

# AI 기반 공포/유머 콘텐츠 서비스

## ‘무서핑’

### 기능 확장 기획서 (PRD)

#### 1. 배경 및 목표

‘무서핑’은 AI로 생성된 귀신 캐릭터와 이야기를 감상하는 웹 서비스로, 현재 React/Next.js 기반 **프론트엔드**만 운영되고 있습니다. 사용자는 회원가입 없이 익명으로 콘텐츠를 즐길 수 있으며, 지금까지는 운영자가 제공한 AI 콘텐츠를 보기만 하는 형태입니다. 다음 단계로, **사용자 참여형 공개 게시판**과 **Google AdSense 광고 수익화** 기능을 도입하여 서비스의 **활성도**를 높이고 **지속 가능한 수익 모델**을 구축하고자 합니다.

이번 기획서(PRD)는 이전 버전(PRD v0.3)에서 제시된 기능 아이디어들을 바탕으로, 기술 검토 결과를 반영하여 **세부 요구사항**과 **구현 방안**을 상세히 정리한 것입니다. 특히 백엔드 도입, 실시간 데이터 관리, AI 이미지 생성 비용, 콘텐츠 중재, 성능 최적화, 광고 정책 준수 등의 측면을 고려하여 기능을 설계합니다. 목표는 새로운 기능 추가로 인한 **사용자 경험 향상**과 **서비스 확장성** 확보이며, 잠재적인 위험 요소에 대한 예방책도 함께 제시합니다.

#### 2. 주요 기능 및 요구사항

##### 2-1. 공개 게시판 및 피드 기능

###### a. 메인 페이지 상단 - 최신 & 월간 베스트 섹션

- **최근 출몰한 존재들:** 최신 게시물 4~6 개를 카드 형태로 표시합니다. 각 카드에는 썸네일 이미지, 제목, 좋아요 수(하트 대신 공포 테마 아이콘) 등이 포함되며, 클릭 시 해당 게시물 상세 페이지로 이동합니다. 최신 게시물 선정은 등록일시 내림차순으로 하며, 새로운 게시물이 업로드되면 실시간으로 목록이 갱신됩니다.
- **이달의 악명 높은 존재들:** 최근 30 일 내 인기 게시물 4~6 개를 표시합니다. 좋아요 수와 게시일을 종합 고려한 가중치 알고리즘으로 선정하며, 기본적으로 최근 한 달간 좋아요 수 상위 게시물을 보여주되 최신 게시물일수록 약간의 가중치를 주어 너무 오래된 게시물이 상단을 차지하지 않도록 합니다. (예: 정렬 점수 = 좋아요 수 +  $\alpha$ \*(최근 게시물 가중치) 방식으로 조정).
- **UI/디자인:** 두 섹션을 메인 화면 상단에 배치하되, 데스크톱에서는 2 행 그리드(한 행당 3~4 개 카드)로, 모바일에서는 가로 스크롤 리스트로 표시해 반응형을 지원합니다. 카드 디자인은 공포 분위기에 맞게 어둡게 하고, 좋아요 숫자와 아이콘은 붉은색 등으로 강조합니다.
- **데이터 제공 방식:** Next.js 서버 측 렌더링으로 페이지 요청 시 최신/인기 게시물을 미리 불러옵니다. 게시물 데이터는 백엔드 DB 에서 쿼리하며, 최신은 created\_at 필드로 정렬, 월간 베스트는 최근 30 일 범위에서 like\_count 로 정렬합니다. 원활한 쿼리를 위해 해당 컬럼들에 인덱스를 적용합니다.

## b. 게시판 화면 - 게시물 목록 및 정렬


- **게시물 목록:** 공개 게시판은 모든 사용자가 게시물을 올리고 볼 수 있는 익명 게시판입니다. 기본 피드 화면에서 최신 게시물이 최신순으로 나열되며, 각 목록 아이템에는 썸네일(작ge), 제목, 간략한 내용(요약 1~2 줄), 작성 시간, 좋아요 수가 표시됩니다.
- **정렬 옵션:** 사용자가 최신순 / 인기순으로 정렬을 전환할 수 있는 토글이나 버튼을 제공합니다. 인기순은 일정 기간 내 좋아요 수가 많은 순으로 정렬하며, 기본적으로 최근 24 시간 또는 7 일 등 기간 제한 인기글 위주로 보여줄지 추후 데이터에 따라 결정합니다 (초기에는 단순 전체 인기순으로 구현). 인기순 정렬 시에도 너무 오래된 게시물이 상단에 오래 머무르지 않도록 기간 제한이나 가중치를 적용합니다.
- **무한 스크롤/페이지네이션:** 사용자가 피드를 계속 내려볼 수 있도록 무한 스크롤을 적용합니다. 최초 10 개 정도 로드 후 스크롤 시 추가 로드하며, 로딩 중임을 나타내는 스피너 UI 를 표시합니다. (만약 구현 복잡도나 성능 이슈가 있다면 버튼식 페이지네이션으로 대체 가능).
- **게시물 상세:** 게시물 카드를 클릭하면 상세 페이지에서 큰 이미지와 제목, 설명(작성자가 입력한 스토리/설명), 좋아요 버튼 등을 보여줍니다. 댓글 기능은 제거되므로 댓글 창은 없으며, 대신 관련 콘텐츠 추천이나 작성자의 다른 게시물은 익명이므로 표시하지 않습니다.

- **반응형 디자인:** 모바일에서는 목록이 한 열로 나오며, 썸네일도 가로폭에 맞게 줄여 표시합니다. 데스크톱에서는 2 열 배열로 한 화면에 더 많은 게시물이 보이도록 할 수 있습니다. Hover 효과나 마우스 대응은 데스크톱에만 적용하고, 모바일에서는 터치 피드백(:active 스타일 등)으로 대체합니다.

## 2-2. 크롬 공룡 스타일 미니게임

- **기능 개요:** 사용자 피드 화면에서 즐길 수 있는 러닝 액션 미니게임을 추가합니다. 이는 Chrome Dino 게임과 유사하게, 캐릭터(예: 귀신)가 장애물을 피해 점프하며 달리는 게임입니다. 점수가 올라갈수록 캐릭터가 **성장 또는 변신**하여 더욱 무서운 모습이 되는 **이스터에그 요소**를 포함합니다. 게임을 통해 사용자는 대기 시간에 재미를 느낄 수 있습니다 (예: 게시물 로딩 중이거나 새로운 콘텐츠가 없을 때).
- **배치 및 호출:** 데스크톱에서는 피드 오른쪽 사이드바(예: 폭 300px 영역)에 고정 배치합니다. 스크롤을 내려도 사이드바에서 계속 보이도록 하여 언제든지 플레이할 수 있습니다. 모바일에서는 화면 공간 제한으로 바로 보여주지 않고, 화면 한 칸에 **작은 유형 아이콘 버튼**을 뒤 터치 시 전체 화면 오버레이로 게임이 실행되도록 합니다.
- **게임 플레이:** 점프/슬라이드 등의 조작은 키보드(데스크톱: 스페이스바/화살표, 모바일: 화면 탭/스와이프)로 입력받습니다. 게임 난이도는 서서히 증가하며, 배경은 어두운 숲이나 폐허 등 공포 분위기로 디자인합니다. **점수 표시**와 **최고 기록**(세션 한정)을 보여주지만, 해당 기록은 *앱 내에 저장하거나 서버에 기록하지는 않습니다* (시간 때우기 용이며 랭킹 경쟁을 도입하지 않음).
- **기술 구현:** HTML5 Canvas 또는 WebGL 을 사용한 게임으로, React 컴포넌트 내 <canvas> 엘리먼트를 활용합니다 . 성능을 위해 **requestAnimationFrame** 을 사용하여 애니메이션을 구현하고, React 상태 업데이트는 최소화합니다. Phaser 와 같은 경량 게임 프레임워크를 도입하는 것도 고려합니다. 이 컴포넌트는 **dynamic import** 로 지연 로드하여 초기 피드 렌더링에 영향을 주지 않도록 합니다. 사용자가 게임을 시작할 때 리소스를 불러오는 방식으로 최적화합니다.
- **기타 고려사항:** 게임 도중에도 게시판 콘텐츠를 볼 수 있도록 데스크톱 사이드바에서는 소리 없이 진행하고, 모바일 오버레이에서는 일시정지 기능을 제공합니다. 과도한 CPU 사용으로 웹앱이 느려지지 않도록 **캔버스 최적화**와 **메모리 관리**를 신경 씁니다. (예: 게임이 보이지 않을 때 requestAnimationFrame 정지).



## 2-3. 존재 생성 및 강화(Evolution) 기능

- **기존 존재 생성:** 현재 서비스에서는 사용자가 간단한 프롬프트(이름, 특징 등)를 입력하면 AI가 **귀신 캐릭터 이미지**와 소개 글을 생성해주는 기능이 있습니다. 이 생성된 **게시물**을 서비스에 등록하여 공유할 수 있게 합니다. (MVP에서는 프론트엔드에서만 처리했으나, 기능 확장 후에는 생성 결과를 DB에 저장하여 게시판과 연계).
- **새로운 ‘강화’ 기능:** 게시물 등록 후 해당 **존재를 한 단계 강화**할 수 있는 옵션을 제공합니다. 사용자가 자신의 게시물(생성한 귀신)에 들어가 보면 “ 강화하기” 버튼이 활성화되고, 이를 누르면 **AI 이미지 2차 생성**이 실행됩니다. 이때 입력 프롬프트나 설정은 초기 생성시 값들을 활용하되, AI 모델 측 파라미터를 달리하거나 프롬프트에 “더 강력한 형태의” 등의 문구를 추가해 **진화된 모습**을 생성하도록 합니다.
- **이미지 비교 표시:** 강화 기능 수행 전후로 게시물에는 **2개의 이미지**(초기 생성 모습 & 강화된 모습)가 함께 포함됩니다. UI 상으로는 before/after 슬라이더를 사용해 두 이미지를 겹쳐놓고 비교할 수 있게 하거나, 두 이미지를 나란히 배치합니다. 사용자들은 강화 전후의 변화를 직관적으로 볼 수 있으며, 강화 버튼 클릭 전에는 슬롯이 빈 상태로 “강화하기를 눌러보세요” 등의 안내를 표시합니다.
- **제한 사항:** 강화 기능은 **게시물당 1회만** 사용할 수 있습니다. 한 번 강화된 존재는 추가 강화를 할 수 없으며, 강화 버튼은 한 번 사용 후 비활성화됩니다. 이를 통해 **AI 사용 횟수 남용**을 방지하고, 콘텐츠의 희소가치를 높입니다.
- **기술 구현:** AI 이미지 생성에는 **Stable Diffusion** 계열 모델(API 활용)을 사용합니다. 백엔드 서버리스 함수 또는 전용 API 서버에서 **Replicate** 나 **Hugging Face Inference API** 호출로 이미지를 생성합니다. 평균 생성 시간은 수 초 정도이며, 응답이 올 때까지 프론트엔드에서 로딩 스피너와 “강화 중... 영혼을 불러오는 중입니다!” 같은 문구로 대기 상태를 표시합니다. 생성된 이미지는 일단 외부 저장소에 업로드된 후 게시물 데이터에 URL로 추가 저장됩니다.
- **비용 및 성능:** 외부 AI API 사용에는 건당 비용이 발생합니다. 예를 들어 Replicate의 Stable Diffusion 모델은 **이미지 1장당 \$0.005-\$0.02(약 12센트)** 수준의 비용이 듭니다. 많은 사용자가 빈번히 사용할 경우 비용 누적이 우려되므로 **사용자당 하루 X회로 제한**하거나, 동일한 요청에 대해서는 **캐싱**하여 재사용하는 전략을 고려합니다. 또한 Replicate 계정 설정에서 월별 사용 한도를 설정해 두어 예산을 초과하지 않도록 합니다. 성능 면에서는 API 호출 지연이 5~10초 이상 길어지지 않도록 모델을 경량화하거나 초고속 API 옵션을 사용할 수 있습니다.

## 2-4. 커뮤니티 및 UX 개선 사항

- **댓글 기능 제거:** PRD v0.3에서는 댓글/답글 기능 추가를 고려했으나, **익명 서비스 특성상 부작용**(분쟁, 악플 등) 우려와 초기 이용자 규모를 감안하여 댓글 대신 **좋아요 중심의 반응** 체계로 단순화합니다. 게시물 상세

페이지에는 댓글 입력 UI를 아예 제공하지 않고, 사용자 반응은 좋아요 수로만 표시합니다. (추후 사용자 수 증가 및 운영 인력 확보 시 댓글 기능을 재논의할 수 있음.)

- **좋아요 버튼 테마화:** 기존 하트 아이콘 좋아요를 공포 콘셉트에 맞춰 **피문은 손** 이나 **해골**  아이콘 등으로 교체합니다. 단, 직관적으로 “좋아요” 의미를 전달해야 하므로 아이콘 옆에 작은 텍스트(예: “좋아요” 대신 “소름...” 같은 컨셉 텍스트)로 표시하여 사용자들이 누르면 무슨 기능인지 이해하도록 합니다.
- **좋아요 기능 사양:** 로그인 없이도 중복 클릭을 방지하기 위해, 클라이언트에서 **쿠키나 로컬스토리지에 사용자 식별 토큰**을 저장하여 각 게시물당 한 번만 좋아요를 누를 수 있게 합니다. 한 번 누른 좋아요는 다시 눌러도 증가하지 않으며, 취소 기능은 익명 환경상 복잡성을 늘릴 수 있으므로 **좋아요 취소는 불가** 처리합니다 (누르면 토글되지 않고 고정). 좋아요 수는 누르는 즉시 프론트엔드에 증가 표시하고, 백엔드에도 반영합니다.
- **게시글 작성 및 삭제:** 사용자는 AI로 생성된 결과를 게시판에 등록할 때 **별도의 계정 없이** 바로 올릴 수 있습니다. 이때 작성자 이름은 모두 “익명”으로 표시되고, 개별 사용자를 구분하지 않습니다. 대신 작성자 본인이 글을 올린 후 **5분 이내 수정/삭제** 옵션을 제공합니다. 수정은 제목/내용만 가능하고 이미지 자체는 처음 생성한 것으로 고정됩니다. 삭제는 확인 다이얼로그를 거쳐 수행됩니다. 작성자를 인증하는 방법으로 **브라우저에 임시 토큰**을 발급하여 해당 토큰을 가진 경우에만 수정/삭제 요청을 허용합니다. (구현: 게시물 생성 시 UUID 토큰을 발행해 응답으로 전달하고, 사용자 브라우저 로컬스토리지에 저장. 수정/삭제 요청 시 이 토큰을 헤더에 포함해 백엔드 검증. 토큰 분실 시 대비책은 없으나 익명 서비스의 제한으로 용인함.) 추후 필요시 게시물마다 사용자가 설정하는 삭제 비밀번호를 도입하는 방법도 고려합니다.
- **UI/UX 전반 개선:** 다크 테마 기반 디자인을 유지하되 **가독성**을 확보합니다. 배경과 글자 명암비를 충분히 주고, 주요 아이콘이나 버튼에는 의미에 맞는 레이블을 추가합니다. 예를 들어, 좋아요 버튼에 마우스를 올리면 “이 존재 소름 돋네요! (좋아요)” 등의 툴팁을 보여줍니다. 또한 **접근성**을 높이기 위해 이미지에는 대체 텍스트를 넣고, 키보드만으로도 조작 가능한지 점검합니다. 반응형 측면에서는 작은 화면에서 글씨 크기, 터치 영역을 충분히 크게 조정하여 사용자 불편을 줄입니다.

## 2-5. Google AdSense 광고 통합

- **광고 도입 배경:** 서비스 트래픽 증가에 대비해 **수익 모델**로서 Google AdSense를 적용합니다. 초기에는 큰 수익을 기대하기 어렵지만, 향후 활성 사용자 수가 늘면 서버 비용을 상쇄하고 수익을 창출할 수 있습니다.

- **광고 배치 전략:** 사용자 경험을 해치지 않는 선에서 광고를 배치합니다. 우선 고려하는 위치는 **상단 배너**(메인 페이지 상단 또는 내비게이션 바로 아래), **사이드바 배너**(데스크톱 게시판 사이드), **피드 목록 중간**(예: 목록 글 5 개당 1 개 정도의 빈도로 네이티브 스타일 광고), **게시물 상세 하단** 등입니다. 한 화면에 광고가 과다 노출되지 않도록 조절하며, **반응형 광고 단위**를 사용해 화면 크기에 따라 적절한 크기로 표출되게 합니다.
- **구현 방식:** AdSense 승인을 받은 후 제공되는 광고 스크립트를 Next.js의 <Head> 영역이나 적절한 컴포넌트에 삽입합니다. React 용으로는 react-adsense 같은 패키지를 쓸 수도 있지만, 스크립트 직접 삽입으로 간단히 구현 가능합니다. 자동 광고(Auto ads)를 활성화하여 Google 이 최적의 위치에 광고를 넣도록 테스트할 수도 있으나, 본 서비스는 특수 테마이므로 **수동 제어**를 우선시합니다. 각 광고 위치에는 <ins class="adsbygoogle"> 요소를 넣고 해당 영역에 맞는 광고 크기(Responsive)로 설정합니다.
- **정책 준수:** 공포 컨셉트 콘텐츠는 자칫 Google 광고 정책의 **유해하거나 충격적인 콘텐츠(shocking content)** 규정에 저촉될 수 있습니다 . 이에 따라 **극단적으로 잔인한 이미지(유혈, 시신 훼손 등)**나 **선정적인 내용**은 게시판 운영 정책상 금지하고, AI 생성 시에도 가이드라인을 준수합니다. 예를 들어 지나친 유혈 표현의 이미지는 검출 시 게시가 제한될 수 있고, 사용자가 그런 콘텐츠를 올릴 경우 **관리자가 즉시 삭제**합니다. Google 은 피가 튀거나 심한 공포 연출 이미지를 **충격적인 콘텐츠**로 간주하여 광고 게재를 제한할 수 있으므로 , 이를 예방하기 위해 **콘텐츠 모니터링 시스템**을 운영합니다. 또한, AdSense 측 요구에 따라 **개인정보 처리방침/이용약관 페이지**를 사이트에 게시하고, **쿠키 동의 배너**를 통해 사용자에게 광고 쿠키 사용에 대한 동의를 얻도록 합니다.
- **광고 성과 측정 및 최적화:** AdSense 대시보드를 통해 광고 노출수, 클릭률을 모니터링하고, 배치 위치별 성과를 분석합니다. 만약 특정 위치의 광고가 사용자 경험을 저해한다면(예: 이탈률 증가), 해당 위치를 조정하거나 제거합니다. 초기에는 트래픽이 많지 않으므로 **과도한 광고 배치는 지양**하고, 서비스 콘텐츠가 충실해지면서 단계적으로 늘리는 것을 권장합니다. 또한, **광고 로딩 속도**도 신경 써서, 페이지 LCP 지표가 떨어지지 않도록 **\*\*지연 로드(Lazy load)\*\***를 적용합니다 . 사용자 중 광고 차단기 사용 비율도 고려하여, 크게 방해되지 않는 광고 디자인을 유지하고, 장기적으로는 프리미엄 모델(광고 제거 유료 구독 등)도 옵션으로 검토할 수 있습니다.

### 3. 기술 구현 계획 및 고려사항

#### 3-1. 백엔드 도입 및 데이터 관리

- 서버리스 백엔드 선택:** 현재까지는 별도 백엔드 없이 동작해 왔으나, 게시판 기능을 위해 **데이터베이스 및 실시간 기능**이 필요합니다. **\*\*Supabase(PostgreSQL 기반)\*\***를 우선 검토하며, 대안으로 **Firebase(Firestore)**도 고려합니다. Supabase의 장점은 SQL 기반 유연한 쿼리와 **실시간 구독**, 그리고 Vercel과 궁합이 좋다는 점입니다. 무료 플랜으로 시작할 경우 **무제한 API 호출, 월 50,000 MAU(Monthly Active Users), 500MB DB** 등이 제공되어 초기 규모엔 충분합니다.
- 데이터 모델:** 게시판 구현을 위해 주요 테이블은 **Posts, Likes** 두 개로 설계합니다.
  - Posts:** 게시물 본문 정보를 저장. 필드 예시 - id (고유키), title, description(내용/스토리), image\_url (원본 이미지), image\_url\_evolved (강화 후 이미지, 없으면 NULL), created\_at, like\_count, author\_token (삭제용 토큰) 등.
  - Likes:** 좋아요 이벤트 저장. 필드 - post\_id, like\_id(고유키), user\_token(누른 사용자 식별, 쿠키 기반), created\_at. → 좋아요 중복 방지 및 통계 분석에 활용. 단, 초기에는 like\_count를 Posts 테이블에 직접 두고, 사용자의 중복 여부는 서버에서 Likes 테이블로 체크하는 방식으로 간소화 가능.
- API 설계:** Next.js의 API Route 또는 Vercel Serverless Function을 이용해 RESTful API 엔드포인트를 구성합니다. 예를 들어 GET /api/posts?sort=latest (최신 게시물 목록), POST /api/posts (새 게시물 생성), POST /api/like (좋아요 클릭) 등을 만들어 프론트엔드에서 활용합니다. Supabase를 사용하면 자체 REST API와 클라이언트 라이브러리도 있지만, **\*\*비즈니스 로직(토큰 검증 등)\*\***을 위해 자체 API 레이어를 거치는 것을 권장합니다.
- 데이터 일관성과 성능:** 다수 사용자가 동시에 좋아요를 누를 때 **동시성** 문제가 발생할 수 있습니다. 이를 해결하기 위해 DB 트랜잭션이나 잠금보다는, **궁극적 일관성(Eventual Consistency)** 모델을 고려합니다. 예를 들어, 좋아요 API 요청 시 즉각 Posts.like\_count를 +1 업데이트하기보다 Likes 테이블에 기록만 하고, 별도의 배치나 트리거로 일정 주기마다 집계하여 Posts.like\_count를 갱신하는 방안을 생각해볼 수 있습니다. 이는 대규모 트래픽에서도 잠금 경합을 줄여주는 방법입니다. 다만 초기 사용자 규모에서는 바로 like\_count를 증가시키는 단순 방식도 큰 무리는 없으므로 우선 구현하고, 추후 트래픽 증가 시 전환을 검토합니다.
- 확장 및 비용 관리:** Supabase 무료 플랜의 제약(동시 연결 수, DB 크기 등)을 수시로 모니터링합니다. 유저가 늘어나면 Pro 플랜으로 업그레이드해야 하며, 특히 **DB 저장 용량과 파일 스토리지 사용량** 증가에 대비합니다. (예: 이미지 파일은 1GB 무료 용량 초과 시 GB 당 추가 비용 발생). 또한 Vercel Serverless는 **콜드 스타트**로 인해 API 첫 호출 지연이 있을 수 있으므로, 트래픽이 들쭉날쭉한 경우 주기적인 워크로드(예: 분당 한 번 dummy 호출)로 함수를 깨워두는 방법도 있습니다.

### 3-2. AI 이미지 생성 통합

- **모델 및 서비스:** Stable Diffusion 기반의 이미지 생성 모델을 사용하되, 직접 호스팅 대신 **API 서비스**를 이용합니다. **Replicate** 를 사용할 경우 REST API 로 원하는 모델을 호출할 수 있고, **Hugging Face** 의 Inference Endpoint 를 쓸 수도 있습니다. 두 서비스 모두 요청당 과금이며, Replicate 는 시간 단위과금(예: GPU 초당 \$0.000X) 또는 모델별 고정과금 형태입니다. 일반적으로 **1 장의 이미지를 생성하는 데 약 \$0.01 내외의 비용**이 예상됩니다 .
- **백엔드 연동:** 프론트엔드에서 직접 API 키를 노출하지 않도록, Next.js API Route (서버측)에서 Replicate 나 HuggingFace 에 요청을 대리 수행합니다. API 키는 Vercel 환경변수에 저장하고, 서버 함수 내에서 fetch 호출 시 헤더에 첨부합니다. 이렇게 하면 브라우저에 노출되지 않고도 API 를 사용할 수 있습니다.
- **생성 속도 최적화:** AI 이미지 생성에는 5~10 초 정도 시간이 소요될 수 있습니다. 사용자가 느리다고 느끼지 않도록 **UX 처리**에 신경 씁니다. 예를 들어, 생성 진행 중에는 로딩 애니메이션과 함께 “AI 가 그림을 그리는 중입니다...” 등의 메시지를 띄워두고, **진행률을 보여주지는 않더라도** 요청이 살아있다는 피드백을 줍니다. 만약 API 응답이 실패하거나 15 초 이상 지연되면 자동으로 요청을 취소하고 에러 메시지 (“생성에 실패했습니다. 다시 시도해주세요.”)를 표시합니다.
- **캐싱 전략:** 동일하거나 매우 유사한 요청에 대한 반복 생성은 비용 낭비일 수 있으므로, 최근 생성된 결과는 일정 시간 **캐싱**합니다. 캐싱은 사용자가 직접 입력한 프롬프트가 다양하기 때문에 히트율은 낮겠지만, 혹시 모를 악용(동일 입력 반복) 대비책입니다. 캐싱은 서버 메모리 또는 Upstash Redis(Vercel KV)를 사용할 수 있습니다.
- **NSFW 필터 및 콘텐츠 안전:** AI 가 생성하는 이미지에 예측 불가능성이 있으므로, **안전 필터**를 적용합니다. Stable Diffusion 모델 자체에 NSFW 필터 옵션을 활성화하고, 추가로 결과 이미지에 대해 **Cloudinary** 와 같은 서비스의 콘텐츠 모더레이션 API 를 활용해 부적절한 이미지(과도한 폭력, 선정성 등)를 검사합니다. 모더레이션에 걸릴 경우 해당 이미지를 폐기하고 사용자에게 재생성 요청이나 실패 안내를 합니다. 이중 장치를 통해 **광고 정책 위반** 가능성을 줄입니다.

### 3-3. 콘텐츠 모더레이션 및 커뮤니티 관리

- **익명성으로 인한 이슈:** 익명 게시판은 자유로운 참여를 장려하지만 동시에 **악성 콘텐츠**가 올라올 위험이 높습니다. 이를 방지하면 커뮤니티 품질이 떨어지고, 궁극적으로 **Google AdSense 정책 위반**으로 광고 게재 제한



또는 계정 정지 위험이 있습니다 . 따라서 **사전 예방 + 사후 대응** 체계를 모두 마련합니다.

- **금지 콘텐츠 가이드라인**: 커뮤니티 이용 수칙을 정리하여, 노출이 심한 폭력/선정물, 혐오발언, 개인정보 침해, 정치/종교 분쟁 유발 내용 등을 금지 목록으로 규정합니다. 서비스 이용약관이나 게시판 공지로 명시하고, 위반 시 통보 없이 삭제 및 이용 제한될 수 있음을 알립니다.
- **욕설 및 비속어 필터링**: 사용자 입력(게시물 제목, 설명)에 대해 **한글 욕설 필터**를 적용합니다. 예를 들어 badwords-ko 라이브러리를 사용하여 일반적인 욕설 단어를 치환하거나 입력 방지합니다. 또한 사용자가 우회 표현(특수문자 삽입, 초성만 입력 등)을 쓰는 경우를 대비해 **정규식 패턴** 검사를 추가로 실시합니다. (예: (ㅅ|ㅆ)Ws\*발 패턴 -> “시발” 탐지 등). 이러한 **키워드 필터링** 기법은 부적절 언어 사용을 자동으로 감지해 차단하는 1 차 수단이 됩니다 .
- **사용자 신고 시스템**: 사용자 **참여형** 관리를 위해 각 게시물에 “신고” 버튼을 제공합니다. 사용자가 부적절한 콘텐츠를 발견하면 사유와 함께 신고할 수 있고, 일정 횟수 신고가 누적되면 관리자 검토 전이라도 **임시 블라인드** 처리를 합니다. 신고 기능을 통해 커뮤니티 멤버들이 자율적으로 질서를 유지하게 유도합니다 . 신고된 콘텐츠와 사유는 별도 DB 테이블에 기록되어 관리자가 볼 수 있도록 대시보드에 표시합니다.
- **관리자 모니터링 도구**: 운영자는 전반적인 콘텐츠를 모니터링하고 조치하기 위한 **관리자 페이지**가 필요합니다. 초기에는 간단하게 Supabase 데이터베이스 콘솔을 활용하거나, 별도의 admin 전용 화면에서 게시물 리스트(신고순, 최신순 등 정렬 가능)와 **삭제/블라인드** 버튼을 제공하는 방식으로 구현합니다. 또한 자동 필터에 의해 차단된 욕설 댓글/게시물 로그를 저장하여 추후 오탐 여부를 검토하고 필터 리스트를 개선합니다.
- **레이트 리미팅**: 특정 사용자가 짧은 시간에 대량의 요청(예: 좋아요 폭탄 클릭, 연속 게시물 등록 등)을 보내는 것을 차단하기 위해 **Rate Limiting** 을 적용합니다. IP 기준으로 게시물 생성은 분당 제한을 두고, 좋아요 클릭 API 도 초당 1~2 회 이상 연속 호출 시 거부하도록 서버에서 처리합니다. 이러한 제한은 서버 부하를 막고 서비스 남용을 예방하는 효과가 있습니다. (추가적으로 **슬라이딩 윈도우 알고리즘** 등의 기법을 활용하면 더 유연한 제어가 가능하나, 우선은 간단한 고정 윈도우로 적용).

### 3-4. 성능 및 UX 최적화

- **초기 로딩 성능**: Next.js 의 정적 최적화 기능을 활용하여, 메인 페이지와 게시판 목록을 **\*\*Incremental Static Generation(ISR)\*\***으로 사전 생성할 수 있습니다. 예를 들어 메인 페이지의 최신/베스트 섹션은 1 분 간격으로 재생성하게 해 실시간성도 확보하고 초기 로딩도 빠르게 합니다. 게시판 페이지도 SSR 을 적용하되, 무한스크롤 부분은 클라이언트에서 추가 로드합니다.

- **이미지 최적화:** 업로드된 귀신 이미지들은 용량이 크지 않도록 **512px~1024px 범위**로 생성하고 포맷은 **WEBP** 로 저장합니다. Next.js `<Image>` 컴포넌트를 사용해 **lazy loading** 및 **정적 CDN** 기능을 얻습니다. 또한 썸네일용으로는 별도로 작게 리사이즈된 이미지를 사용하여 목록 로딩을 가볍게 합니다.
- **코드 스플리팅:** 앞서 언급한 미니게임, 이미지 비교 슬라이더 등의 기능은 필요한 순간에만 불러오도록 코드를 분리합니다. Next.js 의 `next/dynamic` 으로 미니게임 컴포넌트를 동적 임포트하고, `loading` 옵션으로 로딩 중 표시를 제어합니다. 이외에 사용되지 않는 라이브러리는 번들에서 제거하고, 공통 모듈은 캐시 활성화를 위해 `<script src>`로 CDN 에서 불러오는 것도 고려합니다.
- **메모리 관리:** 싱글 페이지 애플리케이션이므로 메모리 누수가 없도록 주의합니다. 특히 Canvas 를 사용하는 미니게임은 컴포넌트 언마운트 시 `cancelAnimationFrame` 과 이벤트 리스너 정리를 확실히 하여 메모리를 해제합니다. 또한 AI 생성 이미지 Blob 등을 다룰 때 메모리 사용에 유념합니다.
- **모니터링:** 성능 관련 이슈를 발견하기 위해 **웹 바이탈(Web Vitals)** 지표 및 사용자 행동 데이터를 추적합니다. Sentry 등의 APM 을 붙여 오류나 응답 지연이 발생한 API 를 로깅하고, Google Analytics 를 통해 페이지별 로딩속도나 이탈률을 분석합니다. 예를 들어, 이미지 생성 요청 실패율, 평균 생성 대기시간, 게시물 리스트 렌더링 시간 등을 주기적으로 체크합니다.

### 3-5. 배포 및 운영

- **CI/CD 파이프라인:** GitHub 저장소와 Vercel 을 연동하여 main 브랜치에 푸시될 때마다 자동으로 빌드 & 배포가 진행되도록 설정합니다. 개발 중에는 프리뷰 배포 링크를 활용해 기능을 확인하고, 이상 없을 시 본 배포로 승격합니다. 환경변수 관리(E.g., API 키)는 Vercel 의 환경변수 설정을 통해 주입하고, 민감 정보가 노출되지 않도록 주의합니다.
- **도메인 및 SSL:** 커스텀 도메인 `nosurfing.com` (예시)를 Vercel 프로젝트에 연결하고, Vercel 이 제공하는 **무료 SSL 인증서**로 HTTPS 를 적용합니다. 이를 통해 보안 및 SEO 에서 이점을 얻습니다. 또한 `robots.txt` 와 `sitemap.xml` 을 제공하여 검색엔진 크롤링에 대비합니다.
- **백엔드 인프라:** Supabase 등 외부 서비스와 연동된 부분은 각각 모니터링이 필요합니다. 예를 들어 Supabase 에서 제공하는 모니터링 툴이나 쿼리 성능 로그를 주기적으로 확인하고, 지표(활성 사용자 수, bandwidth 등)가 임계치 근접 시 사전 대응합니다.
- **장애 대응:** 장애 발생 시를 대비한 알림을 설정합니다. Vercel 함수 오류나 빌드 실패는 Vercel 대시보드 알림으로 파악하고, 심각한 오류(예: DB 연결 불가 등)는 Sentry 를 통해 슬랙/이메일 알림을 받습니다. 또한 백업

전략으로 데이터베이스는 주기적인 백업(export) 스크립트를 돌려 만일의 사태에 데이터를 복구할 수 있도록 합니다.

## 4. 결론 및 기대 효과

새롭게 추가되는 **공개 게시판 기능**과 **광고 수익화 전략**을 통해 ‘무서핑’ 서비스는 사용자 **참여도와 체류 시간**을 크게 높일 것으로 기대됩니다. 이제 이용자들은 단순히 콘텐츠를 소비하는 것을 넘어, 직접 AI로 생성한 ‘**존재**’ 콘텐츠를 **공유**하고 다른 사람들의 반응(좋아요)을 얻는 재미를 느낄 수 있습니다. 공포 콘셉트에 어울리는 **미니게임**과 독특한 UI 연출은 서비스의 차별점을 강화하여 **사용자 충성도**를 높일 것이며, 이는 곧 페이지뷰 증가로 이어져 광고 수익 창출에도 기여하게 됩니다.

또한, 철저한 콘텐츠 **모더레이션**과 **정책 준수**를 통해 부적절한 내용 노출을 최소화하고 Google AdSense 정책 위반 소지를 사전에 차단함으로써, **안정적인 광고 운영**이 가능해집니다. 백엔드 도입으로 서비스 구조가 전문화되어 데이터 영속성과 확장성이 확보되고, 향후 트래픽 증가에도 유연하게 대응할 수 있는 기반이 마련됩니다.

궁극적으로 이러한 기능 확장은 ‘무서핑’이 단순 **AI 캐릭터 생성 사이트**를 넘어 사용자들이 함께 즐기는 **인터랙티브 공포 커뮤니티**로 거듭나는 밑거름이 될 것입니다. 구현 과정에서 예상되는 과제들(예: AI 생성 비용, 익명 악용 등)은 본 문서의 방안을 따라 관리하고, 실제 운영 데이터를 보며 지속 개선해 나간다면, ‘무서핑’은 니치한테나에서도 **독창적이고 매력적인 서비스**로 자리매김할 것으로 예상합니다.

**참고 자료:** Supabase 공식 문서 (Next.js 연동 가이드), Bevy 블로그 (온라인 커뮤니티 모더레이션 가이드), Google 지원 문서 (폭력적/충격적 콘텐츠에 대한 정책), Reddit 사용자 경험 (Replicate API 비용 언급).