대안을 과감히 선택**합니다 (예: 실시간성이 필요하면 Firebase 도입, 서버리스 한계를 느끼면 Express 서버로 전환 등). 초기에 **후자가 더 나은 방향**이라 판단되면 유연하게 방향 선회하는 것도 필요합니다【1】.
- **광고 수익화와 품질 유지:** 광고는 서비스 유지에 중요한 수익원이지만, **한 순간의 정책 위반**이 치명타가 될 수 있습니다. 따라서 **콘텐츠 품질과 광고 정책 준수는 타협 불가능한 0 순위** 과제로 삼아야 합니다. 이를 위해 AI 를 비롯한 기술을 최대한 활용하되, **최종 책임은 사람(운영자)**에게 있음을 인식하고 적극적으로 관리합니다【5】.

- **확장성과 비용:** 제시된 구조는 초기 1~5 만 유저 규모까지는 문제없을 것입니다. 그 이상 성장이 보이면, 아키텍처를 모니터링하면서 **리팩토링과 투자**를 병행해야 합니다. 캐싱, 분산처리로 버틸 수 있는 한계치를 늘리고, 필요시 과감한 인프라 투자를 검토합니다. 다만 수익이 성장 비용을 감당할 수준인지 항상 검토하여 **지속 가능한 성장**을 도모합니다.

이상으로, 무서핑 서비스 공개 게시판 및 광고 수익화 기능에 대한 제품 요구사항 및 기술 구현 방안을 상세히 설명하였습니다. **핵심 성공 요소는 사용자들이 안심하고 참여할 수 있는 양질의 커뮤니티 환경 조성**과 **서비스 운영 안정성**이라 할 수 있습니다. 본 PRD 에 따라 착실히 개발을 진행한다면, 무서핑 서비스는 사용자의 참여도를 높이고 자체 수익 모델을 갖춘 **자생적인 서비스**로 성장할 것으로 기대합니다.