

# AI 기반 유머/공포 콘텐츠 서비스

## ‘무서핑’

### - 공개 게시판 및 광고 수익화 기능 구현 기획서 (PRD)

#### 핵심 요약

- **서비스 개요:** ‘무서핑’은 AI를 활용하여 유머 또는 공포 테마의 이미지를 생성·공유하는 React/Next.js 기반 웹 서비스입니다. 현재는 프론트엔드 위주(Next.js, Vercel 배포)로 동작하며, 사용자는 AI가 생성한 귀신 캐릭터 등의 콘텐츠를 감상할 수 있습니다.
- **차기 개발 목표:** 사용자 참여를 높이고 수익 모델을 마련하기 위해 공개 게시판 기능과 Google AdSense를 통한 광고 수익화를 도입할 예정입니다. 회원 가입 없이 익명 게시판 형태로 자유롭게 게시글 작성·댓글·좋아요가 가능하도록 하고, 사이트 곳곳에 배너 광고를 배치하여 광고 수익을 창출합니다.
- **기술 스택:** 프론트엔드는 Next.js 프레임워크 (React 기반)와 TailwindCSS 등의 UI 라이브러리를 사용하여 반응형 디자인을 구현합니다. 백엔드는 현재 존재하지 않지만, 차기 기능 구현을 위해 서버리스 데이터베이스(예: Supabase, Firebase)와 API 서버(Vercel Serverless Functions) 연동을 계획하고 있습니다. 이미지 생성 AI는 OpenAI 또는 Google Gemini API 등 외부 AI API를 활용하여 구현할 가능성이 높습니다.
- **주요 고려사항:** 익명성으로 인한 보안 취약점(중복 좋아요, 스팸 등)과 콘텐츠 관리(욕설/유해 콘텐츠 필터링)가 중요한 이슈입니다. 좋아요 기능의 동시성 처리와 정렬 성능 최적화를 위해 데이터베이스 인덱싱 및 궁극적 일관성(Eventual Consistency) 모델 도입을 검토합니다. Google AdSense 정책 준수를 위해 사용자 생성 콘텐츠(UGC)에 대한 철저한 모더레이션이 필요하며, 광고 배치는 사용자 경험을 해치지 않는 선에서 신중하게 이뤄져야 합니다.
- **결론:** 제안된 기능들은 현대 웹 기술 스택으로 충분히 구현 가능하며, Vercel 기반의 CI/CD와 서버리스 아키텍처를 통해 빠른 개발 및 배포가 가능합니다. 다만 익명 서비스 특유의 악용 가능성, AI API 사용에 따른 비용 관리, 대규모 트래픽 대비 성능 및 보안 강화, 광고 및 콘텐츠 정책 준수 등의 부분에 대한 사전 대비책을 마련하는 것이 서비스 성공의 핵심입니다.

---

## 서비스 개요 및 개발 목표

**\*\*‘무서핑’\*\***은 AI가 생성한 **유머/공포 테마 콘텐츠**를 제공하는 웹 서비스입니다. 현재는 Next.js로 구현된 **프론트엔드**만 존재하며, 사용자는 AI가 그린 귀신 캐릭터 이미지 등을 확인할 수 있습니다. 서비스 이름에서 알 수 있듯이, **오싹한 공포 요소와 유머**를 결합한 독특한 콘텐츠 경험을 목표로 합니다.

**향후 개발 목표**는 사용자 참여 강화와 수익 모델 확보입니다. 이를 위해 두 가지 핵심 기능을 기획하고 있습니다:

- 1) **공개 게시판 기능**: 사용자들이 AI 생성 이미지를 포함한 게시글을 자유롭게 올리고, 서로 댓글로 소통하며, **좋아요**를 누를 수 있는 커뮤니티 게시판입니다. **로그인이나 회원가입 없이 익명**으로 이용할 수 있게 하여 진입 장벽을 낮추고 활발한 참여를 유도합니다.
- 2) **광고 수익화 기능**: Google AdSense 광고를 서비스에 통합하여 **배너 광고** 수익을 창출합니다. 페이지 상단 헤더, 사이드바, 게시글 목록 중간, 게시글 본문 등 **적절한 위치**에 광고를 배치하여, 서비스 운영에 필요한 비용을 충당하고 수익 모델을 구축합니다. 광고는 **반응형 배너**를 사용해 데스크톱과 모바일에서 모두 자연스럽게 노출될 예정입니다 .

이 제품 요구사항 문서(PRD)는 위 두 가지 기능에 대한 **상세 요구사항**과 **구현 방안**을 다루며, 추가로 서비스 **배포 전략**과 **운영 계획**까지 포괄합니다. 특히 **익명 게시판**의 보안 이슈, **AI 콘텐츠 생성**을 위한 백엔드 연동, **광고 정책 준수** 등을 중점적으로 고려합니다.

## 주요 사용자 및 인증 정책 (익명성)

‘무서핑’은 **별도 회원가입 없이** 서비스를 이용하도록 디자인되어 있습니다. 모든 방문자가 곧바로 콘텐츠를 보고, **익명으로** 게시글 작성이나 댓글 참여를 할 수 있는 **익명 게시판** 형태를 지향합니다. 이러한 익명성은 초기 사용자 유입과 참여 장려 측면에서 큰 장점입니다. 사용자들은 **별다른 신원 정보 제공 없이** 부담

없이 AI 이미지를 공유하거나 의견을 남길 수 있어, 서비스 활성화에 기여할 것으로 기대합니다.

그러나 **익명성 유지**와 동시에 서비스 악용을 방지하기 위한 **기본적인 정책**이 필요합니다. 현재 구상 중인 방안은 다음과 같습니다:

- **쿠키/세션 기반 식별자 부여:** 사용자가 게시글을 쓰거나 좋아요를 누를 때, 브라우저 **쿠키**나 **로컬 스토리지**에 임시 식별자(token)를 생성하여 저장합니다. 이 토큰을 활용해 **동일 사용자의 중복 행동**(예: 동일 게시글에 여러 번 좋아요 누르기)을 방지합니다.
- **IP 기반 제한:** 추가로, 일정 시간 내 **동일 IP 주소**에서 과도한 요청(예: 초당 수십 회 이상 좋아요 시도 등)이 발생하면 해당 IP를 일시 차단하는 **레이트 리미팅**도 고려합니다. 다만, IP 기반만으로는 한계가 있어 (여러 사용자가 같은 IP를 쓸 수 있음) 보조 수단으로 활용합니다.

**한계 및 개선 여지:** 단순 쿠키 또는 IP 기반 식별은 사용자가 쿠키를 삭제하거나 시크릿 모드(Incognito)를 쓰면 무용지물이 되고, VPN 등을 이용해 IP를 바꾸면 우회될 수 있습니다. 따라서 추후 서비스 확장 시 **소셜 로그인(OAuth)** 도입을 고려하거나, **기기 지문(fingerprint)** 기술 등을 검토하여 **익명성을 해치지 않는 범위**에서 악용을 줄이는 방법을 모색해야 합니다. 현 단계에서는 완벽한 통제는 어렵겠지만, **중복 방지 토큰+IP 제한** 조합으로 기본적인 **어뷰징 방어선**을 구축합니다.

또한, 서비스 이용 과정에서 **개인정보 수집을 최소화**하고자 합니다. 회원 체계가 없으므로 **이메일, 이름 등 개인정보는 저장하지 않으며**, 법적 요구에 따른 **개인정보 처리방침**을 명시하고 준수합니다. **쿠키 기반 분석**(Google Analytics 등)이나 **광고 식별자(AdSense)** 사용 시에도 사용자에게 필요한 동의를 얻고, **익명 데이터**로만 활용하도록 합니다.

## 기술 구현 가능성 및 주요 고려사항

이 절에서는 **프론트엔드, 백엔드, 데이터베이스, AI 연동, 성능, 보안, 광고** 등 각 측면에서 ‘무서핑’ 차기 기능 구현의 **기술적 타당성**을 분석하고 **중요 고려사항**을 정리합니다.

## 프론트엔드 및 UI/UX

**프론트엔드 프레임워크 - React/Next.js:** 현재 프론트는 **Next.js** 를 사용 중이며, 이를 유지하는 것이 적절합니다. Next.js 는 **React** 기반의 프레임워크로, **페이지 라우팅**, **SEO 친화적 SSR/SSG 지원**, **API Routes** 등 다양한 기능을 제공합니다. Vercel 에 최적화되어 있어 **배포 편의성**도 높습니다. 예를 들어, 게시판 목록 페이지를 **\*\*정적 생성(SSG)\*\***하고, 개별 게시글 페이지를 **SSR** 로 렌더링하거나 **\*\*ISR(Incremental Static Regeneration)\*\***을 적용해 성능과 최신성을 모두 잡을 수 있습니다 . 또한 **React Query** 혹은 **SWR** 등의 데이터 패칭 라이브러리를 활용하면 API 로부터 가져온 게시글/댓글 데이터의 **캐싱**과 **상태 관리**가 용이해져, 실시간 UX 개선에 도움이 될 것입니다.

**UI 라이브러리 - TailwindCSS / Chakra UI:** 디자인 및 반응형 구현을 위해 **TailwindCSS** 또는 **Chakra UI** 를 사용할 계획입니다. TailwindCSS 는 **유틸리티 클래스** 기반으로 빠르게 스타일링이 가능하고, Chakra UI 는 **미리 디자인된 컴포넌트**를 제공하여 개발 속도를 높입니다. 둘 다 React 와 궁합이 좋아, 테마에 맞는 UI 구현이 수월합니다. 공포 테마에 맞춰 **어두운 배경**, **대비 강한 폰트**, **임팩트 있는 이미지** 등이 필요할 수 있는데, TailwindCSS 의 **dark mode** 지원이나 Chakra UI 의 **테마 커스터마이징** 기능을 이용해 쉽게 스타일 조정을 할 수 있습니다.

**이미지 업로드 및 미리보기 UX:** 게시판에 사용자가 **AI 생성 이미지**를 첨부할 수 있으므로, 이미지 업로드 시 **\*\*미리보기(thumbnail)\*\***를 제공하여 사용자 경험을 개선하려 합니다. HTML5 의 File API (FileReader 등)를 사용하면 업로드한 이미지를 서버로 보내지 않고도 **클라이언트 측에서 미리보기**로 보여줄 수 있습니다. React 용 파일 업로드 컴포넌트인 **React Dropzone** 등을 활용해 구현하면 사용 편의성이 높아질 것입니다. 이 기능은 사용자가 이미지를 올린 후 **게시글 작성 화면에서 바로 확인**할 수 있어, **오류를 줄이고 만족도**를 높입니다. 또한 이미지 용량이 큰 경우 **리사이즈나 압축**을 클라이언트에서 선행하거나, 업로드 후 **CDN 최적화**를 통해 **로딩 속도**를 개선할 계획입니다 .

**반응형 디자인:** ‘무서핑’ 이용자는 PC 웹과 모바일 웹 모두 이용할 것으로 예상됩니다. 레이아웃은 **모바일에서는 단일 컬럼**으로 게시글 피드가 보이고, **데스크톱에서는 좌측에 게시글 리스트**, **우측에 인기글/미니게임/광고 사이드바**가 보이는 2 컬럼 구조를 구상하고 있습니다. TailwindCSS 의 **Responsive Utilities** 를 사용해 화면 크기에 따라 레이아웃이 전환되도록 구현합니다. 또한 **네비게이션**

바와 글쓰기 버튼 등 핵심 UI 요소도 기기별로 최적화합니다. 예를 들어, 모바일에서는 하단에 둥둥 떠있는 ‘글쓰기’ 버튼(FAB)을, 데스크톱에선 상단에 ‘새 글 작성’ 버튼을 배치하는 식입니다.

## 게시판 기능 구현 및 확장성

### 익명성 유지 및 보안: 중복 방지 로직의 한계

익명 게시판에서 **중복 투표/도배 방지**는 어려운 도전입니다. 현재 구상한 **쿠키** 또는 **세션 기반 중복 방지**는 구현이 간단하나, **기본적인 한계**가 존재합니다. 예를 들어, 특정 사용자가 마음만 먹으면 브라우저의 **쿠키를 삭제**하거나 시크릿 모드로 다시 접속해 **같은 게시글에 무한 좋아요**를 누를 수 있습니다. 혹은, **IP** 기반으로 막으려고 해도 **VPN**이나 **프록시**를 사용하면 IP를 쉽게 바꿀 수 있고, 심지어 **일부 공용 IP** (회사, 학교 등)에서는 여러 사용자가 동일 IP를 쓰기 때문에 선의의 사용자가 차단당하는 부작용도 생길 수 있습니다 .

이러한 단순 로직만으로는 **악의적 사용자**의 행위를 완전히 차단하기 어렵습니다. 예를 들어, 스크립트를 이용해 게시판에 **스팸 글 폭탄**을 투하하거나, 좋아요를 자동 반복 클릭하는 **매크로**를 돌리는 경우, 쿠키/IP 차단을 금방 우회하며 서비스 질서를 해칠 수 있습니다. 이 문제는 커뮤니티의 **데이터 신뢰성**을 떨어뜨리고 정상 이용자의 **경험을 저해**하여, 궁극적으로 서비스 **신뢰도**에 악영향을 줍니다. 특히 사용자 콘텐츠가 곧 서비스 품질로 직결되는 상황에서, 이러한 조작이 방치되면 **광고 정책 위반**(부정 클릭 유도 등)으로 이어져 **수익에도 타격**을 줄 수 있습니다 .

**개선 방안:** 근본적으로 익명성을 지키면서 악용을 막기 위해서는 **복합 대책**이 필요합니다. 우선, 서버 측에서 **요청 빈도를 제어**하는 **레이트 리미팅**을 도입합니다. 예를 들어, 한 IP 혹은 한 익명 토큰이 **분당 X회 이상의 좋아요 요청**을 보내면 해당 요청을 거부하거나 지연시키는 방식입니다. 이를 위해 **슬라이딩 윈도우 알고리즘** 등의 레이트 리미터를 적용해, 트래픽이 일정 임계치를 넘으면 **Throttle**을 거는 것이 좋습니다 . 또한, **CAPTCHA**를 도입하는 것도 고려할 수 있습니다. 사용자가 짧은 시간에 연속적으로 게시글을 올리거나 좋아요를 누르면 일정 횟수 후 CAPTCHA를 띄워봄으로써, **자동화된 공격**을 어렵게 만듭니다.

나아가, 한층 고급 방법으로 기기 지문(Fingerprint) 기술도 검토 중입니다. 브라우저의 환경 정보를 해싱하여 만든 기기 지문은 쿠키보다 사용자 단위를 조금 더 고정적으로 식별할 수 있습니다. 다만, 이는 프라이버시 이슈가 있을 수 있어 도입 전 면밀한 검토가 필요합니다. 초기 단계에서는 기본적인 쿠키+IP 제한으로 출발하되, 향후 공격 패턴 데이터가 축적되면 이러한 고급 기법을 순차적으로 도입하여 보안을 강화할 계획입니다.

## 게시글 수정/삭제 권한 관리 (익명 사용자 식별)

익명 게시판에서도 사용자가 자신이 작성한 글을 일정 시간 내 수정/삭제할 수 있게 하는 것은 사용자 편의상 바람직합니다. 현재는 게시 후 5분 이내에만 수정 또는 삭제를 허용하는 방안을 고려 중입니다. 문제는 익명 사용자를 어떻게 증명할 것인가입니다. 로그인한 서비스의 경우 사용자 ID로 본인 여부를 판단하지만, 익명 서비스에서는 게시글 작성 시점의 정보로 판단해야 합니다.

초기 방안으로는 쿠키에 글 작성자 토큰을 저장하거나, IP 기반으로 추정하는 방법이 있습니다. 예컨대, 글 작성 시 생성된 익명 토큰을 게시글 메타데이터에 함께 저장해두고, 사용자가 삭제 요청을 보낼 때 그 토큰을 확인하는 식입니다. 하지만 앞서 언급했듯 쿠키가 지워지면 이 방법은 무용지물이 되고, IP 기반은 IP 변경 시 문제가 생깁니다. 극단적이지만 가능한 악용 시나리오는: 사용자가 글을 올린 뒤 곧바로 쿠키를 지워버려 스스로도 그 글을 삭제하지 못하게 만들거나, 반대로 IP가 우연히 같아진 다른 사용자가 삭제 권한을 행사하는 일이 발생하는 것입니다. 이는 서비스 신뢰성에 중대한 문제를 야기합니다.

**해결 대안:** 많은 익명 게시판들이 채택하는 방식 중 하나는 \*\*“게시글 비밀번호”\*\*입니다. 글 작성 시 사용자가 설정한 비밀번호(짧은 PIN 같은)를 서버에 해시로 저장해두고, 수정/삭제 시 그 비밀번호를 입력하도록 하는 것입니다. 이를 도입하면 쿠키가 없어도 사용자 본인임을 검증할 수 있습니다. 물론 별도의 UX 요소가 늘어나 약간 불편할 수 있으나, 익명성을 유지하면서 권한 관리를 하는 신뢰도 높은 방법입니다.

또 다른 방법은 \*\*편집 토큰(Edit Token)\*\*을 발급하는 것입니다. 글이 게시되면 UI에 이 토큰을 한번 보여주고 복사할 수 있게 하여, 나중에 수정/삭제할 때 그

토큰을 입력하게 하는 방식입니다. 사용자 측 보관의 부담이 있지만, 보안 측면에서는 확실합니다. 만약 사용자가 토큰을 분실하면 사실상 본인도 글을 지울 수 없게 되지만, 최소한 타인이 함부로 삭제하는 상황은 막을 수 있습니다.

현 시점에서는 비밀번호 방식이 사용자 친화적이라 우선 검토되고 있습니다. 5 분이라는 제한 시간 내에서는 큰 부담 없이 기억했다 입력할 수 있기 때문입니다. 추후 운영하면서 문제가 발생하면 관리자介入을 통해 임의 삭제하거나, 토큰 방식을 추가 도입하는 등 유연하게 대처할 계획입니다.

좋아요 기능 구현: 동시성 이슈와

궁극적 일관성(Eventual Consistency)

좋아요(Like) 기능은 커뮤니티 활성화의 핵심 요소 중 하나입니다. 구현은 단순히 보이지만, 데이터 정확성과 성능 두 측면에서 세심한 접근이 필요합니다. 기본 설계는 로그인 불필요이므로 익명 사용자별 중복 방지만 처리하면 됩니다. 좋아요 수를 저장하는 방법은 크게 두 가지입니다: 게시글 테이블에 like\_count 컬럼을 두고 증가시키는 방법, 또는 별도의 Likes 테이블에 개별 좋아요 기록을 추가하는 방법입니다.

1) 즉시 증가 방식: 게시글의 like\_count 를 바로 UPDATE 하는 방식은 간단하지만, 동시성 문제를 유발할 수 있습니다. 여러 사용자가 동시에 좋아요 누르면, 일부 요청의 증가가 덮어써져 카운트 누락이 생길 수 있습니다. 이를 막으려면 트랜잭션 락을 걸거나 데이터베이스의 원자적 연산을 활용해야 합니다. 예를 들어, SQL 의 UPDATE posts SET like\_count = like\_count + 1 WHERE id = X 는 원자적으로 처리되므로 동시 업데이트에 안전합니다. 그러나 짧은 시간에 매우 빈번한 좋아요가 들어올 경우, 해당 레코드에 잠금(lock) 경합이 생겨 DB 부하와 지연이 커집니다. 인기 게시글 한 개가 서비스 전체 성능의 병목이 될 수 있는 것입니다.

2) 이벤트 기록 방식: 좋아요 클릭 자체는 이벤트로 기록만 하고, 실제 게시글의 총 좋아요 수는 나중에 집계하는 방법입니다. 즉, Likes 라는 별도 테이블에 (user\_id or token, post\_id, timestamp) 등의 레코드를 삽입해두고, 일정 주기마다(예: 5 분마다) 해당 테이블을 집계하여 게시글 테이블의 like\_count 를 갱신합니다. 이 접근은 궁극적 일관성(Eventual Consistency) 모델에

해당합니다 . 즉, 좋아요 수가 즉각 오르내리지는 않아도, **약간의 시간 차이를 두고 결국 정확히 반영**되는 것을 목표로 합니다. 장점은, 실시간으로 카운트를 맞추려 DB 를 자주 변경하지 않아 **쓰기 부하가 크게 분산**된다는 것입니다. 또한 각 좋아요 이벤트는 독립 레코드라 **동시성 충돌이 거의 없음**도 이점입니다. 단점은 사용자가 즉시 변동을 못 볼 수 있다는 점이지만, 대다수 SNS 에서도 좋아요 수는 몇 초~수십 초 단위로 지연되어 업데이트되는 것이 일반적이므로 사용자들은 크게 개의치 않습니다 .

**채택 방안:** 우리 서비스는 초기 트래픽이 크지 않을 수 있으므로 1) 방식으로 시작할 수 있지만, **성장에 대비**하여 2) 모델을 염두에 두고 설계하는 것이 바람직합니다. 구현 전략은, **좋아요 클릭 API** 에서 먼저 Likes 테이블에 추가를 하고, **클라이언트에는 즉각 임시적으로+1 된 수치를** 보여주며, 백엔드 배치가 실제 값을 동기화하면 값이 튀지 않게 처리하는 것입니다. 이러한 구조는 **대용량 환경에서도 확장성**이 높고, 잠재적 버그(중복 카운트 등) 발생 시 로그를 재처리하여 복구하기도 수월합니다. 또한 **유저별 좋아요 중복 방지**도 Likes 테이블에서 UNIQUE(post\_id, user\_token) 제약을 주면 자연스럽게 해결됩니다 .

## 대규모 트래픽 대비: 레이트 리미팅 전략

위 **익명성 보안** 파트에서 언급한 레이트 리미팅을 좋아요 기능 맥락에서 좀 더 상세히 다룹니다. 만약 악의적인 스크립트로 특정 게시글에 **수천 번의 좋아요 폭탄**을 날리면, 백엔드 API 서버나 DB 에 심각한 부하가 가해질 수 있습니다. 이 경우 서비스 지연 또는 장애로 이어져 **정상 사용자 피해**가 큼니다.

**\*\*레이트 리미팅(rate limiting)\*\***은 이러한 상황을 방지하는 핵심 기술입니다. 일반적으로 **IP 별, API 키별, 세션별** 등 식별자 단위로 초당/분당 **최대 요청 횟수**를 정해두고, 그 한계를 넘는 요청은 **거부**하거나 **지연**시킵니다 . 우리 서비스에서는 로그인 사용자가 없으므로 **IP 주소와 익명 토큰** 조합으로 제한을 거는 것이 현실적입니다. 예를 들면, “같은 IP 내에서 같은 게시글 좋아요는 1 분에 1 회만 허용” 또는 “익명 토큰별로 좋아요 API 호출은 분당 10 회로 제한” 등의 정책을 수립할 수 있습니다.

이를 구현하기 위해 Vercel Serverless Functions 레벨에서 **메모리 캐시**나 **KV 스토어**(예: Upstash Redis) 등을 활용해 요청 카운트를 추적합니다. **슬라이딩**



**윈도우 알고리즘**은 고정된 1 분 간격이 아닌 **최근 60 초 기준**으로 윈도우를 움직이며 계산하기 때문에, 만약 60 초간 10 회를 넘었다면 그 초과 부분만큼 대기하게 하는 등 **유연한 제어**가 가능합니다 . 이러한 알고리즘은 **트래픽 폭주**에도 비교적 **완만하게 대응**하여 서비스가 버티는 시간을 벌어줍니다.

또한, Vercel 환경에서는 서버리스 함수에 너무 많은 요청이 몰릴 경우 **동시 인스턴스**가 폭증하여 비용이 늘거나 **Cold Start 지연**이 누적될 수 있으므로, 레이트 리미팅으로 **비정상 패턴**을 걸러주는 것이 비용 관리 측면에서도 중요합니다. Cloudflare Turnstile 등의 **비대화식 CAPTCHA** 를 통합해 레이트 리미트에 걸린 사용자의 요청만 추가 검증하는 것도 사용성을 해치지 않으면서 보안을 높이는 아이디어입니다.

## 게시글 정렬 기능 최적화 (최신순 & 인기순)

게시판의 기본 정렬 옵션은 **최신순**과 **인기순** 두 가지를 제공하려 합니다. 최신순은 단순히 **작성 시각 내림차순**으로 정렬하는 것이고, 인기순은 **좋아요 수 + 최근 작성 여부**를 조합한 가중치를 기준으로 상위 게시글을 보여주는 방식입니다. 월간 베스트 등 기간 제한 인기글도 추후 도입할 수 있습니다.

**데이터베이스 인덱싱:** 대량의 게시글 데이터에서도 빠르게 정렬하려면, 데이터베이스 레벨 최적화가 필수입니다. **최신순**의 경우 게시글 테이블의 `created_at` 타임스탬프 칼럼에 **\*\*인덱스(index)\*\***를 생성하고 내림차순 쿼리를 실행하면, 전체 스캔 없이도 최근 N 개를 빠르게 가져올 수 있습니다. **인기순(좋아요 순)**도 마찬가지로 `like_count` 칼럼에 인덱스를 걸어두면 내림차순 정렬 속도가 향상됩니다 . 다만, 인기순은 보통 **동률 정렬 기준**이 추가로 필요합니다(예: 좋아요 같으면 최신 글 우선). 이를 쿼리로 처리할 때는 `ORDER BY like_count DESC, created_at DESC` 식으로 다중 정렬을 사용하게 되는데, 이 경우 (`like_count, created_at`) **복합 인덱스**를 정의하면 더욱 효율적입니다 .

**캐싱 전략:** DB 부하를 줄이기 위해 **캐싱**도 고려합니다. 예를 들어, 메인 화면의 “월간 베스트” 게시글 목록은 매 시간 한 번만 쿼리하고 결과를 캐싱하여, 해당 시간 내에는 캐시된 데이터를 서빙하면 DB 쿼리를 대폭 줄일 수 있습니다. Next.js ISR 기능을 이용해 해당 섹션을 일정 주기(예: 1 시간)로 재생성하도록 설정하면, 코드 단에서 신경 쓰지 않고도 자동 캐싱 효과를 낼 수 있습니다.


**쿼리 최적화 일반 원칙:** 또한, 개발 단계에서 **SQL 쿼리 실행 계획**을 확인해 **필요한 칼럼만 SELECT** 하고, 불필요한 테이블 조인은 없도록 쿼리를 단순화할 것입니다. 인기순 조회 시 Like 테이블과 조인하는 대신 게시글 테이블에 집계된 like\_count 를 쓰는 것도 그러한 최적화의 일환입니다. **LIMIT** 절을 적극 활용해, 한 번에 너무 많은 게시글을 불러오지 않고 페이지네이션 또는 무한 스크롤로 **부분 로딩**하는 것도 서버 부담을 낮춰줍니다.

이러한 조치는 게시글 수가 수만 건 이상으로 늘어나더라도 **사용자에게 빠른 응답**을 제공하고, **데이터베이스 비용을 절감**하는 효과가 있습니다. 초기 구축 시부터 인덱싱과 쿼리 구조를 잘 설계해두면, 향후 트래픽 증가에 안정적으로 대비할 수 있습니다.

## 욕설 필터링 시스템 고도화

게시판과 댓글 시스템에는 **욕설/비속어 필터링**이 필수적으로 들어갑니다. 현재 고려 중인 솔루션은 **오픈소스 라이브러리인 badwords-ko** (한국어 욕설 리스트 기반)와 **bad-words** (영어 등) 등을 활용하여, 금칙어가 발견되면 해당 단어를 \*\*\* 등의 문자로 치환하거나 댓글 자체를 차단하는 방식입니다. 클라이언트(JavaScript) 단과 서버(API) 단에 모두 필터를 적용해 **이중 체크**를 할 계획입니다.

하지만 **사전 리스트 기반** 필터링에는 잘 알려진 **한계**가 존재합니다. 사용자는 의도적으로 **우회 표현**을 사용하여 필터를 뚫으려 할 것입니다. 예를 들어:

- 금지어를 일부 **특수문자**로 치환 (시@발, 개갈\*\* 등)
- 자음을 분리하거나 유사한 다른 문자로 대체 (ㅅㅞ ㄱ1, 씨ㄴ)
- 정상 단어처럼 보이도록 변형 (피에로를 욕설의 일부로 쓰는 등)
- 욕설 중간에 공백이나 이모지를 삽입 (욕  설)

이러한 경우 단순 문자열 일치로는 탐지하기 어렵습니다 . 또한 반대로, 필터가 과도하게 작동하면 **오탐지(False Positive)** 문제도 생깁니다. 예컨대 “이거 \*어 먹지 마”처럼 정상적인 문장에서 일부 글자가 금칙어로 인식되어 걸려지면 사용자 불편이 큼니다 .

**고급 대응:** 이러한 우회 및 오탐 문제를 해결하기 위해 **정규표현식(Regex)** 전처리와 **딥러닝 AI 모델** 활용을 검토합니다. 첫째, 필터링 전에 댓글 내용을 **정규화**하는 방안을 고려합니다. 한국어의 경우 초성/중성 분리된 것을 결합하거나, 특수문자를 제거/대체하는 전처리를 하면 필터의 탐지율이 올라갑니다. 예를 들어 “새끼1” → “새끼”로 복원하거나, “시@@발” → “시발” 식으로 정규화한 뒤 사전 대조를 할 수 있습니다. 이러한 과정은 정규표현식과 유니코드 처리를 통해 구현 가능합니다.

둘째, 궁극적으로는 **머신러닝 기반 욕설 판별 모델**을 도입하는 것입니다. 한국어 욕설 탐지에는 **자모 단위의 CNN 모델**이나 **문장 임베딩+LSTM 모델** 등이 연구되어 왔으며, **90% 이상의 높은 정확도**를 보이는 것으로 알려져 있습니다 . 딥러닝 모델은 금칙어 리스트에 없더라도 **문맥과 패턴**을 학습하여 욕설을 분류해내므로, 지속적으로 새로운 패턴이 생기는 인터넷 환경에 유연하게 대응할 수 있습니다 . 다만, 이러한 모델을 직접 학습하려면 많은 데이터셋과 리소스가 필요하므로, 오픈소스로 공개된 사전 학습 모델을 사용하는 방안을 찾고 있습니다. (예: 카카오나 Naver 등이 공개한 욕설 필터 모델이 있는지 검토)

**운영 전략:** 필터링 결과에 따라 **자동 처리**와 **로그**를 적절히 남길 것입니다. 예컨대, **심한 욕설**이나 **혐오 표현**이 감지된 댓글은 **즉시 차단**하고 DB에 저장하지 않습니다. 사용자에게는 “부적절한 표현으로 인해 댓글이 등록되지 않았습니다”라는 메시지를 보여줄 수 있습니다. 동시에 해당 내용을 **관리자용 로그**에 기록해두어, 추후 검토 및 모델 개선에 활용합니다.

또한 **경미한 욕설**(예: 친구끼리 쓰는 가벼운 비속어 등)은 차단 대신 **\*\*** 등으로 마스킹하여 게시할 수도 있습니다. 이때 마스킹된 사실을 사용자에게 안내하여 **자정 활동**을 유도합니다 (“OO 단어가 **\*\*\***로 표시되었습니다”). 이러한 피드백 루프는 사용자들이 서비스 규칙을 인지하고 스스로 조심하도록 하는 효과가 있습니다.

**실 사례 참고:** 2017년 이후 국내 대형 플랫폼들도 유사한 시스템을 운영 중입니다. 카카오의 경우 뉴스 댓글, 특 채널 등 다양한 서비스에 **자동 욕설 감지 및 음표(ㄴ) 치환** 기술을 적용해 욕설 노출을 감소시켰고, 이로써 **운영자들의 정신적 피로도** 줄었다고 알려졌습니다. 또한 **\*\*한국인터넷자율정책기구(KISO)\*\***에서 네이버·카카오와 협력하여 구축한 **\*\*욕설**

필터링 API 서비스(KSS)\*\*는 2023 년 기준 37 개 기업·기관이 도입하여 1788 만 건 중 33 만 건의 욕설을 걸러내는 성과를 거두었습니다 . KISO 는 특히 특수문자 섞인 변형 욕설 탐지에 주력하고 60 만 건의 욕설 DB 를 80 만 건으로 확대하는 등 지속 업데이트하고 있어 필터 성능이 향상되었습니다 . 이런 사례들처럼, 우리 서비스도 초기에는 간단한 필터로 시작하지만 지속적인 고도화와 외부 자원 활용으로 쾌적한 커뮤니티 환경을 만들어갈 것입니다.

## AI 콘텐츠 생성 및 백엔드 연동

‘무서핑’의 핵심인 AI 이미지/콘텐츠 생성 기능을 구현하기 위해, 외부 AI API 연동 및 그에 따른 백엔드 구성이 필요합니다. 현재 프론트엔드만 존재하므로, 사용자의 콘텐츠 생성 요청(예: “웃긴 귀신 그림 만들어줘”)을 처리하려면 백엔드 서버 또는 서버리스 함수가 AI 모델 API 를 호출하고 결과를 받아와야 합니다.

**AI 모델 선택:** 고려 중인 옵션은 OpenAI API(예: DALL·E 이미지 생성, GPT-4 텍스트 생성 등)와 Google Gemini API 입니다. OpenAI 의 경우 비교적 안정된 상용 API 로서 이미지 생성과 글 생성 모두 지원하므로, 예측 가능하고 품질 보장이 됩니다. Google 의 Gemini(차세대 멀티모달 AI)도 공개 시점에 따라 도입 가능성을 열어두고 있습니다. 이외에도 Stability AI 의 Stable Diffusion 모델을 Replicate 나 HuggingFace Hub API 를 통해 사용하는 방법도 있습니다 .

**API 연동:** 프론트엔드에서 직접 OpenAI 호출을 하지는 않고, API 키 보안을 위해 백엔드에서 호출합니다. Next.js 의 API Routes 또는 Vercel Serverless Function 를 생성하여 /api/generateImage 등의 엔드포인트를 만들고, 여기서 OpenAI 나 Gemini API 를 호출해 이미지를 생성한 뒤 URL 혹은 base64 데이터를 프론트로 전달합니다. 이 작업은 비동기로 시간이 걸릴 수 있기 때문에, 사용자 경험 측면에서 로딩 스피너나 “AI 가 열심히 그리고 있어요...” 같은 메시지를 표시합니다. 평균 생성 시간은 모델과 옵션에 따라 수 초에서 수십 초 정도인데, 10 초를 넘어갈 경우 Progressive update(예: 5 초마다 상태 업데이트)도 고려합니다.

**백엔드 인프라:** 현재 백엔드가 없는 상태이므로, 작은 규모에서는 서버리스 함수로 충분하나 요청이 잦아지면 별도 서버가 필요할 수 있습니다. 우선은 Vercel Serverless 로 구현하고, 필요 시 Node.js 익스프레스 서버나 Cloud Run

등에 마이크로서비스를 띄워 처리하도록 확장성 있게 설계합니다. 백엔드가 생기면 이 기회를 이용해 **게시판 DB 연동**과 **AI API 호출**을 모두 백엔드에서 관리하게 구조를 개편하면 좋습니다. 즉, **Supabase** 등의 DB를 Node 서버에서 제어하고, AI 생성 요청도 해당 서버에서 대행하여, 프론트엔드는 오로지 이 서버와 통신하는 식입니다.

**비용 및 성능:** OpenAI API는 **호출당 과금**(예: 이미지 1장 생성 \$0.02 등)이 발생하므로, 무분별한 사용을 막기 위해 **쿼타**를 둘 필요가 있습니다. 예를 들어 “하루에 5회까지 생성 가능” 또는 “포인트 제도 도입” 등을 고려합니다. 또한, **NSFW**나 **잔혹한 이미지**가 생성되지 않도록 OpenAI API의 **콘텐츠 필터**를 활용하거나, 직접 **검열 로직**을 추가해야 합니다 (예: 출력 결과에 유해 요소가 있으면 표시하지 않음).

**향후 발전:** AI 콘텐츠 생성 결과를 **게시글**로 바로 공유할 수 있게 하여, 생성-공유가 원스텝으로 이루어지게 만들 계획입니다. 이 경우에도 결과 이미지를 먼저 **\*\*임시 저장(예: Supabase Storage)\*\***하고 게시글 데이터와 함께 저장해야 하므로, **백엔드의 파일 처리 능력**도 함께 개발될 예정입니다.

요약하면, AI 기능 구현은 **외부 강력한 API의 힘**을 빌려 충분히 가능하며, 백엔드를 신규 개발해야 하는 부담이 있지만 **서버리스 아키텍처**로 시작하면 빠르게 구축할 수 있습니다. OpenAI/Gemini 등 API 연동 후, 우리만의 **AI 모델 튜닝**이나 **프롬프트 엔지니어링**을 통해 콘텐츠의 독창성과 품질을 높이는 방향으로 나아갈 것입니다.

## 백엔드 및 데이터 저장소

**서버리스 데이터베이스 선택 및 한계:** Vercel KV, Supabase, Firebase

현재 서비스에는 전용 백엔드 DB가 없으므로, 서버리스 데이터베이스 도입을 검토하고 있습니다. 후보로 떠오른 것은 **Vercel KV, Supabase, Firebase** 등입니다. 각 옵션별 특징을 비교하면 다음과 같습니다:

- **Vercel KV (Upstash Redis 기반):** Vercel 에서 제공하는 키-값 스토어로, 글로벌 엣지 환경에 최적화되어 지연이 짧습니다. 구조가 단순하여 캐시나 카운터(예: 조회수, 좋아요수)용으로 적합합니다. 읽기/쓰기가 빠르지만, 데이터가 휘발될 수 있고 복잡한 조회에 부적합하므로 영구 저장소로는 한계가 있습니다.
- **Supabase (PostgreSQL 기반):** 오픈소스 Firebase 라고 불리는 Supabase 는 Postgres SQL DB 와 인증, 스토리지 등을 제공하는 풀스택 BaaS 입니다. 관계형 데이터를 다루기에 좋고, SQL 친숙한 개발자에게 편합니다. 또한 실시간 기능(Realtime DB)도 제공하여, 게시글/댓글 변화가 실시간 반영되게 할 수 있습니다. 다만 Supabase 는 서버 위치가 한정되어 있어 (미국 등) 우리 서비스의 주요 사용자 층(한국)에 대한 지연을 살펴봐야 합니다.
- **Firebase (Firestore/RealtimeDB):** NoSQL 문서형 DB 로, 모바일에서 흔히 쓰이는 Google 의 BaaS 입니다. **Firestore** 는 쿼리가 비교적 유연하고 확장성이 좋으며, **Realtime Database** 는 간단한 트리구조 데이터의 초실시간 동기화에 강점이 있습니다. 초반에는 무료 Spark 플랜으로 시작할 수 있고, 추후 구글 클라우드로 자연스레 확장 가능합니다. 단, NoSQL 이라 데이터 모델링에 신경써야 하고, 쿼리당 비용이 발생하므로 사용 패턴에 따라 비용 예측이 어려울 수 있습니다.

**콜드 스타트 문제:** Vercel 의 서버리스 함수는 내부적으로 AWS Lambda 를 사용하며, 이는 일정 기간 요청이 없으면 **인스턴스가 종료**되고, 다음 요청 시 **재기동(Cold Start)** 되어 응답이 지연되는 현상이 있습니다. 이 현상은 DB 연결에도 영향을 주는데, 예를 들어 Supabase(Postgres)에 연결하는 경우 Lambda 가 새로 뜰 때마다 **DB 커넥션 설정**에 수백 ms 추가 지연이 생길 수 있습니다. 트래픽이 꾸준하면 문제가 없지만, 간헐적인 사용 패턴에서는 첫 요청의 **레이턴시 증가**로 나타납니다 .

이를 완화하기 위한 방안으로 **Provisioned Concurrency** (일정 수의 Lambda 를 항상 대기상태로 유지)나, **Lambda 워머**(주기적으로 함수 호출하여 깨우는) 기법이 있습니다. 그러나 전자는 비용이 늘고 후자는 정확성이 떨어집니다. 대신, **데이터베이스 연결 자체를 캐싱**하거나 **DB Connection Pool** 을 활용하는 것도 고려됩니다. Supabase 의 경우 **커스텀 연결**이 어렵지만, Firebase 는 연결을 관리하는 SDK 가 비교적 콜드스타트 영향을 적게 받습니다. Vercel KV 는 애초에 HTTP 호출로 작동하여 지연이 매우 낮습니다.

**성능 vs 비용:** 서버리스 DB 는 초기 비용 부담이 없고 **자동 확장**되어 편하지만, 반대로 **용량/호출량 증가에 따른 비용 급등** 위험이 있습니다 . 또한, Lambda 는

15 분 실행 제한, CPU/메모리 제한 등이 있어, 대용량 처리나 영상 생성 같은 작업에는 부적합합니다. 이런 경우 별도 서버나 Cloud Run 등을 혼용하게 될 것입니다.

결론적으로, 서비스 초기에는 **Supabase** 로 관계형 DB 를 쓰고, 간단한 카운터나 캐시는 **Vercel KV** 를 이용하는 **혼용 전략**이 유력합니다. Firebase 도 좋은 선택이지만, 기존 React/Next.js 스택과의 궁합 및 SQL 사용 편의성 측면에서 Supabase 가 약간 우세합니다. 다만, 지속적으로 서버리스 환경의 **콜드스타트 모니터링**을 하고, 유저가 늘면 **세션 지속 서버**로 마이그레이션하는 계획도 비추해두겠습니다.

## 각 DB 서비스의 무료 플랜 한계 및 비용 관리

**Supabase:** 무료 플랜은 위에서 언급했듯 **50,000 MAU, 500MB DB, 5GB bandwidth, 1GB Storage** 등이 포함됩니다 . 초기에는 충분하지만, 예를 들어 이미지 파일 저장을 많이 하면 1GB 는 금방 찰 수 있고, 대역폭 5GB 도 이미지/동영상 전송이 많아지면 빠듯할 수 있습니다. Pro 플랜으로 가면 월 \$25 로 **8GB DB, 100GB storage, 250GB bandwidth** 가 제공되며, 추가 초과분에 대해 **저렴한 종량 과금**이 붙습니다 . 따라서 일정 사용자 규모까지는 예측 가능한 비용으로 운영 가능하나, **예상치 못한 쿼리 폭증이나 읽기 폭주**가 생기면 bandwidth 초과 등으로 비용이 증가할 수 있으므로 **모니터링 알림 설정**이 필요합니다.

**Firebase:** Spark(무료) 플랜에서 **Firestore** 는 1GB 저장, 일일 50k 읽기/20k 쓰기 등의 제한이 있습니다 . 스토리지는 5GB, 월 1GB 다운로드 무료 등이고, 프로젝트 생성도 계정당 5~10 개 제한이 있습니다 (정확한 수치는 Firebase 정책에 따라 달라질 수 있음). 초과 시 **\*\*Blaze(종량제)\*\***로 전환되어 사용량만큼 과금됩니다. 예를 들어 Firestore 는 저장 용량 \$0.18/GB/월, 읽기 \$0.06/10 만 회, 쓰기 \$0.18/10 만 회 등의 요금이 있습니다 . 큰 커뮤니티로 성장하면 이 비용이 적지 않기에, **데이터 구조 최적화**와 **쿼리 호출 줄이기**(예: 캐시, 정적 페이지 활용)가 중요합니다.

**Vercel KV:** Hobby 프로젝트에서는 **개인 비상업용**으로만 사용 가능하며, 실제 상업 서비스라면 Pro 이상 플랜이 필요합니다. Vercel 의 Pro 플랜 자체는 월

\$20 부터로, KV 사용에 추가 과금은 크지 않은 것으로 알려졌습니다. Upstash Redis 가격 모델상 월 만 건 단위의 읽기/쓰기는 무료 구간이 넉넉하고, 이후에도 비교적 저렴합니다. 다만 Vercel KV 는 아직 베타 성격이라 안정성을 지켜보아야 합니다.

**비용 모니터링:** 세 서비스를 혼합 사용할 경우 **모니터링 지표**가 분산될 수 있습니다. 이를 위해 통합 대시보드나 경보를 설정해, **Supabase** **쿼리량/스토리지**, **Firebase** **일별 사용량**, **Vercel** **함수 호출수** 등을 매일 체크할 계획입니다. 최악의 경우 예산을 넘길 시 자동으로 서비스가 중지되거나, 일정 임계치 이상 기능(예: AI 생성) 사용을 제한하는 **셀프 디그레이드** 전략도 고려합니다.

정리하면, 무료 플랜 한도 내에서 시작하되 **성장 곡선**을 항상 주시하고, 비용이 급증 조짐이 보이면 **상위 플랜 업그레이드** 또는 **아키텍처 변경**(예: 자체 서버 구축) 등 결정을 신속히 내릴 체계를 갖추겠습니다.

## 이미지 파일 저장소 및 전송 최적화

AI 가 생성한 이미지나 사용자가 업로드한 이미지를 저장하고 제공하기 위해 **파일 스토리지** 선택이 필요합니다. 앞서 DB 항목에서 소개한 **Supabase Storage**(S3 호환)나 **Vercel Blob Storage** 가 후보입니다.

- **Supabase Storage:** 무료 1GB 저장, Pro 100GB 제공, 추가시 \$0.021/GB 정도로 저렴합니다 . Supabase SDK 를 통해 업로드/다운로드를 관리할 수 있고, **URL 기반 접근**도 지원됩니다. 다만 글로벌 엣지 전송이 기본으로 되는지는 확인이 필요합니다 (S3 기반이므로 리전 전송일 수 있음).
- **Vercel Blob Storage:** 2023 년에 베타 출시된 Vercel 의 스토리지로, Vercel CDN 과 바로 연결되어 **전 세계 엣지 캐싱**을 활용합니다. Hobby 에 10GB 데이터 전송 포함, 이후 GB 당 소액 과금이 있습니다. Blob Storage 는 **간단한 API**로 파일을 올리고 URL 을 받는 식이라 사용이 편리합니다. 단, 아직 일반적 S3 대비 생태계가 제한적입니다.

**CDN 활용:** 이미지는 사용량이 폭증할 수 있으므로, **CDN(Content Delivery Network)** 최적화가 필수입니다. 다행히 Vercel 플랫폼 자체가 정적 파일에 Cloudflare CDN 을 적용해주므로, Blob Storage 나 프로젝트 내 public



디렉토리에 둔 이미지 파일은 대부분 캐시 히트로 빠르게 전달됩니다. 또한, **Next.js 컴포넌트**를 사용하면 **자동으로 크기별 최적화**된 이미지를 제공하고, 브라우저 **Lazy Loading**도 기본 지원하여 초기 로딩에 부담을 줄입니다.

추가적으로, 이미지 최적화 기법으로 **WebP**나 **AVIF** 같은 최신 포맷으로 변환하여 전송하면 용량을 40~80% 줄일 수 있다는 통계가 있습니다. EXIF 메타데이터 제거, 품질 압축 등도 고려할 사항입니다. Vercel or Cloudinary 등의 서비스로 이러한 처리를 자동화할 수 있습니다.

**데이터 전송 비용:** 이미지 트래픽이 증가하면 스토리지 비용보다 **대역폭 비용**이 주요해집니다. Supabase 5GB/250GB 무료 전송이 있지만, 예를 들어 이미지 하나 1MB 라고 할 때 하루 5천 조회면 5GB, 한달 150GB 를 소모합니다. 금방 무료 구간을 넘길 수 있으므로, 인기 이미지는 CDN 에 확실히 캐싱되고, **중복 이미지 업로드 방지** 등으로 낭비를 막아야 합니다.

또한, 사용자가 올리는 이미지에 **용량 제한**(예: 5MB 이하)을 두고, 업로드 시 **화면 해상도에 맞게 리사이즈**하도록 유도할 것입니다. 이는 저장 공간도 아끼고 사용자 로딩도 빠르게 하는 일석이조의 효과입니다.

결론적으로, 초기에는 Supabase Storage 로 통일하여 관리 단순성을 취하되, **전송량 추이에 따라 Vercel CDN 활용** 여부를 검토합니다. 필요한 경우 Cloudflare Images 나 Cloudinary 와 같은 **전문 이미지 CDN 서비스** 도입도 가능성 열어두겠습니다.

## 광고 수익화 통합 및 정책 준수

### Google AdSense 광고 배치 전략 및 UX 고려

**광고 배치 계획:** PRD 상 제안된 위치는 다음과 같습니다: **상단 네비게이션 바 아래 가로 배너**, **사이드바 고정 영역 스카이라이더 배너**, **게시글 목록 중간 네이티브 광고 형태**, 그리고 **게시글 본문 중간 삽입 배너** 등입니다. 이러한

배치는 사용자 시선이 많이 가는 지점에 광고를 노출하여 수익률을 높이려는 의도입니다. 특히 **본문 중간 광고**는 긴 글을 읽을 때 자연스럽게 노출되어 효과가 좋다고 알려져 있습니다. **사이드바 고정 광고**는 스크롤해도 항상 보이므로 **노출 횟수**를 극대화해 CPM 수익에 기여할 수 있습니다.

**반응형 및 자동 광고:** AdSense에서는 **반응형(adaptive)** 광고 코드를 제공하여, 화면 폭에 따라 자동으로 크기가 조절되는 광고를 게재할 수 있습니다. 이를 활용하면 데스크톱에서는 728x90 이던 것이 모바일에서는 320x50 등으로 알아서 표시되어 편리합니다. 또한 **Auto Ads(자동 광고)** 기능을 활성화하면, 구글의 알고리즘이 페이지 구조를 분석해 최적의 광고 위치에 자동으로 광고를 추가합니다. Auto Ads는 편의성은 높지만, 우리가 원하는 레이아웃과 다르게 예측 못 한 위치에 넣기도 하므로, 초기에는 **수동 배치**로 시작하고 나중에 일부 Auto Ads 병행을 검토합니다.

**사용자 경험(UX):** 중요한 것은 **광고와 콘텐츠의 균형**입니다. 광고를 많이 넣으면 수익이 오를 것 같지만, 과하면 사용자 **이탈**을 부릅니다. 일반적으로 **컨텐츠 대비 광고 비율**이 30%를 넘지 않도록 권장됩니다. 예를 들어 한 화면에 카드 뉴스 3개와 광고 1개 정도까지는 괜찮지만, 스크롤마다 광고가 보이면 거부감이 큼니다. 또한 **광고의 형태**를 콘텐츠와 명확히 구분해야 합니다. AdSense 정책상도 주변 콘텐츠를 모방한 광고 배치는 금지되어 있습니다. 따라서 광고 앞뒤에는 “광고” 또는 “Sponsored” 레이블을 붙이고, 시각적으로도 카드 스타일을 살짝 다르게 해 사용자가 **헛갈리지 않게** 합니다.

모바일 사용자의 경우 화면이 작기에 **과도한 배너**는 더 치명적입니다. 그래서 모바일에선 **팝업형 광고**는 지양하고, 배너도 하나 정도만 아래쪽에 깔리도록 (앵커 광고) 하는 편이 좋습니다. AdSense는 **\*\*앵커 광고(화면 하단 고정)\*\***와 **벗어나는 광고(스크롤시 따라오는 작은 광고)** 등을 제공하는데, 이것도 **사용자 편의와 수익의 트레이드오프**라 추후 실험을 통해 최적점을 찾아야 할 것입니다.

**광고 최적화 팁:** 업계 사례에 따르면, **모바일 최적화**가 중요합니다. 한 통계에서는 **모바일에서의 광고 단위 최적화**만으로도 매출이 증가한 사례가 있다고 합니다. 또한 **광고 색상/스타일**을 사이트 테마와 어울리게 (그러나 너무 섞이지 않게) 조정하면 클릭률이 올라가기도 합니다. AdSense는 네이티브 광고 스타일 편집을 지원하므로, ‘무서핑’의 공포 분위기에 맞춰 차분한 톤으로 꾸밀 수 있습니다.

결론적으로, 광고 배치는 초기에는 보수적으로 시작하여, 유저 피드백과 수익 데이터를 보면서 **점진적으로 최적화**할 계획입니다. 사용자 설문이나 행태 분석을 통해 “광고가 너무 많다”는 불만이 나오면 즉시 조정하여 **유저 이탈**을 막는 것을 최우선시하겠습니다.

## Google AdSense 승인 조건 및 정책 위반 방지

AdSense 를 달기 위해서는 **Google 의 승인을 받는 것부터** 선결과제입니다. 승인 요건을 충족하고, 이후에도 정책을 준수해야 **계정 정지** 등의 불상사가 없습니다.

**승인 조건 충족:** 우선, **사이트 콘텐츠의 충실도**가 중요합니다 . 구글은 **\*\*“가치 있는 고유 콘텐츠”\*\***가 있는 사이트에만 광고를 허용합니다. 따라서 ‘무서핑’에 충분한 양의 게시글과 볼거리가 있어야 합니다. 일반적인 가이드로는 **최소 5~10 개 이상의 자체 콘텐츠 페이지**가 필요하다고 합니다 . 현재 AI 생성 콘텐츠만 있다면, 이를 소개하는 설명글이나 사용자가 쓴 후기 등 **텍스트 콘텐츠**도 보강하는 것이 좋습니다. 또한 **저작권 문제가 없는 콘텐츠**이어야 합니다. AI 생성물은 저작권 문제가 상대적으로 적지만, 혹시 사용자가 외부 이미지를 올리다면 **저작권 침해 게시물**은 신속히 제거하는 프로세스가 있어야 합니다.

기술적으로는 **사이트 소유 확인**을 위해 <head> 태그에 AdSense 코드 스니펫을 넣어야 하는데, Next.js에서는 \_document.js 나 \_app.js 에서 쉽게 추가 가능합니다. **만 18 세 이상**의 계정 소유자 요건 등도 충족해야 합니다 (운영자가 해당 연령임을 확인).

**정책 위반 요소 점검:** AdSense 프로그램 정책은 매우 광범위합니다. 특히 **금지 콘텐츠**(성인물, 폭력, 증오, 위험행위 조장 등)에 광고가 붙으면 안 됩니다 . 공포 콘텐츠 자체는 **폭력성 여부**에 따라 해석될 수 있으므로, **지나치게 잔혹한 이미지**는 광고 게재 페이지에서 제외하는 등 조치가 필요할 수 있습니다. 또한, **\*\*사용자 생성 콘텐츠(UGC)\*\***이므로 게시판에 **어떤 글이 올라올지 예측 불가**입니다. 명예훼손, 혐오 발언 등이 올라오면 정책 위반이 될 수 있으므로 **모니터링**을 철저히 해야 합니다.

**광고 배치 규칙:** 기술적으로 지켜야 할 규칙도 있습니다. 예를 들면, **광고와 인터랙티브 콘텐츠(게임 등)** 간에는 앞서 언급한 **150px 이상의 거리**를 뒤편하고, **광고를 사용자에게 오해 살 만한 방식**(예: “다운로드” 버튼처럼 꾸미거나, 메뉴로 오인하게)으로 배치하면 안 됩니다 . 우리 서비스에서도 **미니게임을** 고려하고 있는데, 게임 canvas 근처에 광고를 붙이지 않도록 해야 합니다.

또, **AdSense 코드 자체를 변경**하거나 리프레시를 조작하는 행위는 금지되어 있습니다 . 반드시 Google 이 제공한 코드를 그대로 써야 하며, 광고를 임의로 새로고침하는 스크립트를 삽입하면 정책 위반입니다.

**무효 트래픽 방지:** 스스로 광고를 클릭하거나, 사용자들에게 “광고 많이 눌러주세요”라고 유도하는 행위도 절대 금물입니다 . Google 은 자동 시스템과 인력을 통해 **부정 클릭/노출**을 감지하고, 심하면 계정을 차단합니다 . 따라서 운영진이나 지인들이 호의로 광고를 자주 눌러주는 일도 없어야 하고, 외부에서 트래픽을 구매하여 강제 노출시키는 행위도 피해야 합니다.

**준수 대책:** 이러한 정책들을 준수하기 위해, **콘텐츠 가이드라인**을 마련하여 커뮤니티에 공지하고, 위반 시 삭제/차단 조치를 명시합니다. 또한 AdSense 에서 **정책 위반 경고**가 오면 즉시 해당 내용을 검토하여 조치하고, **필요 시** **이의신청**을 통해 오해를 풀도록 해야 합니다. UGC 기반 사이트는 Google 측에서도 **민감하게 보는** 편이므로, 처음부터 **안전장치**(필터, 신고, 관리자 상주 등)를 잘 갖추는 것이 승인을 수월하게 하고 지속적인 운영을 가능케 할 것입니다 .

## 광고 로딩 속도 최적화 및 광고 차단 대응

**로딩 속도 영향:** 광고 스크립트는 페이지 로딩 속도에 영향을 주는 요소입니다. AdSense 의 `<script async src="...adsbygoogle.js">` 스크립트를 넣으면 비동기로 로드되어 그나마 낫지만, 광고 슬롯이 많아지면 각 슬롯에서 **HTTP 요청**이 발생하여 **웹 바이탈 지표**(LCP 등)에 악영향을 줄 수 있습니다. 이를 개선하려면 **레이지 로딩(Lazy Loading)** 기법이 유효합니다 . 사용자가 아직 보지 않는 아래쪽 광고는 스크롤에 근접할 때까지 로딩을 지연시켜, 초기 페이지 렌더링을 빠르게 합니다. AdSense 도 **레이지 로드 옵션**을 지원하는데, `<ins`

class="adsbygoogle" data-ad-format="auto" data-full-width-responsive="true" data-matched-content-rows-num="2" ...> 등에 속성을 추가하거나, Intersection Observer 로 직접 구현할 수 있습니다.

또한, Next.js 의 코드 분할 기능을 이용해 **광고 컴포넌트를 필요한 페이지에서만 로드**하게 합니다. 예를 들어, 광고가 없는 페이지(예: help, settings 등)에서는 아예 AdSense 스크립트를 넣지 않아 불필요한 리소스 로딩을 막습니다.

**광고 차단기(AdBlock) 이슈:** 사용자 중 상당수가 AdBlock 이나 브라우저 내장 광고 차단을 사용합니다. 통계에 따르면 전 세계 **약 37%의 사용자**가 광고 차단기를 쓰며, 그 중 절반 가량은 “광고가 너무 성가셔서”라는 이유를 듭니다 (일부 통계). 광고가 차단되면 우리 수익도 줄어들지만, 이를 강제로 막을 방법은 제한적입니다. 한 가지 접근은 **온건한 광고 전략**으로 사용자가 굳이 차단하지 않아도 되게 만드는 것입니다. 과도한 팝업이나 전체화면 광고를 안 쓰고, **콘텐츠에 방해되지 않는 선에서** 광고를 넣으면, 몇몇 사용자는 차단을 해제하기도 합니다. 실제로 광고 차단 이용자의 50%는 **한두 사이트에서 차단을 예외 설정**해놓는다고 합니다 (믿을 만한 콘텐츠 제공 시).

우리도 서비스에 “**광고 수익으로 운영됩니다**” 안내를 띄워 공감대를 얻거나, 장기적으로 **광고 없는 유료 구독 모델**(프리미엄 회원) 등을 제시할 수 있습니다. 물론 초기에는 그런 것보다 **콘텐츠 품질**에 집중하여, 사용자가 광고쯤은 눈감아줄 만한 가치있는 사이트를 만드는 것이 우선입니다.

**광고 최적화 지속 개선:** 운영 중에는 AdSense 대시보드의 **RPM, CTR** 등을 체크하며 어떤 위치의 광고가 효과적인지 데이터 기반으로 분석할 것입니다. 효과가 낮은 광고는 제거하고, 사용자 선호에 따라 광고 카테고리를 차단하거나(AdSense 는 주제별 차단 가능), 반대로 수익 높은 카테고리를 노출 강화하는 등의 튜닝을 할 수 있습니다. 다만 특정 광고주/네트워크를 차단하면 경쟁 입찰이 줄어들어 전체 수익이 떨어질 수 있으니 , 너무 지나친 차단은 피하고 **Google 의 자동 최적화**를 신뢰하는 편이 좋습니다.

**배포, 호스팅 및 CI/CD**

## Vercel 기반 배포 전략 및 이점

현재 프론트엔드는 Vercel 에 연결되어 자동 배포되고 있습니다. Vercel 을 지속 활용하는 것은 여러모로 이점이 있습니다:

- **Zero Config 배포:** Next.js 프로젝트는 Vercel 이 자동으로 빌드/배포해주며, 별도 서버 세팅이 필요 없습니다. 브랜치 연결만 해두면 커밋 시 즉시 배포가 진행됩니다.
- **글로벌 CDN 및 엣지 네트워크:** Vercel 은 전 세계 엣지에 콘텐츠를 배포하므로, 한국 사용자뿐 아니라 다른 지역 사용자도 빠른 응답을 얻을 수 있습니다. 정적 리소스는 Cloudflare CDN 을 통해 자동 캐싱되므로, 별도 설정 없이 **성능 개선** 효과가 있습니다.
- **Serverless Functions 지원:** Next.js 의 API Routes 나 Vercel Functions 를 사용하면, Node.js 기반의 백엔드 코드를 각 지역 서버리스 함수로 배포할 수 있습니다. 이는 SSR 이나 API 엔드포인트 구현에 활용되고 있으며, 게시판의 **게시글/댓글 CRUD API**, **좋아요 처리 API**, **AI 생성 API** 등을 구현하는 데 쓰입니다. 필요에 따라 **Edge Functions** (더 가벼운 JS 함수)도 사용 가능하며, 쿠키 기반 A/B 테스트나 리디렉션 처리 등에 활용할 수 있습니다.
- **자동 HTTPS 와 도메인 관리:** Vercel 은 커스텀 도메인을 손쉽게 연결하고 SSL 인증서를 자동 적용해주므로, 별도의 인증서 구매나 설정이 필요 없습니다. 이는 운영상 번거로움을 크게 줄여줍니다.

## CI/CD 파이프라인 구성 (Git 연동 자동 배포)

**개발 워크플로우:** GitHub 저장소와 Vercel 을 연동하여 **\*\*지속적 통합/배포(CI/CD)\*\***를 구현합니다. 구체적으로, main 브랜치에 코드가 푸시될 때마다 Vercel 이 자동으로 빌드 및 프로덕션 배포를 수행합니다. 또한 feature/\* 등의 브랜치에 푸시 시 **Preview Deployment URL** 을 생성하여 해당 브랜치 버전을 팀원이 미리보기 할 수 있습니다. 이 미리보기 링크는 새로운 기능이나 수정 사항을 빠르게 공유하고 피드백받는 데 유용합니다.

**환경 변수 관리:** AdSense 클라이언트 ID, Supabase API 키, OpenAI Key 등 민감 정보는 Vercel 프로젝트 설정의 Environment Variables 에 저장해두고 Next.js 에서 process.env 로 참조합니다. 이렇게 하면 Git 저장소에 노출되지 않고도 빌드/런타임에 안전하게 값을 사용할 수 있습니다.

**품질 관리 자동화:** CI 단계에서 **\*\*린팅(Lint)\*\***과 **테스트**도 자동화할 예정입니다. 예컨대 GitHub Actions 를 설정해 PR 생성 시 **ESLint** 검사와 **Jest/Cypress** **테스트**를 돌리고, 통과 여부를 표시하도록 할 수 있습니다. Vercel 도 기본적으로 빌드 시 타입스크립트 오류나 빌드 에러가 있으면 배포하지 않으므로, 문제가 실서비스에 나가는 것을 어느 정도 막아줍니다.

**릴리즈 전략:** 초기에는 메인 브랜치에 곧바로 배포하지만, 안정화 후에는 dev(staging)와 prod(production) 브랜치를 나누는 것도 고려합니다. dev 브랜치는 비공개 테스트용 URL 로 배포되어 내부 검증을 거친 뒤, prod 로 머지되어 실제 사용자에게 제공되는 구조입니다. 다만 규모가 작을 때는 overhead 일 수 있어, 유연하게 결정할 부분입니다.

요약하면, **Vercel + Git** 을 통한 CI/CD 는 **속도와 안정성**을 모두 잡는 방향입니다. 개발자가 코드에 집중하고, 배포는 자동으로 이뤄지며, 롤백도 이전 배포로 손쉽게 돌아갈 수 있어 위험성이 낮습니다. 이는 스타트업이나 소규모 프로젝트에 최적화된 환경으로, ‘무서핑’의 빠른 업데이트 사이클에 부합합니다.

## 운영 및 안전성

### 콘텐츠 모더레이션 및 관리자 기능

운영단계에서 가장 중요하면서도 손이 많이 갈 수 있는 부분이 **\*\*커뮤니티 모더레이션(운영)\*\***입니다. 익명 게시판 특성상 **악성 이용자나 부적절 콘텐츠**가 나타날 수밖에 없고, 이를 잘 관리해야만 **건강한 커뮤니티**와 AdSense 정책 준수를 동시에 달성할 수 있습니다 .

**자동 필터 및 차단:** 앞서 논의한 **욕설 필터링 시스템**은 1 차 방어선입니다. 이로써 상당수의 욕설, 비속어, 광고성 도배 문구 등을 걸러낼 수 있습니다. 또한 **스팸 링크**(예: 도박 사이트 URL 등) 패턴도 목록화하여 필터링할 예정입니다. 이런 자동화 장치는 **실시간**으로 작동하여, 문제 콘텐츠가 애초에 노출되지 않게 막아주는 역할을 합니다.

**신고 기능:** 그러나 자동 필터만으로는 한계가 있으므로, **사용자 신고 시스템**을 도입합니다. 각 게시글이나 댓글 옆에 **“신고” 버튼**을 배치하여, 사용자들이 부적절한 내용을 발견하면 신고할 수 있게 합니다. 신고 사유도 **스팸, 욕설, 음란물, 혐오, 기타** 등으로 받아 분류합니다. 신고가 접수되면 우선 해당 콘텐츠를 **임시 블라인드(숨김)** 처리하고, 관리자가 검토 후 삭제/복구 여부를 결정하게 합니다.

**관리자 도구:** 운영자가 볼 수 있는 관리자 대시보드도 추후 개발할 계획입니다. 우선은 간단하게 **신고 접수 목록과 내용, 신고 횟수** 등을 보여주고 처리할 수 있는 UI 면 되며, 여력이 된다면 **사용자별 활동 로그, IP 별 히스토리, 특정 단어 필터 로그** 등도 볼 수 있게 하면 좋습니다. 초기에는 개발팀이 DB 를 직접 조회하거나 admin SQL 쿼리를 사용하는 형태로 최소한의 운영을 할 수 있겠지만, 서비스가 커지면 전담 관리 인력이 쉽게 사용할 수 있는 툴이 필요합니다.

**차별 및 방어:** 문제 이용자에 대해서는 **IP 차단**이나 **쿠키 차단** 조치를 내릴 수 있습니다. 일정 기간 (예: 7 일) 차단하거나 영구 금지할 수 있으며, 차단된 IP 에서 접속 시 경고 메시지를 띄울 것입니다. 다만 VPN 등으로 재접속이 가능하므로, 근본적으로는 **문제 상황을 빠르게 수습**하여 다른 이용자들이 피해를 덜 입게 하는 데 초점을 맞춥니다. 예컨대 불법 광고글이 올라오면 수 분 내 삭제하고 해당 패턴을 필터에 추가하는 식의 **반응 속도**가 중요합니다.

**운영 인력 고려:** 초반에는 운영을 개발자가 겸하지만, 사용자 규모가 늘면 **모더레이터**를 둘 필요도 있습니다. 커뮤니티의 신뢰성을 위해서는 운영 원칙을 투명하게 공개하고, 이용자와 쌍방향 소통 (공지사항, 피드백 받기)을 잘 하는 것이 좋습니다. **FAQ/가이드라인 페이지**를 만들어 “이런 콘텐츠는 금지, 위반 시 조치” 등을 명시하겠습니다.

요약하면, 모더레이션은 **기술과 인력의 결합**이 필요합니다. 기술적으로 최대한 자동화하되, 인간 운영자의 판단이 필요한 영역을 체계화하여, AdSense 정책에 저촉되지 않고 이용자들도 안심하고 활동할 수 있는 공간을 만들어가겠습니다 .



## 개인정보 보호 및 최소한의 트래킹

‘무서핑’은 로그인 없는 서비스인 만큼 수집하는 개인정보가 거의 없습니다. 하지만 **광고나 분석 툴**을 쓰면 일부 사용자 데이터가 처리가 될 수 있으므로, 이에 대한 고지와 보호 대책을 서술합니다.

**수집 항목:** 기본적으로 쿠키 식별자(익명 토큰), IP 주소(서버 로그), 브라우저 정보 정도가 운영상 수집됩니다. 여기에 Google Analytics 를 사용할 경우 **사용자 행동 익명 통계**가 추가되고, AdSense 는 **광고 쿠키**를 통해 사용자 관심사 기반 광고를 노출할 수 있습니다. 이러한 활동은 모두 **개인 식별이 불가능한 수준**에서 이뤄집니다. 다만 법적 요구에 따라, 이용자가 **쿠키 거부**를 원하면 거부할 수 있는 옵션을 제공해야 합니다.

**쿠키 동의 배너:** 한국 및 EU 등 규제를 준수하기 위해 **쿠키 동의 배너**를 초기 접속 시 표시하는 것을 고려합니다. “본 사이트는 원활한 서비스 제공을 위해 쿠키 등 기술을 활용합니다. 계속 이용하면 이에 동의하는 것으로 간주합니다.” 정도의 문구와 확인 버튼을 제공하고, **개인정보처리방침** 링크를 함께 둡니다.

**개인정보처리방침:** 서비스 출시 전에 간략하게나마 사이트에 **Privacy Policy** 페이지를 게시할 계획입니다. 여기엔 **수집되는 데이터, 이용 목적, 보관 기간, 제 3 자 제공(광고/분석), 문의처** 등을 명시합니다. 특히 AdSense 의 개인정보 관련 정책에 따라, Google 광고 쿠키 사용에 대한 언급과 사용자 선택권(opt-out) 안내를 포함합니다.

**최소한의 트래킹:** 사용자 경험을 해치지 않는 범위에서만 분석 도구를 사용할 것입니다. 예를 들어 **Page View, 세션 길이, 인기 게시글 순위** 정도를 파악하기 위해 Google Analytics v4 를 넣을 수 있지만, **개별 사용자의 이동 경로**까지 상세 추적하거나, **과도한 A/B 테스트**를 하여 쿠키를 많이 심지는 않을 것입니다. 또한 절대로 이메일이나 전화번호 등 PII 는 수집하지 않으므로, 해킹 당해도 유출될 민감정보는 없게 설계합니다.

**데이터 보안:** 백엔드에 저장되는 최소한의 데이터(게시글, 댓글 등)가 유출되지 않도록 **DB 보안 규칙**을 설정하고, 필요한 경우 **암호화**도 검토합니다. 예를 들어

IP 주소는 그대로 저장하지 않고 해시처리하여 저장하면 운영상 식별은 되면서도 개인정보 노출 위험은 줄일 수 있습니다.

이러한 조치들을 통해 **이용자의 프라이버시를 존중**하면서 서비스 운영을 할 것이며, 이는 커뮤니티의 신뢰 기반을 다지는 일이라고 믿습니다.

## 기술 스택 제안 요약

마지막으로, 앞서 언급한 구현을 뒷받침할 **주요 기술 스택**을 항목별로 정리합니다:

- **프론트엔드:** Next.js (React 18) + TypeScript
  - Routing, SSR/CSR, API Routes 등에 사용.
  - SWR or React Query for data fetching & caching.
- **UI/스타일:** TailwindCSS + Headless UI 또는 Chakra UI
  - 빠른 스타일링과 반응형 디자인 구현.
  - 다크 모드 및 테마 커스터마이징 지원.
- **에디터/폼:** 이미지 업로드에 React Dropzone, 미리보기 FileReader API 사용.
  - 게시글 작성 폼은 간단한 텍스트 + 이미지 첨부 형태로 커스터마이징.
- **백엔드:** Vercel Serverless Functions (Node.js 18)
  - 게시판 CRUD API, 좋아요 API, AI 호출 API 등을 구현.
  - 필요시 Express.js 마이크로서비스를 추가 배포 (Railway, Heroku 등 고려).
- **데이터베이스:** Supabase(PostgreSQL) + Prisma ORM
  - 게시글, 댓글, 좋아요 이벤트 저장 등 관계형 데이터 관리.
  - Supabase Auth 는 당장은 미사용, 향후 소셜 로그인에 대비.
  - Vercel KV (Upstash Redis)
    - 실시간 카운터, 레이트 리미팅 키 등 휘발 데이터 저장.
- **파일 스토리지:** Supabase Storage (S3 호환) 초기 도입
  - 이미지 파일 업로드/호스팅에 사용.
  - Vercel Blob Storage 및 Cloudflare CDN 은 추후 트래픽 증가 시 병행.
- **AI API:** OpenAI API (또는 대안: Replicate Stable Diffusion)
  - 이미지 생성 등 콘텐츠 생성에 사용.
  - 백엔드에서 비동기 호출, 응답을 클라이언트에 전달.
- **광고:** Google AdSense

- 코드 스니펫 `<script async src="adsbygoogle.js">`를 `_document.js`에 추가.
  - 요소들을 적절 위치에 배치.
  - 광고 스타일 튜닝을 위해 `ads.txt` 설정 및 AdSense 제어패널 활용.
- **CI/CD & 호스팅:** Vercel + GitHub
  - git push 시 자동 배포 (Preview/Production).
  - GitHub Actions 로 테스트/린트 자동화 (필요 시).
  - 모니터링은 Vercel Analytics 혹은 자체 로그 수집 (초기에는 Sentry 고려).
- **모니터링/로깅:** Sentry (에러 트래킹), Google Analytics (유저 트래픽 분석)
  - 프론트 및 백엔드 에러를 Sentry 로 실시간 감시.
  - GA4 로 페이지뷰/사용자 행동 분석 (개인정보 비수집 설정).

이상의 스택은 **현대 웹 서비스의 표준적 구성**으로, 개발 생산성과 초기 실행 속도에 중점을 둔 선택입니다. 추후 트래픽 증가와 서비스 확장에 따라, 각 요소는 교체/보완이 가능하도록 **유연한 구조**로 설계할 것입니다.

## 결론 및 권고사항

본 문서를 통해 ‘무서핑’ 서비스에 **공개 게시판**과 **광고 수익화** 기능을 추가하는 기획을 상세히 검토했습니다. 기술적인 구현 가능성은 전반적으로 **매우 높게** 평가됩니다. React/Next.js + 서버리스 백엔드 조합은 초기 개발에 적합하고, AI API 활용도 현 시점에서 현실성이 있습니다. 그러나 성공적인 서비스 운영을 위해서는 아래와 같은 **핵심 쟁거야 할 사항**들이 도출되었습니다:

- **1. 익명 서비스의 보안 강화:** 익명성은 양날의 검입니다. 사용자 진입을 쉽게 하지만, 악용을 막기 어렵습니다. 따라서 **중복 방지 토큰** 정도의 간단한 장치로는 부족하며, **고급 레이트 리미팅**과 **CAPTCHA 도입** 등으로 악성 행동을 걸러야 합니다. 또한 게시글 수정/삭제를 위해 **비밀번호 인증** 같은 보완책을 넣어, **본인 게시물 관리** 신뢰성을 높이길 권장합니다.
- **2. 성능 및 확장성 확보:** 좋아요 수나 조회수 등의 **높은 빈도 업데이트** 기능은 처음부터 **확장성 고려**하여 설계해야 합니다. 가능하면 **비동기 이벤트 처리**(이벤트 큐 + 배치 업데이트)로 전환해 **DB 락 경쟁**을 줄이고, **인덱싱**과 **캐싱**으로 **읽기 성능**도 최적화해야 합니다. 또한 서버리스 사용 시 피할 수 없는 **콜드 스타트**를 인지하고, 사용자 피드백을 통해 문제시 보완(예: 첫 요청 지연시 로딩 메시지 더 잘 보이게)하거나, 일정 트래픽 이상일 땐 **상시 서버 프로세스**로 전환을 검토합니다.

- **3. 콘텐츠 모더레이션 & 정책 준수:** 커뮤니티가 활성화될수록 콘텐츠 관리가 중요합니다. 욕설/악성 콘텐츠 필터를 지속 개선하고, 신고/운영자 대응 체계를 마련해 유해 콘텐츠 빠른 제거를 실천해야 합니다. 이것은 사용자 경험을 위해서도, AdSense 정책 준수를 위해서도 필수입니다. 특히 광고가 게재되는 페이지의 UGC는 모두 검열 대상이라는 인식을 갖고, 문제 발생 시 Google 측과 협력하여 해결해야 합니다.
- **4. 비용 모니터링 및 비즈니스 모델:** AI API 호출, 서버리스 DB, CDN 사용 등은 모두 **사용량 기반 비용**이므로, 유저 증가에 따라 비용이 선형 또는 지수적으로 증가할 수 있습니다. 그러므로 **\*\*초기 매출(광고)\*\***이 안정될 때까지 비용을 면밀히 관리하고, **임계치 경보**를 설정하며, 필요 시 **기술적 비용 절감**(예: 캐싱 향상, 이미지 최적화)을 해야 합니다. 또한, 장기적으로 **수익원을 다양화**(예: 프리미엄 구독, 제휴 콘텐츠 등)하여 광고에만 의존하지 않도록 기획을 병행하는 것을 권장합니다.
- **5. 사용자 피드백과 지속 개선:** 끝으로, 서비스는 출시가 끝이 아니라 시작입니다. 사용자들의 피드백 (UI 불편, 기능 제안, 버그 제보 등)을 적극 수렴하는 채널을 열고, 빠른 개선 사이클로 반영해야 합니다. 예를 들어, 만약 “익명이라 너무 혼란스럽다, 닉네임이라도 있으면 좋겠다”는 의견이 많으면 **게스트 닉네임 기능**을 추가 검토할 수 있습니다. 기술적으로 크게 어렵지 않으면서 만족도를 올릴 수 있는 부분을 찾아나가야 합니다.

요약하면, ‘무서핑’ 서비스의 새로운 기능들은 충분히 **실현 가능**하며, 제대로 운영된다면 **유익미한 사용자 참여 증가와 수익 창출**을 기대할 수 있습니다. 다만 위에서 강조한 **보안, 모더레이션, 성능, 비용** 측면의 이슈들을 간과하지 말고 선제적으로 대응하는 것이, **작지만 끈질긴 커뮤니티 서비스**로 성장하는 비결입니다. 이 PRD에서 도출된 방안을 토대로 개발을 진행하고, 실제 사용자 반응과 데이터를 보면서 더욱 정교하게 서비스 품질을 향상시켜 나가길 바랍니다.